United States Patent [19]

Holt

[11] **4,130,862**

[45] **Dec. 19, 1978**

[54] **DC POWER SUPPLY**

[75] Inventor: **Frederick R. Holt, Saratoga, Calif.**

[73] Assignee: **Apple Computer, Inc., Cupertino, Calif.**

[21] Appl. No.: **874,124**

[22] Filed: **Feb. 1, 1978**

[51] **Int. Cl.^{2}** **H02M 3/335**
[52] **U.S. Cl.** **363/49; 363/21**
[58] **Field of Search** 363/18-21, 363/30, 49

[56] **References Cited**

U.S. PATENT DOCUMENTS

2,920,259	1/1960	Light	363/18
3,504,263	3/1970	Schaefer	363/49 X
3,523,235	8/1970	Schaefer	363/49 X
4,063,307	12/1977	Stephens	363/49 X

OTHER PUBLICATIONS

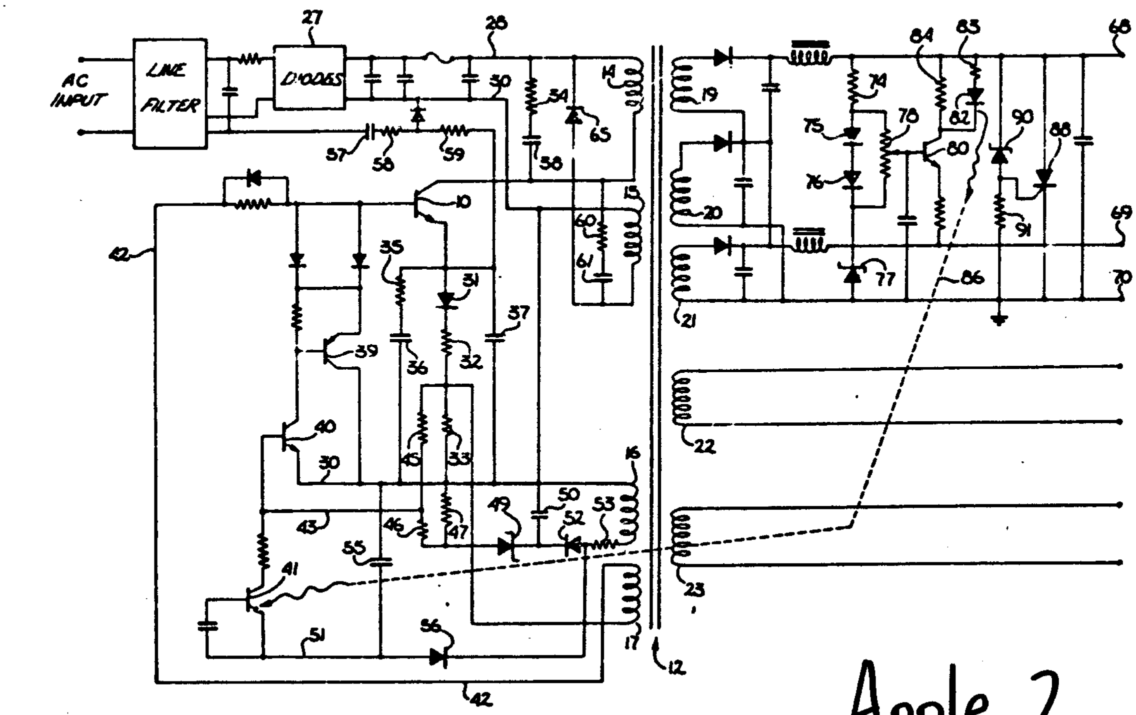
IBM Technical Disclosure Bulletin, "Transistor Switching Regulator Start Circuit", W. A. Moorman, vol. 13, No. 9, Feb. 1971, p. 2763.

Primary Examiner—William M. Shoop
Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman

[57] **ABSTRACT**

A direct current (DC) power supply of the single-ended flyback type particularly suited for providing power for integrated circuits is described. The power supply is self-exciting and thus does not employ an auxiliary drive or oscillator. The starting/restarting circuitry provides protection against faults. Because of this fault protection, a relatively simple over-voltage circuit is employed at the output of the supply. An additional primary winding is used to provide protection for no-load conditions.

11 Claims, 1 Drawing Figure



Apple 2

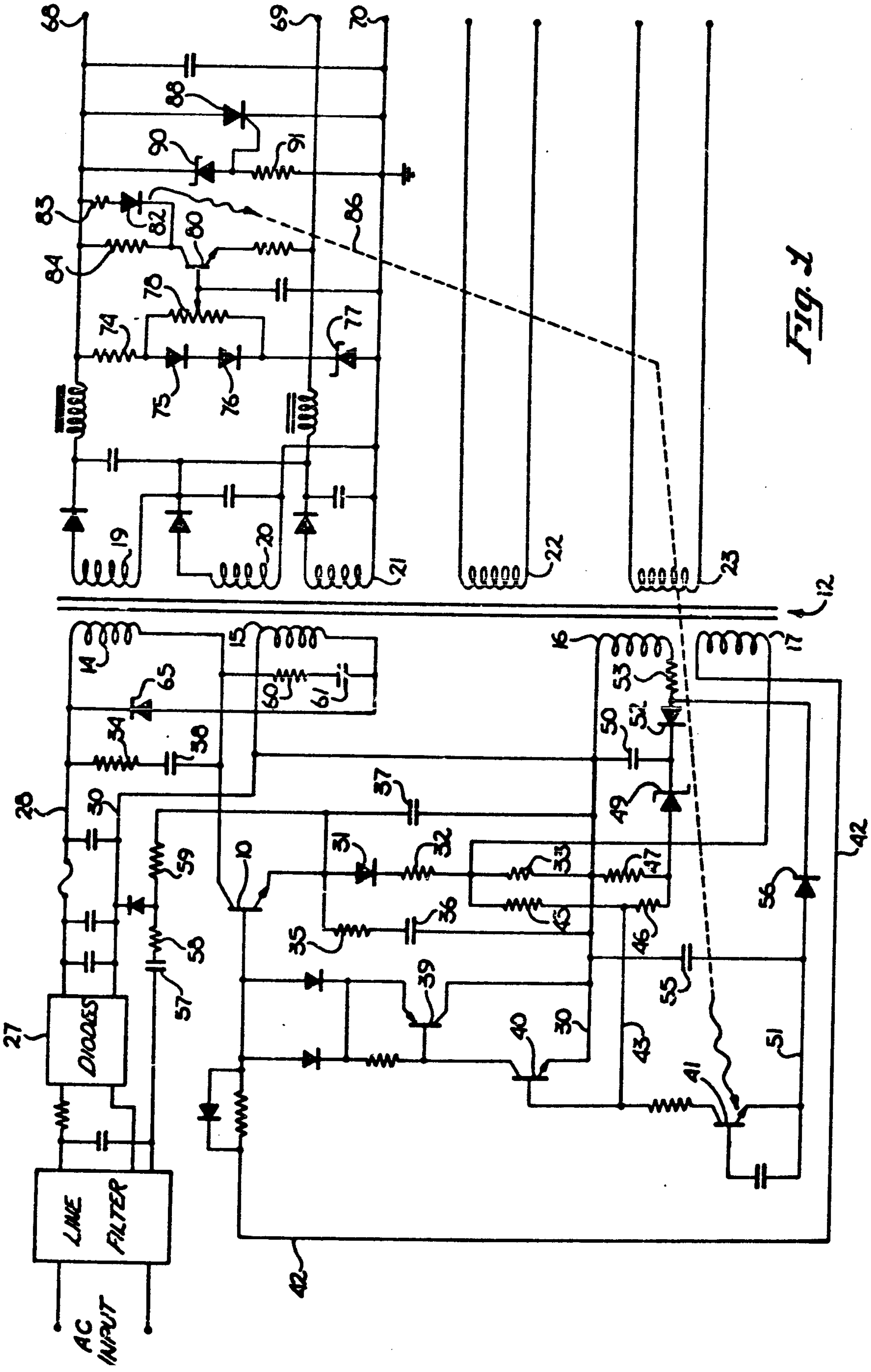

68
69
70
88
90
91
83
82
80
86
84
78
74
77
75
76
19
20
21
22
23
12
14
15
65
60
61
34
38
28
30
37
16
53
50
52
49
42
31
32
10
33
47
59
45
58
35
36
46
56
55
57
27
DIODES
39
40
38
43
41
51
LINE FILTER
AC INPUT
42

Fig. 1

DC POWER SUPPLY

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention relates to the field of DC power supplies.

2. Prior Art

DC power supplies which employ transformers to convert a first DC potential to a second DC potential are well-known. In some cases, these supplies are single-ended, that is, magnetic flux is induced in only one direction within the transformer. Most often these power supplies are of the "flyback" type, that is, power is transferred after conduction has ceased in the primary winding. Regulation of the output voltage is accomplished by controlling the amount of energy stored in the magnetic field of the transformer. This is generally done by controlling the peak current in the primary winding through a power transistor.

For the self-exciting type of power supply, an auxiliary drive or oscillator is often employed to initiate oscillations. As will be seen, the starting/restarting circuitry employed with the present invention consists substantially of passive circuit components which are inherently reliable. This circuitry also provides excellent fault protection.

In flyback systems employing transformers, a problem has existed in dissipating all the energy stored in the magnetic field of the transformer. Because of stray inductance and capacitance, and because complete magnetic coupling does not occur between the primary and secondary windings, all the energy introduced in the magnetic field from the primary winding is not coupled to the secondary winding. Typically, some of the energy stored in the field is dissipated in the power transistor which controls the current in the primary drive winding and in damping diode networks, etc. This subjects these components to severe stress in some cases. Moreover, when a fault condition occurs such as an open secondary, or a no-load condition, all of the energy stored in the magnetic field must be dissipated on the primary side. As will be seen with the present invention, an additional primary winding is employed which in effect returns a substantial amount of the energy in the magnetic field which is not transferred to the secondary winding, back into the primary power supply.

SUMMARY OF THE INVENTION

A direct current power supply of the single-ended flyback type is described. The power supply includes a transformer which has at least one primary winding and at least one secondary winding. The primary winding is coupled between a source of direct current potential and the collector of a power transistor. Starting means are employed to initiate oscillations in the power supply. The starting means are coupled to the emitter of the power transistor and initiate oscillations by causing a relatively low current flow through the emitter of the power transistor. In the presently preferred embodiment, the starting means comprises a series connected capacitor and resistor which are coupled to receive the negative portion of the AC line supply. When a fault occurs which causes the oscillations in the power supply to cease, the starting circuit attempts to restart the oscillations. Even if the fault remains, attempts to restart the oscillations do not harm the supply since the emitter current drawn by the starting means is low.

BRIEF DESCRIPTION OF THE DRAWING

The FIGURE is an electrical schematic of the presently preferred embodiment of the invented DC power supply.

DETAILED DESCRIPTION OF THE INVENTION

A direct current (DC) power supply which is particularly suited for providing power for integrated circuits is described. In the following description, numerous well-known concepts associated with DC power supplies of the type described have not been set forth in detail in order not to obscure the present invention in unnecessary detail. In other instances, specific details are set forth in order to provide a thorough understanding of the present invention. However, it will be obvious to one skilled in the art that the invention may be practiced without employing these specific details.

The described DC power supply is of the general type which converts energy from a first (primary) DC power supply into magnetic energy, and then converts this magnetic energy into a second (output) DC potential. This "through convertion type" power supply, in the presently preferred embodiment, employs a double wound transformer with the energy transferred to the secondary windings during the flyback cycle, that is, when conduction has ceased in the primary drive winding. The output DC potential is regulated by controlling the peak current in the primary drive winding.

Referring now to the FIGURE, the transformer **12** includes a plurality of primary windings **14, 15, 16** and **17**, and a plurality of secondary windings **19, 20, 21, 22** and **23**. The primary drive winding **14** transfers energy from the primary DC supply into the magnetic energy of the transformer. The parasitic winding **15** has the same number of turns as the winding **14** for reasons which will be described in greater detail. The winding **16** is employed to provide a control potential for controlling the current through the drive winding **14**. The feedback winding **17** provides a positive feedback signal for the power transistor **10**. The secondary windings **19, 20, 21, 22** and **23**, include rectification means which assure that current flows in the secondary windings only during the flyback period to provide the DC outputs. In the presently preferred embodiment, a gapped transformer is employed having an air gap of approximately 0.0028 inches. With this gap, substantially all of the energy associated with the magnetic field is stored in the air gap rather than in the ferromagnetic core of the transformer.

The primary DC supply is derived from an alternating current (AC) line source. This source is coupled through a line filter to a diode bridge **27**. The output of this diode bridge provides a positive DC potential on line **28**. This DC potential is coupled to one lead of the primary drive winding **14**. The other lead of this winding is coupled to the collector terminal of the transistor **10**. As will be described in greater detail, the transistor **10** is used to control the flow of current through the drive winding **14**. The line **28** is also coupled to the collector terminal of the transistor **10** through the resistor **34** and capacitor **38**.

The parasitic winding **15** which in one embodiment may be bifilar with the primary winding **14** has one of its leads coupled through diode **65** to line **28**. This lead is also coupled to the collector terminal of transistor **10** through the capacitor 61 and resistor 60. The other lead of the winding 15 is coupled to the ground node 30.

A starting circuit is employed to initiate oscillations for the power supply. (Oscillations refer to the repeated building up and decaying of the magnetic energy in the transformer 12 and the related currents). This starting circuit includes a resistor 35 of relatively high resistance which is coupled in series with capacitor 36 between the emitter terminal of transistor 10 and the ground node 30. The capacitor 36 is charged, as will be described, by the negative portion of the AC supply through a charging circuit. Coupling to the AC supply is obtained through capacitor 57 and resistor 58. One terminal of resistor 58 is coupled through the diode 62 to the ground node. Resistor 58 is also coupled to the emitter terminal of transistor 10 through the resistor 59. The emitter terminal of transistor 10 is coupled to the ground node through a relatively small capacitor 37 which, as will be explained, provides high frequency bypass. The main emitter current path during normal (heavy) oscillations includes the diode 31, the current limiting resistor 32, and resistor 33, all of which are coupled in series between the emitter terminal of the transistor 10 and the ground node 30.

Regulation of the DC output potentials is obtained by controlling the turnoff of transistor 10 through an active turnoff circuit which includes transistors 39, 40 and 41. Positive feedback from the winding 17 is coupled through line 42 and the parallel combination of a resistor and diode to the base terminal of the transistor 10. The base terminal of transistor 10 is coupled through a diode to the emitter terminal of the transistor 39. This emitter terminal of transistor 10 is also coupled through another diode and a resistor to the collector terminal of the transistor 40. The emitter terminal of transistor 40 and the collector terminal of transistor 39 are coupled to the ground node 30.

The base terminal of the transistor 40 is coupled to line 43. This line is coupled to the collector terminal of a light-sensitive transistor 41 through a resistor. The emitter terminal of transistor 41 is coupled to line 51, also the base terminal of this transistor is coupled to line 51 through a capacitor. The ground node 30 is coupled to line 51 through a capacitor 55. Line 51 is coupled to one lead of the primary winding 16 through a diode 56 and a resistor 53. The common junction between the diode 56 and resistor 53 is coupled through a diode 52 to one terminal of a capacitor 50. The other terminal of this capacitor and the other lead of the winding 16 are coupled to the ground node 30. The capacitor 50 is coupled through the Zener diode 49 to resistors 46 and 47. The resistor 47 is coupled to the ground node 30 while the resistor 46 is coupled to line 43. Line 43 is also coupled through resistor 45 to the junction between resistors 32 and 33; this junction is common with one lead of the feedback winding 17.

In the presently preferred embodiment, the power supply provides +12 volts on line 68 and +5 volts on line 69. The windings 22 and 23 are coupled to rectification means to provide output potentials of −5 volts and −12 volts. These rectification means are well-known means and are not illustrated in the FIGURE.

The windings 19, 20 and 21 are coupled through diodes to provide a positive potential on the lines 68 and 69 with reference to the ground line 70. Capacitors and inductors are employed in a well-known manner to provide filtering of these DC potentials as shown in the drawing.

A transistor 80 which is employed as a comparator, as will be explained, has its collector terminal coupled to line 68 through resistor 84 and through the series combination of resistor 83 and light-emitting diode 82. The light-emitting diode 82 is optically coupled to the light-sensitive transistor 41 as indicated by the path 86. The emitter terminal of the transistor 80 is coupled through a resistor to line 69. The base terminal of this transistor is coupled to the potentiometer 78. This potentiometer is coupled across the diodes 75 and 76. Diodes 75 and 76 are coupled to line 68 through the resistor 74 and to ground through the Zener diode 77. The Zener diode 77 provides a reference potential for the base of transistor 80. This Zener diode is temperature compensated by the diodes 75 and 76.

Over-voltage protection is provided through the silicon controlled rectivier (SCR) 88. This SCR is coupled between the lines 68 and 70. A triggering potential for the gate terminal of this device is provided by the Zener diode 90, which diode is coupled in series with resistor 91 between line 68 and ground. As is apparent, when the potential on line 68 exceeds a predetermined value (over-voltage) the Zener diode 90 conducts, thereby triggering the SCR 88. When the SCR 88 is triggered, line 68 is directly coupled to line 70 thereby shorting the over-voltage condition.

As mentioned, the power supply is self-exciting, thus it requires some means for initiating oscillations. A portion of the AC signal is coupled through the capacitor 57 to the emitter terminal of transistor 10. Because of the diode 62 and the resistors 58 and 57, only a portion of the negative potential is coupled to the emitter terminal. The diode 31 prevents this negative potential from being coupled to the ground node 30. As negative charge accumulates on the capacitor 36 it eventually lowers the emitter potential to approximately −0.6 volts at which time approximately 15 mA of emitter current flows through the emitter of the transistor 10. The transistor 10, assuming there are no faults, has substantial power gain. (The DC potential for the presently preferred embodiment on line 28 is approximately 140 to 200 volts). The positive feedback to the emitter of transistor 10 from the winding 17 along with the fact that high frequency signals are bypassed through the capacitor 37 causes rapid regeneration oscillations to start. During these oscillations current flows through the winding 14, through transistor 10 to the ground node 30 through the diode 31 and resistors 32 and 33. These oscillations once initiated are sustained through known phenomena. Such oscillations as is well-known rely upon stray capacitance and inductance, and upon the ringing of the underdamped system to reinitiate conduction in transistor 10 after the flyback portion of the cycle. During these oscillations, the capacitors 36 and 37 play substantially no part in the operation of the circuit, and in fact, the emitter terminal of transistor 10 rises to a potential of approximately 2.4 volts in the presently preferred embodiment.

Once heavy oscillations are underway and power is being transferred to the secondary windings, the turn-off point of transistor 10 is controlled by transistors 39 and 40. These transistors shunt base current from transistor 10 and act as an active turn-off circuit. Transistors 39 and 40 are coupled to sense both the AC line potential and the DC output potential. Local loop regulation is provided by the winding 16 and the potential which is developed across capacitor 50. Conditions such as heavy loads affect the potential developed across the capacitor 50. The potential across this capacitor since it is coupled to the base of transistor 40 partly controls the turn-off point of the transistor 10.

On the output side of the power supply, the transistor 80 compares the reference potential which is developed by Zener diode 77 with its emitter potential. The emitter potential is a function of the output potential on line 69. The results of this comparison determine the amount of current which flows through the light-emitting diode 82. The amount of current through this diode regulates the characteristics of the transistor 41. This transistor controls the turn-off point for transistor 10 since transistor 41 is coupled to the base of transistor 10 through transistors 39 and 40.

By way of example, if the potential on line 69 drops it will cause more current to flow through the light-emitting diode 82. The additional light from this diode causes the transistor 41 to become more conductive. This greater conductance of transistor 41 lowers the potential applied to the base of transistor 40. This prevents transistor 40 from conducting as readily as it might otherwise conduct, and hence less base current for transistor 10 is shunted through this transistor. Thus more positive feedback through line 42 reaches the transistor 10, increasing the maximum current which flows through the drive coil 14. This additional current results in more energy transfer to the secondary windings, thereby increasing the potential on line 69. The other DC output potentials follow this regulation loop.

Assume for sake of discussion that a fault condition occurs such as a short at the output of the power supply or a short caused by the conduction of the SCR 88. This fault substantially reduces the power gain associated with transistor 10 and causes the heavy oscillations to cease. The current through the current limiting resistor 32 also ceases. This allows the capacitor 30 to slowly become negatively charged again. The time constant associated with this charging is made relatively long to give time for fault correction. As the potential on the emitter of transistor 10 becomes negative, the relatively small emitter current is again drawn and the circuit attempts to oscillate. If the fault is removed, heavy oscillations occur and power is transferred to the secondary windings. On the other hand, if the fault remains the circuit nonetheless attempts to restart. These continued attempts to restart do not damage the circuit since the emitter current is relatively low (12 ma for the presently preferred embodiment). Thus, the restarting circuitry provides protection against faults since unsuccessful, continued attempts to restart do not damage the supply.

As previously mentioned, one inherent problem in flyback type systems in that all the energy stored in the magnetic field is not linked to the secondary windings, and thus a portion of this energy must be dissipated on the primary side of the supply, particularly in the power transistor and other circuit components. This problem is greatly aggravated during a no-load condition when all the power stored in the transformer must be dissipated on the primary side. Because of the winding 15 and its interconnection with the primary DC supply, a substantial portion of this non-transferred energy is returned to the primary D.C. supply. Assume for purposes of explanation that the secondary windings are open. When condition ceases through transistor 10, a substantial amount of energy is stored within the magnetic field of the transformer. At the moment that conduction ceases, the potential on the collector terminal of transistor 10 begins to rise sharply. At the same time, the potential on the ungrounded lead of the winding 15 rises in the positive sense. This potential is equal to the potential on the collector terminal of the transistor less the DC component on line 28. When the potential on the collector of transistor 10 reaches twice the potential of the DC supply, the potential on the ungrounded lead of winding 15 approximately equals the potential on line 28 causing diode 65 to conduct. The current through this diode is approximately equal to the peak current drawn by transistor 10 during the on-time of this transistor. This current returns substantially all the energy from the magnetic field back to the primary DC power supply. Were it not for the parasitic winding 15, this energy would be dissipated within the power supply and could destroy the transistor 10. In practice, not all the energy is returned to the DC power supply because of copper and iron losses, losses in the transistor and the fact that perfect linkage does not exist between the windings 14 and 15. Also in practice, the parasitic winding 16 is more economical to fabricate when wound on a separate layer rather than as a bifilar winding. This, of course, increases the leakage inductance. To compensate for this, the two windings are coupled together through the capacitor 61 and resistor 60. It is estimated that in the presently preferred embodiment, approximately 1/10 of the energy that would be otherwise returned to the primary DC power supply is lost through resistors 34 and 60.

Thus, a DC power supply has been disclosed of the general single-ended flyback type. The starting circuit for the power supply provides protection against faults. An additional primary winding which is coupled to the drive winding allows the return of non-transferred energy to the primary DC power supply. This feature reduces the stress on components during no-load conditions and the like.

I claim:

1. A direct current power supply comprising:

a transformer having at least one primary winding and one secondary winding, one lead of said primary winding for coupling to a source of direct current;

a transistor having a collector, base, and emitter terminal, said collector terminal coupled to the other lead of said primary winding;

starting means for initiating oscillations such that power may be transferred through said transformer from said primary winding to said secondary winding, said starting means comprising a first resistor and first capacitor coupled to the emitter terminal of said transistor and charging means for charging said first capacitor, said starting circuit for controlling the flow of emitter current so as to initiate said oscillations without damaging said transistor;

rectification means coupled to said secondary winding for providing an output direct current potential;

whereby oscillations are initiated in said direct current power supply without damage to said supply during a fault condition.

2. The power supply defined by claim 1 including a second resistor for limiting said emitter current.

3. The power supply defined by claim 2 including a diode coupled in series with said second resistor to allow the charging of said first capacitor without loss of current through said second resistor.

4. The power supply defined by claim 3 including second rectification means for coupling to an alternating current source and for providing said source of direct current for said primary winding.

5. The power supply defined by claim 4 wherein said charging means couples a portion of alternating current from said alternating current source to said first resistor for charging said first capacitor.

6. The power supply defined by claim 5 including a second capacitor coupled to said emitter terminal for providing a high frequency bypass to aid in the initiation of said oscillations.

7. The power supply defined by claim 1 wherein said starting circuit is effectively decoupled from said emitter terminals once oscillations are initiated.

8. A direct current power supply comprising:

a transformer having at least a first primary winding and a second primary winding and at least one secondary winding, one lead of said first primary winding coupled to a DC source;

a transistor having its collector terminal coupled to the other lead of said first primary winding for controlling the flow of current in said first primary winding;

starting means for initiating current flow in said transistor coupled to said transistor;

said second primary winding coupled to said first primary winding such that energy stored in the field of said transformer may flow to said DC course from said second primary winding;

a circuit means interconnecting said first and said second primary windings for compensating for leakage inductance;

whereby during a no load condition said energy stored in said transformer is returned to said DC source thereby protecting said power supply.

9. The power supply defined by claim 8 wherein said first and second primary windings are coupled through a diode.

10. The power supply defined by claim 8 wherein said starting means comprises a capacitor and resistor coupled to the emitter terminal of said transistor, said resistor coupled to receive a portion of an AC signal.

11. The power supply defined by claim 9, wherein said circuit means comprises a series coupled resistor and capacitor.

* * * * *

United States Patent [19]

Wozniak

[11] **4,136,359**

[45] **Jan. 23, 1979**

[54] **MICROCOMPUTER FOR USE WITH VIDEO DISPLAY**

[75] Inventor: **Stephen G. Wozniak**, Cupertino, Calif.

[73] Assignee: **Apple Computer, Inc.**, Cupertino, Calif.

[21] Appl. No.: **786,197**

[22] Filed: **Apr. 11, 1977**

[51] **Int. Cl.**^{2} **H04N 9/44**
[52] **U.S. Cl.** **358/17**
[58] **Field of Search** 358/17, 148, 150

[56] **References Cited**

U.S. PATENT DOCUMENTS

3,581,011	5/1971	Ward et al.	358/17

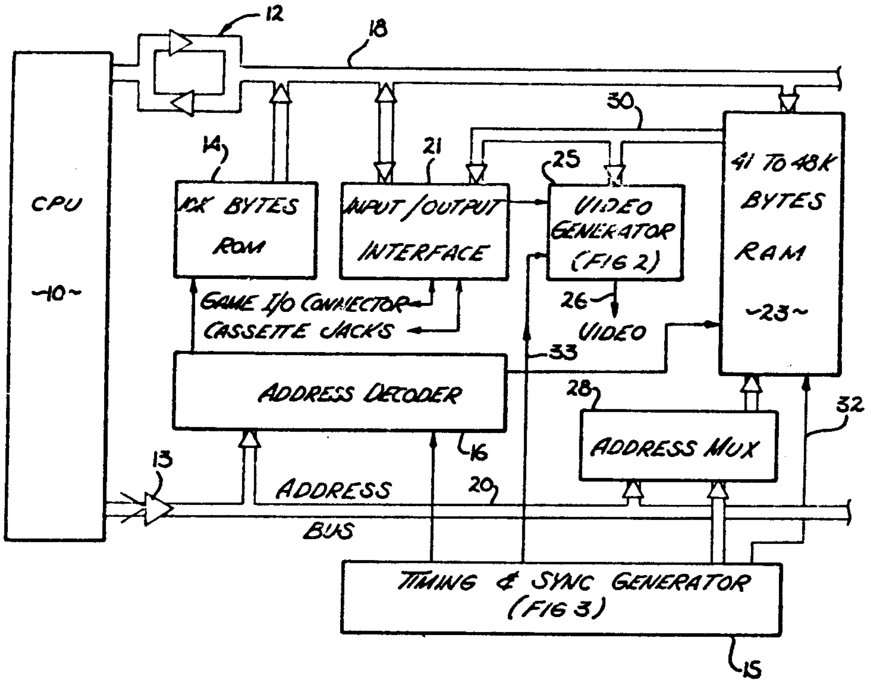
Primary Examiner—Richard Murray

Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman

[57] **ABSTRACT**

A microcomputer including a video generator and timing means which provides color and high resolution graphics on a standard, raster scanned, cathode ray tube is disclosed. A horizontal synchronization counter is synchronized at an odd-submultiple of the color subcarrier reference frequency. A "delayed" count is employed in the horizontal synchronization counter to compensate for color subcarrier phase reversals between lines for the non-interlaced fields. This permits vertically aligned color graphics without substantially altering the standard horizontal synchronization frequency. Video color signals are generated directly from digital signals by employing a recirculating shift register.

8 Claims, 4 Drawing Figures



Apple 2

U.S. Patent Jan. 23, 1979 Sheet 1 of 2 4,136,359

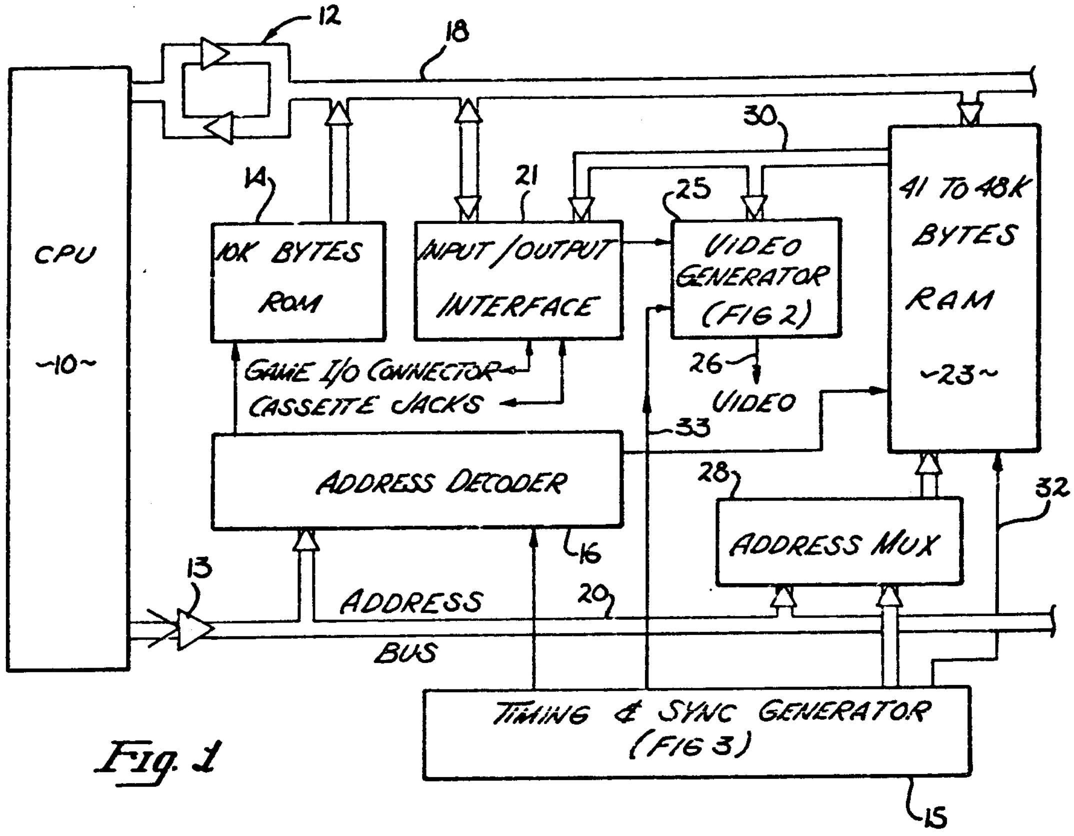


Fig. 1

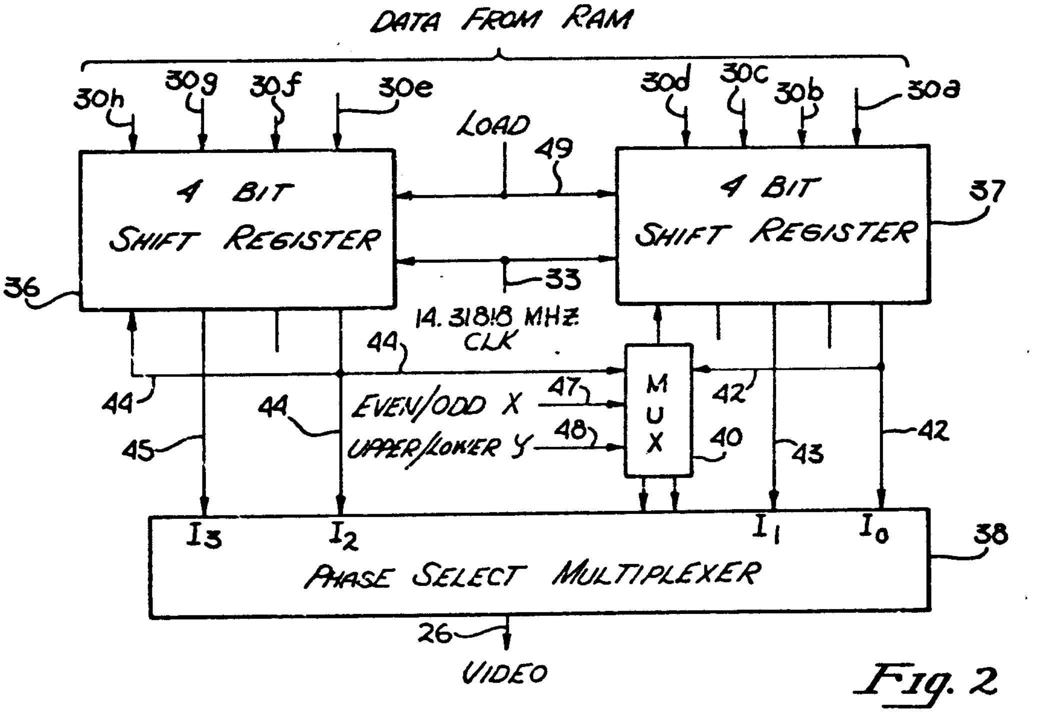


Fig. 2

"APPLE_PAT_4_136_359_02" 145 KB 2000-02-21 dpi: 300h x 300v pix: 1846h x 2896v

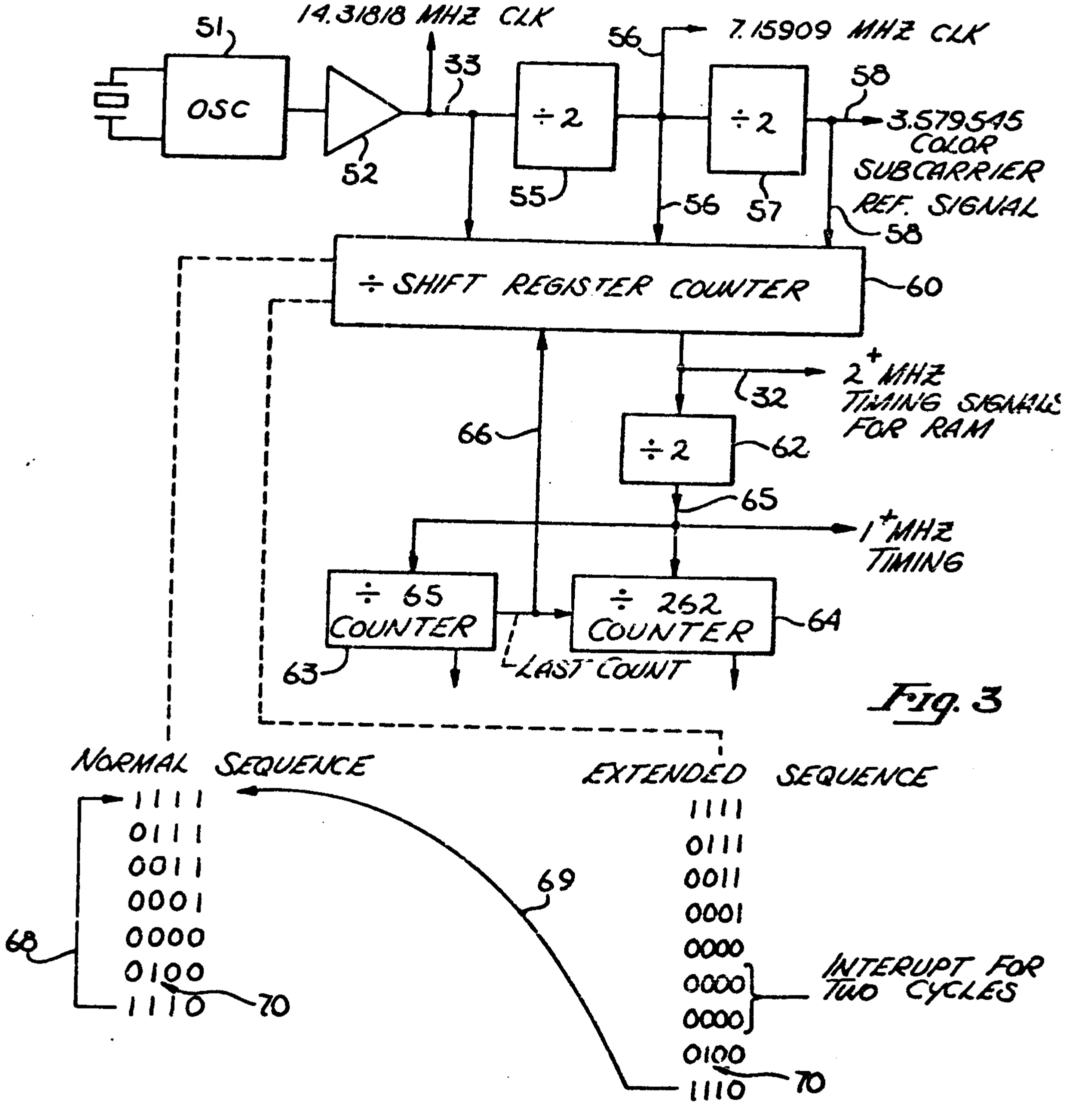

51
OSC
52
14.31818 MHZ CLK
33
÷2
55
56
7.15909 MHZ CLK
÷2
57
56
58
3.579545 COLOR SUBCARRIER REF. SIGNAL
58
÷ SHIFT REGISTER COUNTER
60
2+ MHZ TIMING SIGNALS FOR RAM
32
66
÷2
62
65
1+ MHZ TIMING
÷ 65 COUNTER
63
÷ 262 COUNTER
64
LAST COUNT
NORMAL SEQUENCE
1111
0111
0011
0001
0000
0100
1110
68
70
69
EXTENDED SEQUENCE
1111
0111
0011
0001
0000
0000
0000
0000
0100
1110
INTERUPT FOR TWO CYCLES
70

Fig. 3

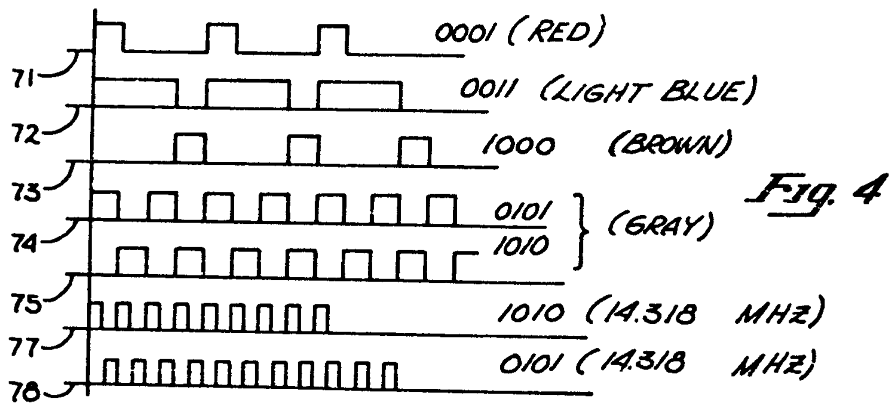

0001 (RED)
0011 (LIGHT BLUE)
1000 (BROWN)
0101
1010
(GRAY)
1010 (14.318 MHZ)
0101 (14.318 MHZ)
71
72
73
74
75
77
78

Fig. 4

MICROCOMPUTER FOR USE WITH VIDEO DISPLAY

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention is for the generation of signals for raster scanned video displays employing digital means.

2. Prior Art

With the reduced cost of large scale integrated circuits it has become possible to provide low-cost microcomputers suitable for home use. One such use which has flourished in recent years is the application of microcomputers in conjunction with video displays for games and graphic displays. Most often an ordinary television receiver is employed as the video display means. The standard, raster scanned, cathode ray tubes employed in these receivers and like displays, present unique problems in interfacing these displays with the digital information provided by the microcomputer.

In presenting color graphics it is, of course, desirable to provide high resolution lines and to avoid "ragged" lines. In a microcomputer controlled display, typically a single frequency reference source is employed to generate the color subcarrier reference signal of 3.579545Mhz and the horizontal and vertical synchronization signals. If the frequency of the horizontal synchronization signals is to remain close to its normal frequency (i.e. 15,750hz) the horizontal synchronization means must operate at an odd-submultiple of the color subcarrier frequency. When this occurs there is a phase reversal or phase shift of the color subcarrier reference signal when compared to color control signal between each of the lines of the display. This results in ragged vertical lines unless the color signals are changed for each line. One prior art solution to this problem has been to operate the horizontal synchronization counter at an even submultiple of the color subcarrier frequency (i.e. 15,980hz). This deviation from the standard horizontal synchronization frequency typically requires manual adjustment of the receiver and for some receivers horizontal synchronization may be more difficult to maintain.

As will be described with the invented microcomputer, the horizontal counter operates close to its standard frequency (15,734hz). Through use of a timing compensation means, counting in the horizontal synchronization counter is delayed to compensate for the fact that the counter operates at an odd-submultiple frequency of a color reference signal. In this manner, phase reversal of the color reference signal is eliminated and sharp graphic displays are provided without complex programming.

In many prior art microcomputer controlled displays, color information is stored as four digital bits which are used to designate green, red, blue, and high/low intensity. The color generation means generally includes a signal generator for generating the pure color signals (CW). These pure color signals are then gated and mixed in accordance with the binary state of the four bits to provide a color signal compatible with standard television receivers. Generation of the video color signal in this manner is complex and requires a substantial amount of circuitry.

The invented microprocessor includes a recirculating shift register which circulates four bits of information. In this manner video color signals are generated directly from digital information without the cumbersome generation techniques employed in the prior art.

SUMMARY OF THE INVENTION

A microprocessor for use with a video display is described. The microprocessor includes an improved timing apparatus which provides well-defined color graphics on a standard, raster scanned, cathode ray tube. A timing reference means is employed to provide a color reference signal for the video display. A horizontal synchronization means which is synchronized to the timing reference means provides horizontal synchronization signals for the display. These signals occur at a rate which is an odd-submultiple of the color reference signal frequency. The timing apparatus includes a compensation means which is coupled to both the timing reference means and the synchronization means for periodically adjusting the horizontal synchronization signals such that these signals remain in phase relationship with the color reference signal.

The microcomputer also includes a unique color signal generation means which uses a recirculating shift register. This register receives digital signals representative of color from memory and circulates this data at a predetermined rate. In this manner a color signal suitable for use with a video display is generated from the digital signals.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a general block diagram illustrating the invented microcomputer in its presently preferred embodiment.

FIG. 2 is a block diagram of the video generator employed in the microcomputer of FIG. 1.

FIG. 3 is a block diagram of the timing and synchronization generator employed in the computer of FIG. 1; and

FIG. 4 is a graph illustrating several waveforms generated by the video generator of FIG. 2.

DETAILED DESCRIPTION OF THE INVENTION:

A microcomputer is disclosed which is particularly suitable for controlling color graphics on a standard, raster scanned, cathode ray tube. The described microcomputer includes a video generator which generates color signals directly from digital information, and a timing means which provides well defined color graphics, particularly in the vertical direction, without complex programming.

In the following description, numerous well-known circuits are shown in block diagram form in order not to obscure the described inventive concepts in unnecessary detail. In other instances, very specific details such as frequencies, number of bits, specific codes, etc., are providing in order that these inventive concepts may be clearly understood. It will be apparent to one skilled in the art that the described inventive concepts may be employed without use of these specific details.

Referring now to FIG. 1, the microcomputer includes a central processing unit (CPU) or microprocessor 10. While any one of a plurality of commercially available microprocessors may be employed such as the M6800 or 8080, in the presently preferred embodiment, a commercially available microprocessor, Part No. 6502, is employed. CPU 10 communicates with the data bus 18 through a bidirectional tri-state buffer 12. The CPU 10 is also coupled to the address bus 20 through a tri-state buffer 13.

The microcomputer, in its presently preferred embodiment, includes two memories. The first is a 12K (bytes) read-only memory (ROM) 14 which is coupled to the data bus 18. This ROM may be a mask programmable memory, E PROM or other read-only memory. The primary data storage for the computer comprises the random-access memory 23. In the presently preferred embodiment, this memory may contain 4K to 48K (bytes) and comprises commercially available dynamic MOS memories. The RAM 23 is coupled to the input/output interface means 21 via bus 30, the data bus 18 and the video generator 25.

The timing signals for the microcomputer are provided by the timing and synchronization generator 15. The novel portions of this generator shall be described, in detail, in conjunction with FIG. 3. This generator provides timing signals for the microcomputer, and additionally, synchronization signals for the video display. Among the signals provided by the generator 15 are 2+Mhz timing signals on lines 32 for the RAMs 23 and a 14.31818Mhz signal on line 33 for the video generator 25. The timing and synchronization generator 15 also provides timing signals for the decoder 16 and for the address multiplexer 28.

The address decoder 16 receives address signals from the address bus 20 and decodes them in a well-known manner. The address decoder 16 is coupled to the ROM 14 and to the RAM 23. Address signals are also received from the bus 20 by the address multiplexer 28 which couples these signals to the RAM 23.

The input/output interface means 22 provides ports which allows the microprocessor to be electrically coupled to a cassette jack or to a connector used for receiving game input/output signals. Known buffers and timing means may be employed for this purpose.

The video generator 25 receives signals from the input/output interface means 21 and also from the RAM 23. This generator provides an output video signal on line 26. Video generator 25 shall be described, in detail, in conjunction with FIG. 2.

In the presently preferred embodiment, the entire microcomputer of FIG. 1 is fabricated on a single printed circuit board. This board includes connectors to allow the computer to be connected to a cassette playback means, or other devices. As will be appreciated, numerous well-known interconnections, driver means and other circuits employed in the microcomputer are not shown in FIG. 1. For a detailed description of circuits and interconnections which may be employed in the microprocessor of FIG. 1, including a transparent refresh cycle for the RAMs 23, see "*A CRT Terminal Using The M6800 Family*" by Roy & Morris, *Interface Age*, Volume 2, Issue 2, January 1977.

Referring now to FIG. 3, the timing and synchronization generator (timing means) includes a frequency reference source, crystal oscillator 51. The output of oscillator 51 is coupled to a buffer 52 which provides a 14.31818Mhz signal on line 33 for the presently preferred embodiment. This signal is coupled to the video generator of FIG. 2 as will be described, and is also coupled to the shift register counter 60 and the divider 55. The divider 55 divides the 14.31818Mhz signal by two, thereby providing a 7.15909Mhz signal on line 56. This signal is employed by the microprocessor as a timing signal, and additionally, is employed by the shift register counter 60 as a feedback synchronization signal. The signal on line 56 is further divided by two, by divider 57, to provide the standard color subcarrier reference signal of 3.579545Mhz on line 58. The signal on line 58 is used in an ordinary manner by the video display and also is used as a feedback synchronization signal by the shift register counter 60.

The 14.3Mhz signal on line 33 is divided by seven, by the shift register counter 60 to provide a 2+Mhz signal on line 32. This signal is used by the RAMs 23 of FIG. 1. This 2+Mhz signal is further divided by divider 62 (divided by two) to provide a 1+Mhz timing signal on line 65. This 1+Mhz signal in addition to being employed elsewhere in the microprocessor is used by counters 63 and 64.

The "divide-by-65" counter 63 is used to provide the horizontal synchronization signals for the non-interlaced display. When the maximum count is reached within the counter 63, a signal is provided on line 66 to shift register 60 and also to the vertical synchronization counter 64. The counter 64 is employed to divide this signal by 262 to provide vertical synchronization signals.

In the presently preferred embodiment, the display is divided into a 65×262 array. However, 25 of the 65 horizontal character positions are employed for blanking and 70 of the 262 lines are also employed for blanking.

It is apparent from FIG. 3 that the horizontal synchronization signals from counter 63 occur at a frequency of approximately 15,734hz. This is very close to the standard horizontal synchronization rate of 15,750hz. Each count of the counter 63 includes 3½ color cycle of the color subcarrier reference frequency; moreover, the total number of color cycles per line is a non-integer. As a result, the color subcarrier reference signal will be shifted 180° for each new line. Unless some corrective action is taken this will result in ragged vertical lines. As will now be described, compensation is provided by delaying the occurrence of the 1+Mhz timing signal once for each line by a period of time corresponding to ½ cycle of the 3.58Mz subcarrier reference signal.

As shown in FIG. 3, the normal counting sequency for the shift counter 60 includes seven states. When the last stage of the four stage counter contains a binary-zero, a binary-one is loaded into the second stage (position 70). The first and second stages receive the output of the second stage when the last stage contains a binary-zero. Thus, the states become 1110 after the next shift, and finally the states become 1111 as indicated by path 68.

Each time a signal occurs on line 66 (every 65 cycles of the 1+Mhz signal) the normal sequencing within the counter 60 is altered as shown by the extended sequence of FIG. 3. When a signal occurs on line 66 and when the count of 0000 is reached, the loading of the binary-one into the second stage (position 70) is delayed for two cycles of the 14.318Mhz clock. These two cycles correspond to 180° of the 3.58Mhz signal. After these two cycles, a binary-one is then loaded into the second stage, followed by the loading of binary-ones into the first and third stages. As indicated by path 69, a normal counting sequence then occurs. By extending the count within counter 60 as described, compensation occurs which provides vertical color alignment from line-to-line.

Referring now to FIG. 2, the video generator 25 of FIG. 1 includes two, four bit shift registers 36 and 37. Each of these four bit shift registers is coupled to receive four bits of data on lines 30 from the RAM 23.

The registers 36 and 37 receive a load signal on line 49 which causes the data on lines 30a through 30h to be shifted into the registers. The first stage of register 37 (I_0) is coupled to a multiplexer 38 by line 42. The third stage of register 37 (I_1) is also coupled to the multiplexer 38 by line 43. In a similar fashion, the first stage of the register 36 (I_2) is coupled by line 44 to the multiplexer 38, and the third stage of this register (I_3) is also coupled to the multiplexer 38 by line 45.

Line 44 is coupled to the fourth stage of register 36 in order that four bits of data within register 36 may be recirculated. (Registers 36 and 37 shift data from left to right, that is, toward their first stage). The line 42 may be selectively coupled to the fourth stage of register 37 through the multiplexer 40 in order that four bits of data within register 37 may be recirculated. Line 44 may be coupled through the multiplexer 40 to the fourth stage of the shift register 37. When this occurs, the shift registers 36 and 37 operate as a single eight bit shift register.

Control signals designated as even/odd X (line 47) and upper/lower Y (line 48) are used to control multiplexer 38. During the color graphics mode the registers 36 and 37 operate as separate registers and data is alternately selected for coupling to line 26 by multiplexer 38. The upper/lower Y signal, during the color graphics mode, allow selection of data from either register 36 or 37. The odd/even X signal then toggles the data from the selected register by alternating selecting I_0 or I_1 if register 37 is selected, or I_2 or I_3 if register 36 is selected.

During the color graphics mode as presently implemented, eight bits of color information are shifted (in parallel) into the registers 36 and 37 from the RAM 23 at a 1^+Mhz rate. This data is recirculated within registers 36 and 37 at a rate of 14.31818Mhz by the clocking signal received on line 33. The circulation of the data bit within the registers 36 and 37 at this rate provides signals having a 3.58Mhz component and as will be described, these signals may be readily employed for providing color signals for video display.

In the color graphics mode, as presently implemented, each of the display characters is divided into an upper and lower color rectangle. The RAM 23 provides the four bits of color data for the upper rectangles to registers 36 and for the lower rectangles to register 37. This color data for the presently preferred embodiment is coded as follows:

Red; 0001
Pink; 1011
Blue; 0010
Light Blue; 0111
Dark Green; 0100
Light Green; 1110
Brown; 1000
Yellow; 1101
Medium Violet; 0011
Medium Blue; 0110
Medium Green; 1100
Orange; 1001
White; 1111
Grey; 1010
Gray; 0101

When colors are coded in this manner and circulated at the rate of 14.318Mhz in the registers, video color signals compatible with standard television receivers are produced. The resultant signal for red is shown on line 71 of FIG. 4, light blue on line 72, brown on line 73 and gray on lines 74 and 75.

Briefly referring again to FIG. 3, each count of the horizontal synchronization counter 63 corresponds to 3½ cycles of the subcarrier reference signal. Thus, a 180° phase shift occurs from character-to-character with respect to the color subcarrier reference signal. This means that the color signals must be shifted by 180° by the generator of FIG. 2, or the coding for these signals must be alternated for odd and even horizontal character positions. In the presently preferred embodiment, a 180° phase shift for the color signals is obtained by toggling between the first or third stages of the selected registers. For example, assume that the lower portion of a character is being displayed and that the color information is thus contained within register 37. Further assume that this information is being circulated, that is, line 42 couples stage 4 to stage 1 through the multiplexer 40. For even horizontal character positions, as indicated by the signal on line 47, the phase select multiplexer 38 couples the I_0 signal to line 26. For the odd positions, a 180° phase shift is obtained by selecting the I_1 signal.

During a second mode of operation the generator of FIG. 2 is used for providing high resolution graphics. In this case, eight bits of information are provided by the RAM 23 to the registers 36 and 37. For this high resolution mode line 42 is coupled to the video line 26 and the eight bits of data from RAM 23 are serially coupled to the video line 26 at the 14.318Mhz rate. The multiplexer 40 couples line 44 to the fourth stage of register 37 to provide a single eight bit shift register. The resultant signals are shown on lines 77 and 78 of FIG. 4. The signals on lines 77 and 78 provide either a green or violet display. In the presently preferred embodiment, data changes are employed to obtain the compensation provided by the multiplexer 38 during the color graphics mode.

Thus, a microcomputer has been disclosed which is particularly suitable for controlling a color video display. The unique timing means provides well defined vertical color lines without complicated programming changes while allowing the generation of horizontal synchronization signals at close to the standard rate. The unique video generator allows the generation of color signals directly from digital signals without the complex circuitry often employed in the prior art.

I claim:

1. In a microcomputer for use with a video display an improved timing apparatus comprising:

a timing reference means for providing a color reference signal for said video display;

a horizontal synchronization means for providing horizontal synchronization signals for said display, said synchronization means coupled to said timing reference means for synchronization with said reference means such that said synchronization signals occur at an odd-submultiple of said color reference signal;

timing compensation means coupled to said timing reference means and said horizontal synchronization means for adjusting said horizontal synchronization signals such that said horizontal synchronization signals are in phase relationship with said color reference signal;

whereby the color graphics on a raster scanned cathode ray tube are sharply defined in the vertical direction.

2. The apparatus defined by claim 1 wherein said horizontal synchronization means comprises a digital counter.

3. The apparatus defined by claim 2 wherein said timing compensation means periodically delays counting in said counter.

4. The apparatus defined by claim 3 wherein said color reference signal is an approximately 3.58Mhz signal and said horizontal synchronization signals occur at a frequency of approximately 15,734Hz.

5. In a microcomputer for use with a video display an improved timing apparatus comprising:

a horizontal synchronization counter;

a timing reference means for synchronizing said counter and for providing a color reference signal, said reference signal frequency being an odd-multiple greater than the rate at which counting occurs in said counter;

delay means for delaying counting in said counter when the count in said counter reaches a predetermined count, said delay means coupled to said horizontal synchronization counter and said timing reference means;

whereby well-defined color graphics may be readily stored and displayed on said video display.

6. The apparatus defined by claim 5 including a digital divider for dividing by an odd-integer coupled between said reference means and said counter.

7. The apparatus defined by claim 6 wherein said digital divider includes a shift register counter and wherein the loading of digital signals in said register counter is interrupted when said predetermined count is reached.

8. The apparatus defined by claim 7 wherein said color reference signal is an approximately 3.58Mhz signal and said predetermined count is reached at a frequency of approximately 15,734Hz.

* * * * *

United States Patent [19]

Wozniak

[11] **4,210,959**

[45] **Jul. 1, 1980**

[54] **CONTROLLER FOR MAGNETIC DISC, RECORDER, OR THE LIKE**

[75] Inventor: **Stephen G. Wozniak, San Jose, Calif.**

[73] Assignee: **Apple Computer, Inc., Cupertino, Calif.**

[21] Appl. No.: **904,420**

[22] Filed: **May 10, 1978**

[51] **Int. Cl.²** **G06F 13/08; G06F 3/06**

[52] **U.S. Cl.** **364/200;** 360/78

[58] **Field of Search** ... 364/200 MS File, 900 MS File,
364/200, 900; 360/69, 71, 78, 135

[56] **References Cited**

U.S. PATENT DOCUMENTS

3,668,647	6/1972	Evangelista	364/200
3,909,800	9/1975	Recks et al.	364/200
3,987,419	10/1976	Morrill et al.	364/200
3,990,055	11/1976	Henderson et al.	364/200
4,080,651	3/1978	Cronshaw et al.	364/200
4,096,579	6/1978	Black et al.	364/900
4,100,601	7/1978	Kaufman et al.	364/200
4,148,098	4/1979	McCreight et al.	364/200

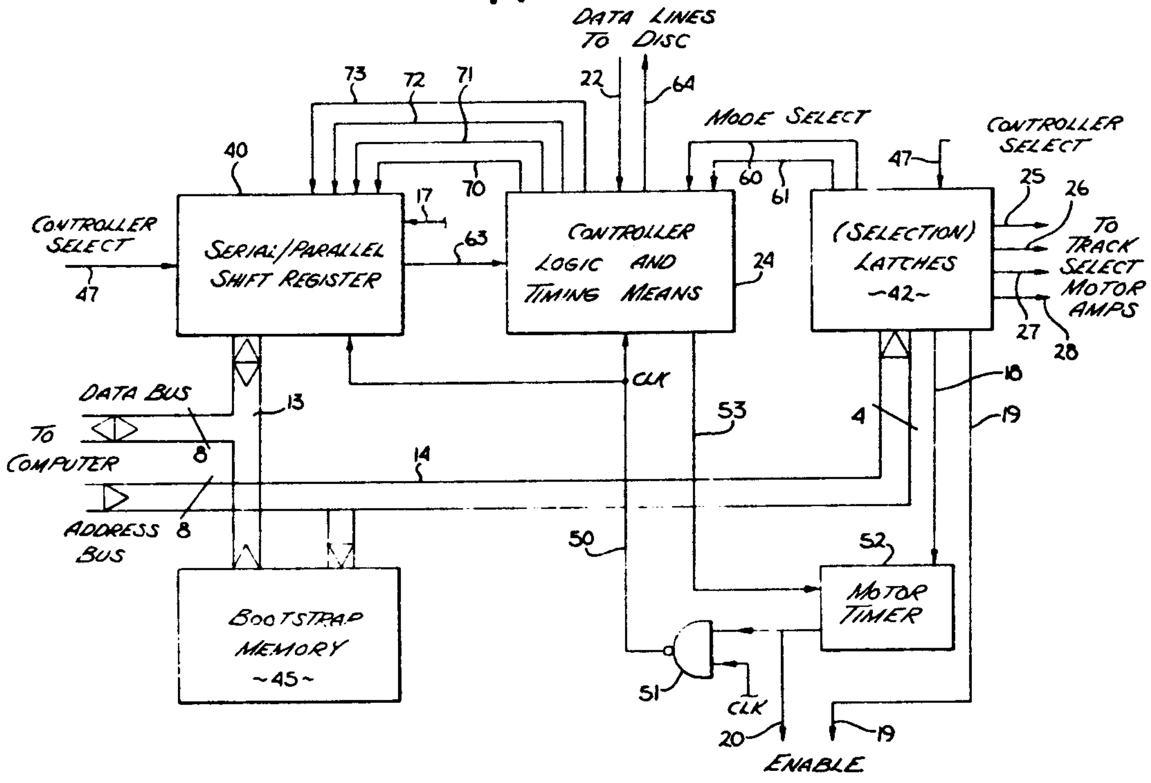
Primary Examiner—Mark E. Nusbaum
Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman

[57] **ABSTRACT**

A controller for interfacing between a digital computer and a magnetic disc recorder, such as a "floppy" disc, or other recorder or memory is disclosed. The controller, which permits accessing of the disc with minimum control by the computer, is realized with relatively few components and is particularly suited for interfacing between a microcomputer and a "minifloppy". Track selection with a computed velocity profile is employed. Synchronization with a soft sectored disc is achieved without additional hardware.

14 Claims, 6 Drawing Figures

Apple 2



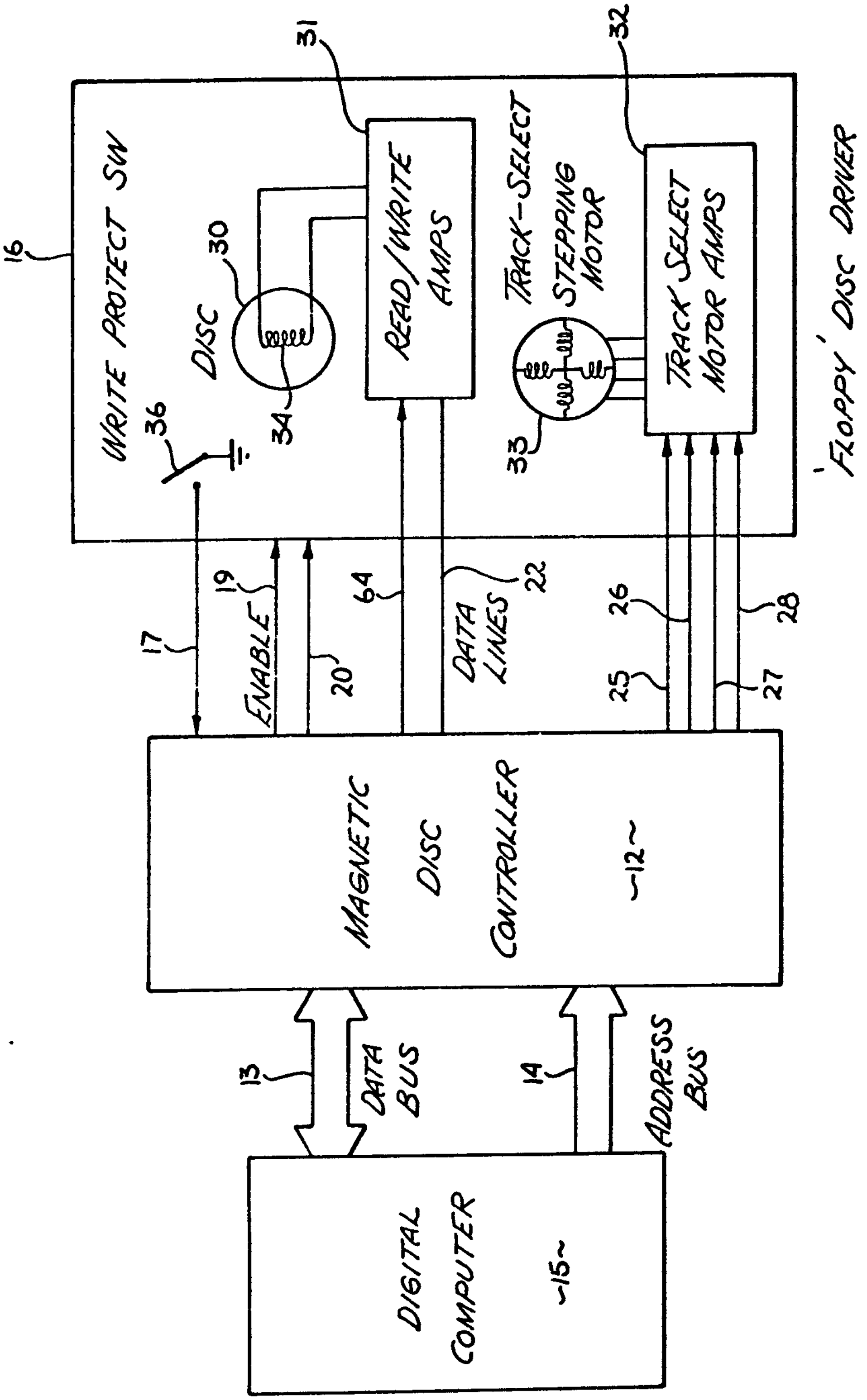

16
WRITE PROTECT SW
36
DISC
30
34
31
READ/WRITE AMPS
TRACK-SELECT STEPPING MOTOR
33
32
TRACK SELECT MOTOR AMPS
'FLOPPY' DISC DRIVER
17
ENABLE
19
20
64
DATA LINES
22
25
26
27
28
MAGNETIC DISC CONTROLLER
~12~
13
DATA BUS
14
ADDRESS BUS
DIGITAL COMPUTER
~15~

Fig. 1

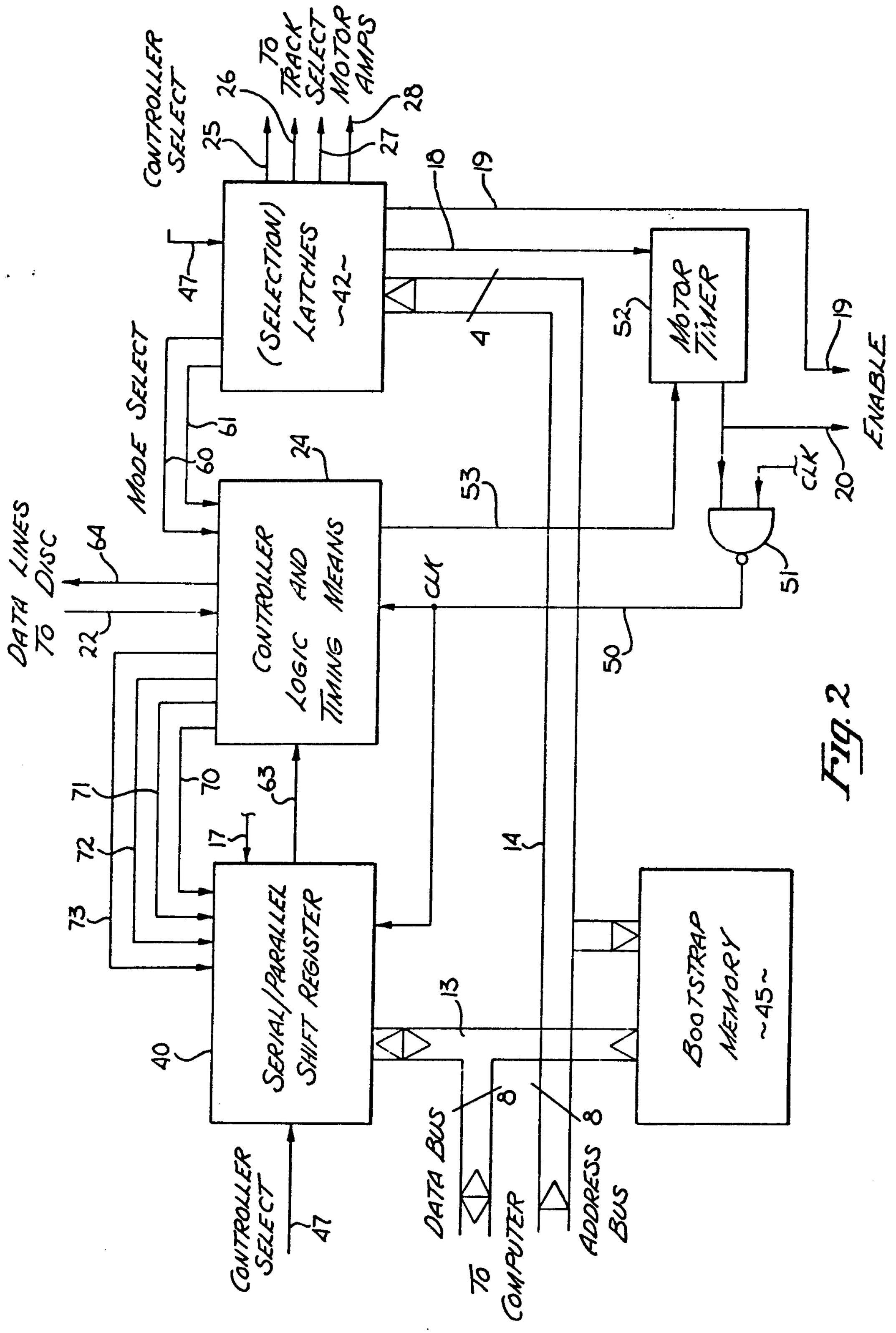

CONTROLLER SELECT
25
26
TO TRACK SELECT MOTOR AMPS
27
28
18
19
(SELECTION) LATCHES ~42~
47
4
52
MOTOR TIMER
19
ENABLE
MODE SELECT
60
61
24
CONTROLLER LOGIC AND TIMING MEANS
53
CLK
20
51
DATA LINES TO DISC
64
22
CLK
50
71
70
63
72
17
73
14
SERIAL/PARALLEL SHIFT REGISTER
40
13
BOOTSTRAP MEMORY ~45~
DATA BUS
8
8
TO COMPUTER
ADDRESS BUS
CONTROLLER SELECT
47

Fig. 2

U.S. Patent Jul. 1, 1980 Sheet 3 of 4 4,210,959

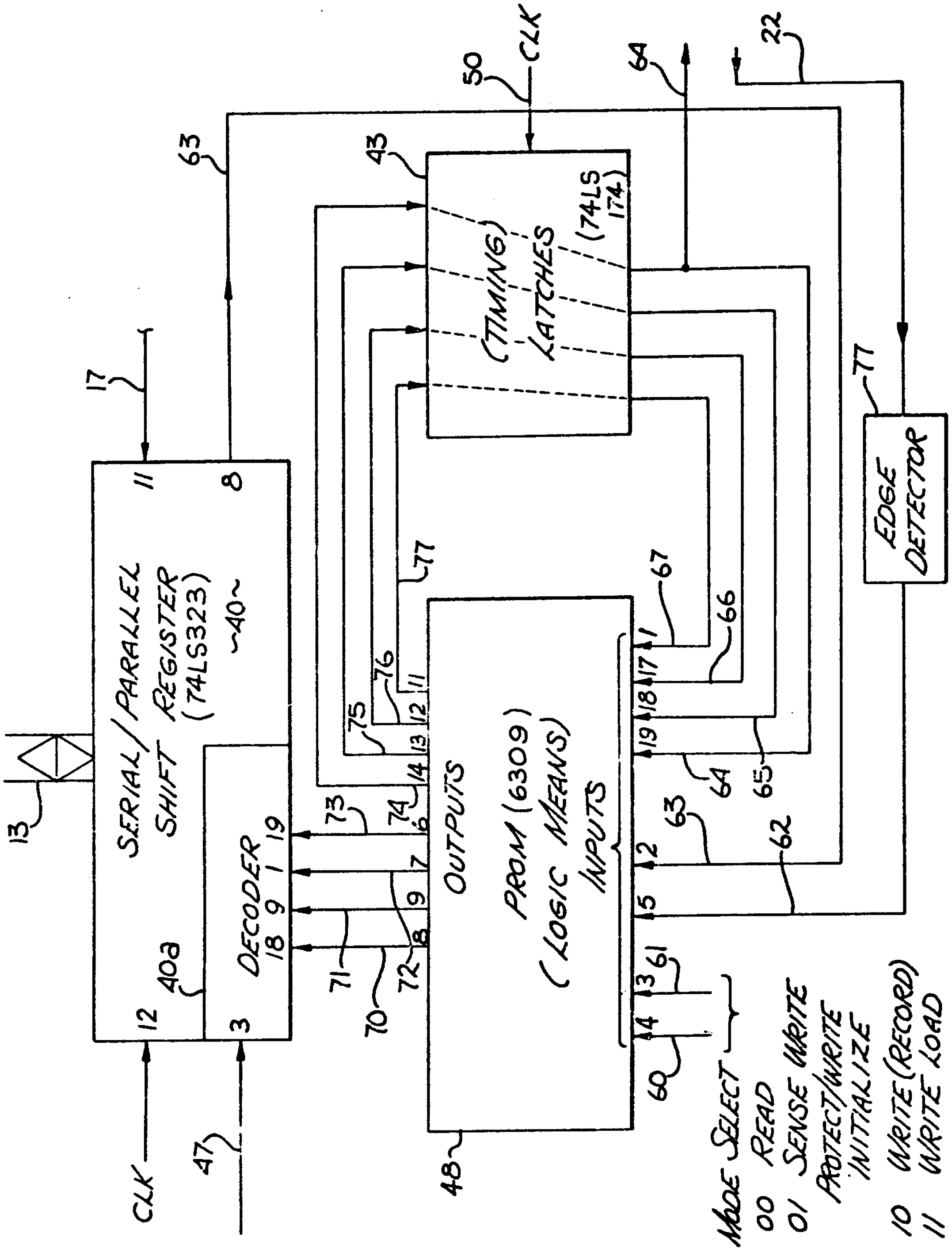


Fig. 3

“APPLE_PAT_4_210_959_04” 159 KB 2000-02-21 dpi: 300h x 300v pix: 1827h x 2823v

U.S. Patent Jul. 1, 1980 Sheet 4 of 4 4,210,959

Fig. 4

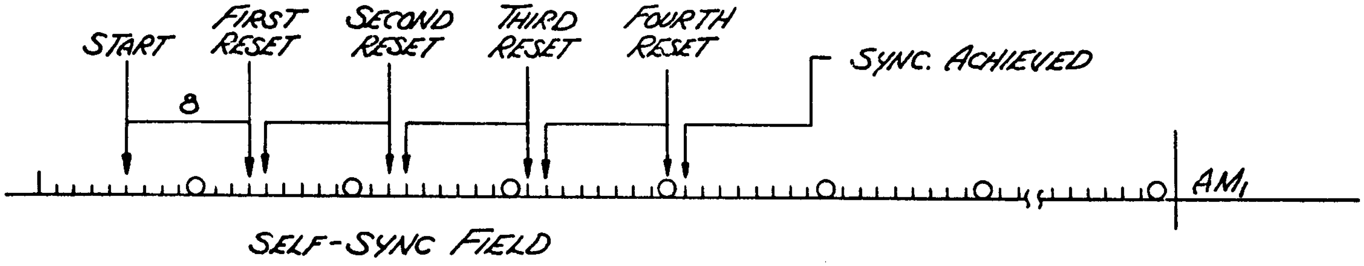


Fig. 5

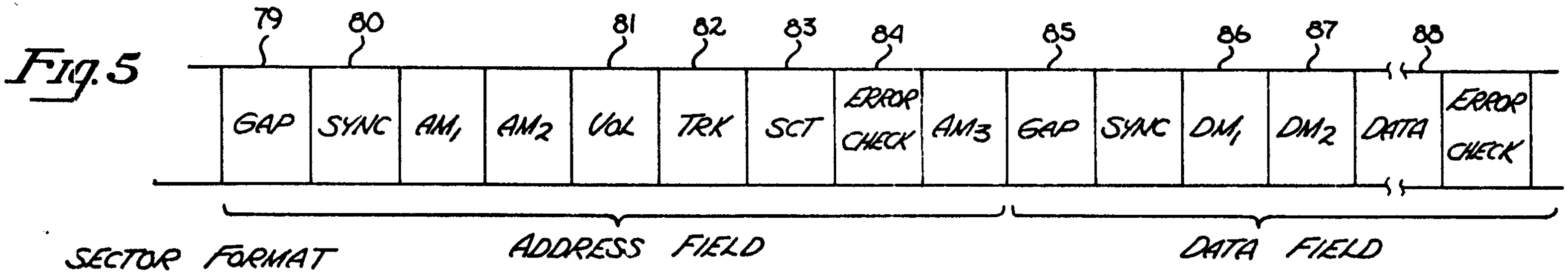
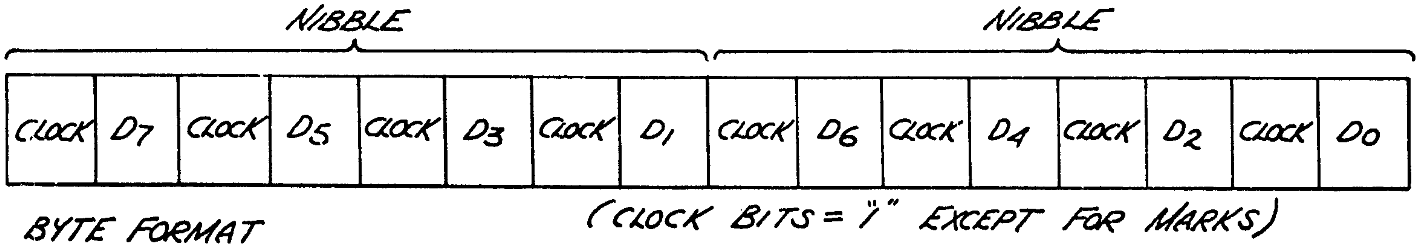


Fig. 6



“APPLE_PAT_4_210_959_05” 160 KB 2000-02-21 dpi: 300h x 300v pix: 1809h x 2921v

CONTROLLER FOR MAGNETIC DISC, RECORDER, OR THE LIKE

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention relates to the field of controllers, particularly controllers for interfacing between a digital computer and a magnetic recorder such as a floppy disc or other memories.

2. Prior Art

Numerous controllers are commercially available for interfacing between digital computers and magnetic disc recorders such as the commonly employed floppy disc recorders. These discs include a plurality of endless, concentric tracks used for storing digital data. The controller typically accepts data in parallel form from the computer and provides the data in serial form to the recorder. Serialized data from the recorder is converted to parallel form for the computer. Controllers perform other functions such as track selection and synchronization.

Commercially available controllers, particularly those for floppy disc recorders, are generally complex and expensive. Because of their cost, they do not lend themselves to the consumer field (e.g., hobby and home uses) or small business use. As will be seen, with the present invention a relatively simple, inexpensive controller is described which is suitable for consumer and small business applications. However, the principles employed in the described controller are applicable to larger, more elaborate systems.

Often disc controllers provide track selection signals to the recorder. These signals control a stepping motor which drives the read-write head to the desired track. If the motor is stepped from track-to-track, considerable time is lost in selecting non-adjacent tracks. In the prior art, complex means are used to allow the stepping motor to accelerate and decelerate to and from higher speeds when selecting tracks which are separated from one another by some distance. With the present invention, a computed velocity profile is easily implemented, thus allowing rapid selection of non-adjacent tracks.

When a track is selected, synchronization between the controller/computer and the data recorded on the track is necessary. In some cases, permanent markers, such as holes, are included with each track to provide a fixed reference point. In "soft-sectored" discs, permanent markers are not used. These discs provide wider flexibility since the user is able to format the disc to a particular application. Somewhat intricate hardware is used to provide synchronization with these soft-sectored discs. A method is described in this application which provides rapid synchronization for soft-sectored discs.

BRIEF DESCRIPTION OF THE INVENTION

An interfacing means for interfacing between a digital computer and a magnetic disc recorder, such as a floppy disc or other memories, is described. A serial/parallel register is employed for communicating with the computer on the data bus. A logic means, which may be a read-only memory, receives input signals (addresses) and provides output signals in response thereto. The logic means is controlled by a timing means which received a synchronization signal from the computer. Output signals from the logic means are coupled to the register and to the timing means. Input signals to the logic means (address signals) are received from the timing means, recorder and resister. Thus, some of the output signals from the logic means are used as address signals for selecting the next output from the logic means.

A method is described for synchronizing an n-bit digital register from the recorder. A synchronization field which comprises a plurality of codes, each having n-bits of one binary state and at least one bit of the other binary state, is coupled to the register. The register is automatically reset each time a bit of the one binary state is moved into the n^{th} stage of the register. With this synchronization field, the register automatically becomes synchronized with the codes and words.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 illustrates the controller of the present invention interfacing between a digital computer and a disc driver (recorder).

FIG. 2 is a block diagram of the controller of this present invention.

FIG. 3 is a detailed block diagram of the controller logic and timing means shown in FIG. 2.

FIG. 4 is a graph of the synchronization field used to synchronize a register with recorded data.

FIG. 5 is a graph illustrating the format for each sector of each track, in the presently preferred embodiment.

FIG. 6 is a graph illustrating the byte format employed in the presently preferred embodiment.

DETAILED DESCRIPTION OF THE INVENTION

A controller for providing an interface between a digital computer and a magnetic disc recorder or other memory means is described. While the following description is directed towards a floppy disc recorder, the invented concepts may be employed with other memory means, particularly where data is recorded in serial form such as in a charge-coupled device (CCD), bubble memory, etc. In the following description, numerous specific details are set forth such as specific word lengths, etc., to provide a thorough understanding of the present invention. However, it will be obvious to one skilled in the art that the present invention may be practiced without these specific details. In other instances, well-known circuits have been shown in block diagram form in order not to obscure the present invention in unnecessary detail.

In the presently preferred embodiment, the described controller and method of synchronization are employed to provide an interface between a microcomputer and a minifloppy disc recorder. The controller is particularly suited for use in the consumer field such as for home, hobby or small business use. In particular, the presently preferred embodiment of the controller is employed to provide an interface between an APPLE-2 computer, manufactured by Apple Computer, Inc. of Cupertino, Calif. and a SHUGART drive, Part No. SA-400, SA-390 or equivalent.

Referring first to FIG. 1, the magnetic disc controller of the present invention, shown as controller 12, interfaces between a digital computer 15 and a disc driver 16. The digital computer 15 is coupled to the controller through a data bus 13 and through an address bus 14. The controller 12 is coupled to the driver 16 through a plurality of control and data lines. The selection of the track is controlled through the signals on lines 25, 26, 27 and 28, which are coupled to the four phases of the track-select stepping motor 33 through the track-select motor amplifiers 32. The motor 33 and amplifier 32 are ordinary components commonly employed in disc drivers.

A data signal for recording data onto the disc 30 is coupled to the driver 16 through a line 64 which communicates with the head 34 through the read/write amplifiers 31. Data read from the disc 30 is coupled through the amplifiers 31 to the controller 12 via line 22. The head 34 is moved by the stepping motor 33 to the desired track on the disc 30. Enabling signals to control the driver 16 are coupled to the driver from the controller 12 via lines 19 and 20. A write protect switch 36 couples a signal 17 to the controller 12 when there is an indication within the driver that the information on a particular disc is not to be erased. This is a common signal employed with numerous disc drivers.

Before describing the controller in detail, the byte format employed in the presently preferred embodiment should be described since it will enable a better understanding of the controller. As shown in FIG. 6, the byte format consists of 8-bit nibbles. Each nibble consists of four data bits and four clock bits; two nibbles are required to store a byte of data. The clock bits are always binary ones. Thus, two consecutive binary zeros never occur in a normal data field; however, two consecutive binary zeros are employed for markers, as will be described. A nibble does not include either the more significant bits or the least significant bits; rather, the odd data bits, D_1, D_3, D_5 and D_7, are included in one nibble and the even data bits, D_0, D_2, D_4 and D_6, are included in the other nibble. By distributing the data bits in this manner, merging of the two nibbles into a standard byte is much easier. Note that if the nibbles are in parallel registers, a shift in one direction by one stage allows merger of the two nibbles into a single byte.

Referring now to FIG. 2, the main portions of the controller 12 of FIG. 1 comprise a serial/parallel shift register 40, a controller logic and timing means 24 and latches 42. The controller logic and timing means 24 is shown in its presently preferred embodiment in FIG. 3. The latches 42, which are ordinary digital latches, store data for selecting modes of operation, tracks on the disc and other control signals, as will be explained in greater detail. Also shown in FIG. 2 are a motor timer means 52 and a bootstrap memory 45.

The controller of FIG. 2, which is coupled to the computer via the data bus 13 and the address bus 14, receives a clocking signal at one input terminal of the NAND gate 51 from the computer. A controller-select signal on line 47 is coupled to the register 40 and latches 42 to indicate that the controller has been selected for operation by the computer. Other well-known control signals and lines such as those associated with power supplies are not shown in FIG. 2.

The controller of FIG. 2 is coupled to the recorder through the lines 25, 26, 27 and 28, which are also shown in FIG. 1. The enable signals on lines 19 and 20, the data lines 22 and 64, and the write protect switch signal on line 17 are also shown in FIG. 2.

The serial/parallel shift register 40 is an ordinary digital register for receiving 8-bit words, in parallel, from bus 13 and from shifting this data, serially, onto line 63 during the writing/reading mode. During the reading mode, data is serially shifted into the register 40 and then removed, in parallel, onto bus 13. The register 40 is controlled by signals coupled to the register on lines 70, 71, 72 and 73. These lines originate in the logic and timing means 24. During the reading mode, as will be described, the data shifted into the register is controlled by signals on these lines. During the reading mode, the register 40 is automatically cleared when its n^{th} stage (8th stage) contains a binary one. Note that with the byte format of FIG. 6, the first bit of each nibble is always a binary one (clock bit). In the recording mode, data is shifted into the register 40 from right to left. The register 40 is also able to shift data from right to left; this is done, as will be described in greater detail, to sense the signal on line 17.

While in the presently preferred embodiment an 8-bit register 40 is employed, a 16-bit register or two 8-bit registers may be coupled in series to allow the transfer of 16-bit words to an appropriate bus.

The bootstrap memory 45, which is coupled to the address bus 14 and the data bus 13, may be a read-only memory such as a PROM. In the presently preferred embodiment, this optionally employed bootstrap memory is a 256-byte memory used to set initial conditions for operating the system software. The memory may be employed for the reading of operating systems from the disc, or like functions.

The controller logic and timing means 24 is able to sense if the disc is up to speed and provides a signal on line 53 so indicating. This signal through timer 52 and NAND gate 51 prevents the clocking signals from being coupled to line 50 unless the disc is up to speed. The motor timer 52 controls the disc drive motor of the recorder via a signal on line 20. After data is written or read, the timer 52 prevents the disc motor from stopping for a predetermined period of time (e.g., ten seconds). Note that without this timer a considerable amount of time would be required to wait for the disc to be brought up to speed. The signals on lines 18 and 19, which are stored within the latches 42, are used to enable the recorder, including its disc motor.

In the presently preferred embodiment, the latches 42 consist of eight latches which act as a storage means and decoding means. Four lines of the address bus 14 are coupled to the latches 42. Three of these lines are used to select one of the eight latches and the remaining line is used to furnish data (binary one or zero) to the selected latch. In this manner, 8-bits of data are loaded into the latches 42. Four of these data bits are used to control the four phases of the track-select motor 33 (FIG. 1); these bits are coupled to the recorder on lines 25 through 28. Two of these data bits are coupled to lines 18 and 19 for generation of the enabling signals for the recorder. The remaining two bits are coupled to lines 60 and 61 as will be described in greater detail in conjunction with FIG. 3 to select a mode of operation for the controller.

In typical operation, the computer through the controller of FIG. 2 senses the position of the head (current track) over the disc. Specifically, the track number which is read by the head is coupled to the computer through the register 40. The computer is able to compute the ideal velocity profile for moving the head to the desired track from the current track with a relatively simple algorithm. Since all four phases of the stepping motor 33 are controlled through the latches 42, rapid acceleration and deceleration, and higher rates of rotation, are obtainable when compared to stepping the motor from track-to-track.

To achieve an efficient velocity profile for a stepping motor, the prior art often resorted to relatively complex hardware. With the latches 42 and its coupling to the computer through the address bus, an ideal velocity profile may be quickly and efficiently computed without such hardware.

In FIG. 3, the register 40 and the data bus 13 are again shown. The controller logic and timing means 24 of FIG. 2 comprises a logic means 48 and a timing means 43. The logic means 48 may be any logic means adaptable for receiving input signals and for providing predetermined output signals in response thereto. Thus, ordinary logic gates or other known logic arrays may be used. In the presently preferred embodiment, a read-only memory, specifically a PROM, is employed. The timing means, in the presently preferred embodiment, comprises four (4) latches which are controlled by the clocking signal, line 50.

The logic means (PROM) 48, in the presently preferred embodiment, comprises a 256-byte memory which provides an 8-bit output on lines 70 through 77 for each 8-bit address received on lines 60 through 67. The specific functions controlled by the PROM shall be discussed below. The specific program stored within the PROM 48 for the currently preferred embodiment is shown in TABLE I.

Two of the address signals to the PROM 48 are coupled from the latches 42 of FIG. 2 on lines 60 and 61. These signals select the four possible modes of operation; specifically, read (00), sense write protect/write initialize (01), write (10) and write load (11). Another input to the PROM 48, line 62, is the signal sensed by the recorder head which is coupled to the controller on line 22. An ordinary edge detector 77 is used for detecting the edge of this signal and for providing a binary signal on line 62. The input signal on line 63 is the serial output from the register 40. The remaining four address signals to the PROM 48, lines 64, 65, 66 and 67 are output signals from the latches 43.

As is apparent from FIG. 3, four of the eight bits of output from the PROM 48 provide address signals for the PROM. The signals on lines 75 through 77 provide input address signals for lines 63 through 67. One of these lines, line 64, also provides the recording signal for recording data onto the disc.

Assume that the controller is in the reading mode as determined by the 00 signal applied to the PROM 48 on lines 60 and 61. The latches 43 operate at twice the cycle rate of the microprocessor which corresponds to a rate eight times faster than the bit cell disc rate; thus the latches continually release address signals to the PROM. Initially, the register 40 is empty (all zeros). In the presently preferred embodiment, if a transition occurs on line 62 in 11 or fewer of such latch cycles, a binary one is recognized and the PROM provides an output on lines 70, 71, 72 and 73 which, after decoding by the decoder 40*a*, shifts a binary one in the first stage of the register 40. If twelve such cycles occur without a transition, a zero is shifted into the register 40. (As will be described later, if the first bit is a zero it will be skipped). This continues until the register is full. Counting effectively occurs by the repeated addressing of the PROM as the signals pass through the latches 43.

The computer senses a full register by polling the data bus and by specifically determining if a binary one is in the n^{th} stage of the register. As mentioned, the first bit of each nibble is always a binary one. When the register is full, the computer removes the data through the data bus 13 and the register 40 is cleared. In the interim, the PROM 48 waits for a binary one and the following bit. Then it writes this binary one and the next binary bit into the register 40. The temporary delay of shifting into the register is necessary to provide ample time for the computer to withdraw the contents of the register 40. When a full register occurs, if the first bit sensed by the PROM is a zero, it is effectively skipped, although shifting a zero into the empty register would not affect the operation of the device. In this manner, nibble after nibble is read from the disc, shifted serially into the register 40 and then removed in parallel onto the data bus 13. The PROM 48 provides the logic to insure the shifting of the correct binary bits into the register 40 as a function of the signal on line 22, which is coupled to the PROM on line 62.

In the recording (writing) mode, the mode select signal (10) is applied to the PROM 48 on lines 60 and 61. (Previously, each nibble is shifted into the register 40, in parallel, from the computer during the write load mode (11).) Every eight clock cycles, the signals on lines 70 through 73 cause the register 40 to shift its contents to the right by one stage. For each such shift, the next bit in the register is coupled to line 63. The signal on line 63 determines the output signal from the PROM, and particularly, the signal on line 74 which is coupled to the recorder via line 64 after passing through latches 43. Each of the 8-bits are thus shifted from the register 40 and supplied to the recorder.

In the presently preferred embodiment, the mode select signals change to 11 for the loading of the register 40 from the data bus 13. Note that this is not necessary, and that the computer could directly communicate with the register 40 for purposes of loading data into the register.

During the sense write protect/write initialize mode, the signal on line 17 is shifted to the left by the register 40 and sensed by the computer. In this manner, the computer can determine if the particular disc on the recorder should not be written onto and provide an appropriate indication to the operator. Other data or signals may be transmitted on line 17 where appropriate.

As mentioned, in the presently preferred embodiment, a soft-sectored disc is employed. When the recorder is first selected, signals from the disc are coupled through lines 22 and 62 to the PROM 48 (reading mode). These are shifted into the register 40. One problem with a soft-sectored disc is that there is no immediate way of determining where in a nibble reading first occurred. Some means or method must be provided to align or synchronize the shifting of the data into the register 40 with the nibbles recorded on the disc.

Referring to FIG. 4, a self-synchronizing field of coded words are employed to provide synchronization. Each word consists of eight binary ones followed by a binary zero. This self-synchronization field, in a more general form, consists of n-binary ones where n corresponds to the number of stages in the register followed by at least one binary zero. As will be seen with these n+1 codes, the register 40 resets with every n bits until the first bit is a binary zero. Then the register resets with every n+1 bits.

Referring to FIG. 4, assume that the disc includes the above-described synchronization field. Assume further that reading begins where indicated by the start line. This first binary one is shifted into the register 40 of FIG. 3. Eight bits later the first reset occurs and the register automatically clears since a binary one is in the n^{th} stage. Following this, eight bits later, the second reset occurs. The third reset occurs eight bits later as indicated in FIG. 4, and finally, the fourth reset occurs. (When the fourth reset occurs, a binary zero is in the first stage of the register.) From the next bit forward the register will automatically clear every $n+1$ bits later, and thus the register will be completely cleared when the address maker (AM_1) reaches the register. Likewise, if reading begins before the synchronization field, synchronization will be achieved before the end of the field.

In the presently preferred embodiment, each of the tracks is divided into 11 sectors; one such sector is shown in FIG. 5. Each sector includes gaps, such as gap **79**, to compensate for variations in the disc rate, since the disc is not always driven precisely at the same rate of rotation.

Following the gap **79**, there is a synchronization field **80** which corresponds to the field shown in FIG. 4. While in theory only 8 codes of n binary ones and a binary zero are required to synchronize the register, eleven such codes are used within the synchronization field **80** to assure synchronization.

Following the synchronization field, the address marker identified as AM_1 appears on the track. This marker is used to indicate that an address follows. In the presently preferred embodiment, two consecutive address markers are employed as shown by AM_1 and AM_2. The address markers are distinct and immediately recognized by the computer. As previously mentioned with the normal data field and other information on the disc, every other bit is a clock bit (binary one). Thus, two binary zeros do not occur in succession. However, each marker includes both a missing clock bit and data bit. If reading beings in the middle of a data nibble, only a data bit or a clock bit may be missing, but not both. Thus, the computer cannot mistake data or other information for markers.

After the computer senses the address markers, it then knows that address information follows. Three words consisting of a volume number, a track number and a sector number (shown as words **81**, **82** and **83**, respectively) are read from the discs to provide an address. Following the sector number, an error check is made on the volume, track and sector numbers. In the presently preferred embodiment, an exclusive ORing of these three numbers is employed and checked with the error check word **84**. A third address marker, AM_3, is used in the presently preferred embodiment to close the address field.

Next the data field begins, starting with a gap **85**. Following the gap **85**, synchronization is again required and thus a synchronization field, such as the field **80**, is repeated. Two data markers **86** and **87** are used to introduce the stored data **88**. In the presently preferred embodiment, 256 8-bit words (256 nibbles) are stored within data **88**. Then an error check is made.

No matter where reading begins within the address field or data field of FIG. 5, synchronization is achieved before the computer accepts data. The computer will not accept any data (including addresses) unless it is preceded by a recognized marker. To recognize a marker, the marker must be properly aligned within the stages of the register **40**.

By way of example, if reading begins in the middle of the volume number, the data corresponding to this number, the track number, sector number and error

check will be serially moved through the register. The marker AM_3 will not be recognized since synchronization has not yet occurred. Following the gap **85** a synchronization field is reached and synchronization occurs. Then the data field markers DM_1 and DM_2 are recognized. However, since they were not preceded by an appropriate address marker, the data which followed is ignored. After a gap (corresponding to gap **79**), a synchronization field (corresponding to synchronization field **80**) is reached, and synchronization occurs. The markers AM_1 and AM_2 are recognized allowing the identification of the volume, track and sector. Then the data stored within that volume, track and sector will be read, if required.

Thus, a controller for interfacing between a digital computer and a recorder, or the like, has been described. A minimum of hardware is required to fabricate the controller. Synchronization with a soft-sectored disc is achieved without additonal hardware by reading a predetermined self-synchronization field from the disc.

TABLE I

C700-	DA	0D	18	38	0A	OA	0A	0A
C708-	18	39	18	39	18	3B	18	3B
C710-	18	38	08	38	0A	0A	0A	0A
C718-	18	39	18	39	18	3B	18	3B
C720-	OD	OD	28	48	0A	0A	0A	0A
C728-	28	48	28	48	28	48	28	48
C730-	28	48	28	48	0A	0A	0A	0A
C738-	28	48	28	48	28	48	28	48
C740-	0D	0D	58	C8	0A	0A	0A	0A
C748-	58	78	58	78	58	78	58	78
C750-	58	78	58	78	0A	0A	0A	0A
C758-	58	78	58	78	58	78	58	78
C760-	0D	0D	C8	C8	0A	0A	0A	0A
C768-	68	08	68	88	63	08	68	88
C770-	68	88	68	88	OA	OA	OA	OA
C778-	68	08	68	88	68	08	68	88
C780-	0D	CD	C8	C8	0A	0A	0A	0A
C788-	98	B9	98	B9	98	BB	98	BB
C790-	98	BD	98	B8	OA	OA	OA	OA
C798-	98	B9	98	B9	98	BB	98	BB
C7AO-	OD	D9	C8	C8	OA	OA	OA	OA
C7A8-	A8	C8	A8	C8	A8	C8	A8	C8
C7BO-	O9	39	A8	AO	OA	OA	OA	OA
C7B8-	A8	C8	A8	C8	A8	C8	A8	C8
C7CO-	D9	FD	D8	F8	OA	OA	OA	OA
C7C8-	D8	F8	D8	F8	D8	F8	D8	F8
C7DO-	D9	FD	D8	F8	OA	OA	OA	OA
C7D8-	D8	F8	D8	F8	D8	F8	D8	F8
C7EO-	1D	DD	E8	EO	OA	OA	OA	OA
C7E8-	E8	88	E8	O8	E8	88	E8	O8
C7FO-	1D	6D	E8	EO	OA	OA	OA	OA
C7F8-	E8	88	E8	O8	E8	88	E8	O8

I claim:

1. An interfacing means for interfacing between a digital computer and storage device such as a magnetic disc recorder, comprising:

register means having a plurality of parallel input-/output lines for coupling to said computer and a serial input/output line for coupling to said storage device, said register means also having a control means having a plurality of register control lines for controlling transfer of data to and from said register means;

latch means having a plurality of latch input lines and a plurality of latch output lines, said latch means for controlling the flow of digital signals between latch input lines and latch output lines in response to a timing signal,

clock means for generating said timing signal, read-only memory means having an address means with a plurality of address input lines, and a plurality of data output lines, said read-only memory means for receiving input signals on said address input lines and for providing predetermined output signals on said data output lines in response thereto;

a portion of said read-only memory means data output lines being coupled to said register control means via said register control lines for controlling data flow to and from said register, another portion of said read-only memory means data output lines being coupled to said latch means input lines, said latch means output lines being coupled to said addressing means via a portion of said read-only memory means address lines for providing a portion of an address thereto such that some of said output signals from said read-only memory means are employed as next address signals to said read-only memory means in response to said timing signal;

said interfacing means further including means for coupling the addressing means via another portion of said read-only memory means address lines to said computer whereby said computer may communicate with said storage device through said interfacing means.

2. The interfacing means defined by claim 1 including storage means for receiving digital signals from said computer and for providing control signals to said storage device for controlling track selection, said storage means being coupled to said computer and said storage device.

3. The interfacing means defined by claim 2 wherein said means for coupling the addressing means via a portion of said read-only memory address lines to said computer comprises said storage means.

4. The interfacing means defined by claim 3 wherein said input signal to said read-only memory means from said storage means is used to select a reading mode or a recording mode.

5. The interfacing means defined by claim 4 wherein said storage means comprises digital latches.

6. The interfacing means defined by claim 5 wherein said computer has an address bus and a data bus and wherein said ditigal latches are coupled to said address bus of said computer.

7. The interfacing means defined by claim 1 wherein at least one of said output signals from said read-only memory means is coupled to said storage device to provide a signal for a writing mode.

8. The interfacing means defined by claim 7 wherein said register means serially provides a digital signal on said serial input/output line as one of said address inputs to said read-only memory means during said writing mode.

9. The interfacing means defined by claim 8 wherein said output signals from said read-only memory means controls the serial loading of a digital work into said register means during a reading mode.

10. The interfacing means defined by claim 1 wherein said register means receives digital words in parallel form from said computer and serially couples said words to said read-only memory means during a writing mode.

11. The interfacing means of claim 1 wherein said several input/output line of said register means is coupled to said recorder through said latch means and said read-only memory means.

12. An interfacing means for interfacing between a digital computer and storage device such as a magnetic disc recorder, comprising:

register means having a plurality of parallel input/output lines for coupling to said computer and a serial input/output line for coupling to said storage device, said register means also having a control means having a plurality of register control lines for controlling transfer of data to and from said register means;

latch means having a plurality of latch input lines and a plurality of latch output lines, said latch means for controlling the flow of digital signals between latch input lines and latch output lines in response to a timing signal;

clock means for generating said timing signal;

read-only memory means having an address means with a plurality of address input lines, and a plurality of data output lines, said read-only memory means for receiving input signals on said address input lines and for providing predetermined output signals on said data output lines in response thereto;

a portion of said read-only memory means data output lines being coupled to said register control means via said register control lines for controlling data flow to and from said register, another portion of said read-only memory means data output lines being coupled to said latch means input lines, said latch means output lines being coupled to said addressing means via a portion of said read-only memory means address lines for providing a portion of an address thereto such that some of said output signals from said read-only memory means are employed as next address signals to said read-only memory means in response to said timing signal,

said interfacing means further including means for coupling the addressing means via another portion of said read-only memory means address lines to said computer, said read-only memory means being coupled to receive a data signal from said storage device as an address input on one of said address input lines during reading of data, said read-only memory means being coupled to said serial input/output line of said register means to receive a data signal from said register means as an address input on another of said address input lines, and one of said read-only memory data output lines being coupled to said storage device to provide a data signal thereto during recording of data, whereby said computer may communicate with said storage device through said interfacing means.

13. The interfacing means defined by claim 12 wherein said control signal for said latching means comprises a synchronization signal from said computer.

14. The interfacing means defined by claim 12 including storage means coupled to said computer, said read-only memory means and said storage device for receiving signals from said computer for controlling track selection by said storage device and for selecting said recording or reading of data.

* * * * *

United States Patent [19]

Wozniak

[11] **4,217,604**

[45] **Aug. 12, 1980**

[54] **APPARATUS FOR DIGITALLY CONTROLLING PAL COLOR DISPLAY**

[75] Inventor: **Stephen G. Wozniak, San Jose, Calif.**

[73] Assignee: **Apple Computer, Inc., Cupertino, Calif.**

[21] Appl. No.: **941,032**

[22] Filed: **Sep. 11, 1978**

[51] **Int. Cl.²** **H04N 9/38**

[52] **U.S. Cl.** **358/16;** 358/82; 358/183

[58] **Field of Search** 358/16, 17, 81, 82, 358/183

[56] **References Cited**

U.S. PATENT DOCUMENTS

3,742,125	6/1973	Siegel	358/183
3,936,868	2/1976	Thorpe	358/183

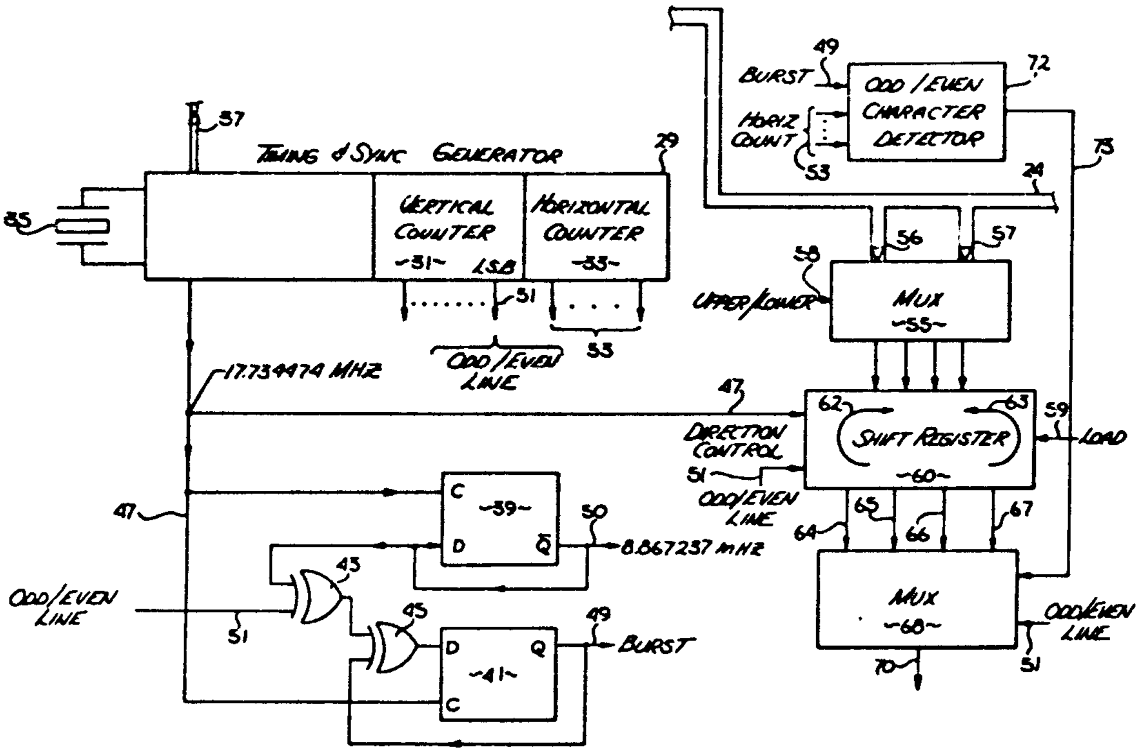
Primary Examiner—Richard Murray
Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman

[57] **ABSTRACT**

An apparatus for digitally controlling the display of color in a phase alternation line (PAL) video display is disclosed. Coded digital signals are shifted in a recirculating shift register. The direction of shifting in the register and the stage at which the signals in the register are sensed are changed as a function of odd/even display lines to compensate for PAL phase reversals.

12 Claims, 6 Drawing Figures

Apple 2



"APPLE_PAT_4_217_604_01" 129 KB 2000-02-21 dpi: 300h x 300v pix: 1894h x 2548v

U.S. Patent Aug. 12, 1980 Sheet 1 of 2 4,217,604

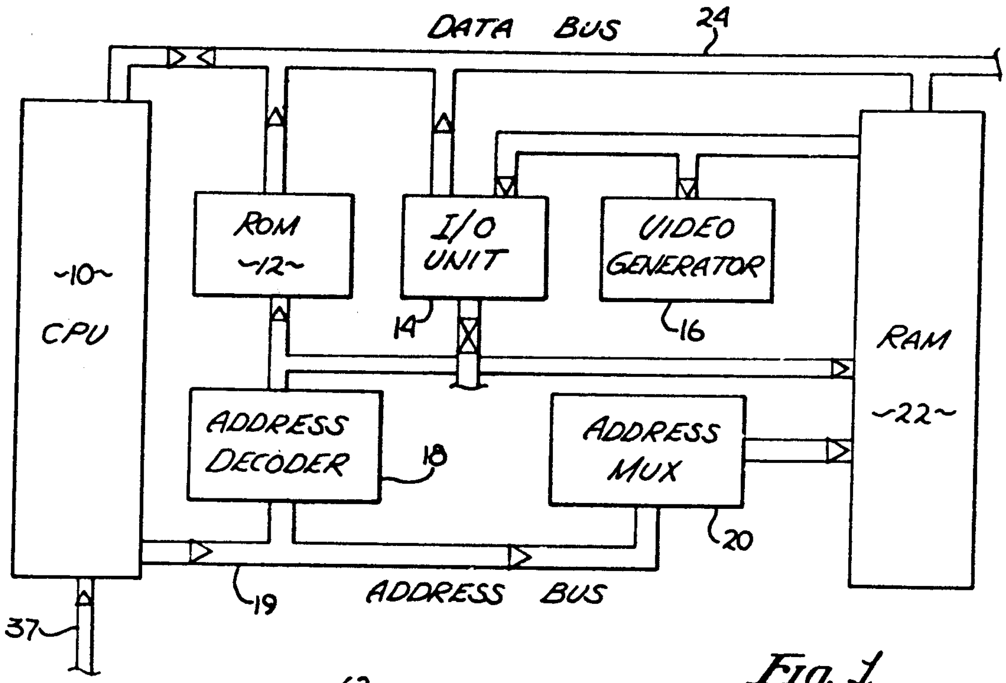


Fig. 1

Fig. 3a

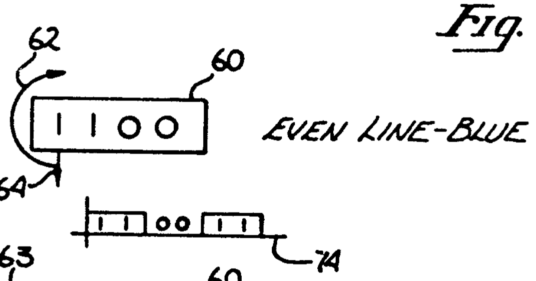


Fig. 3b

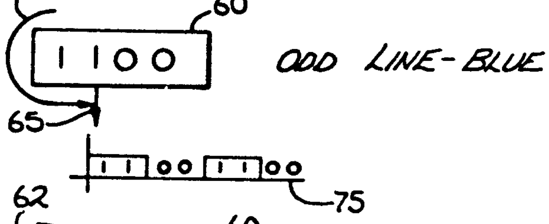


Fig. 3c

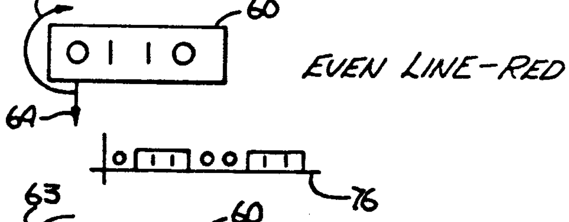
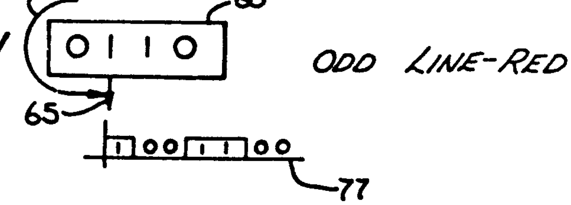
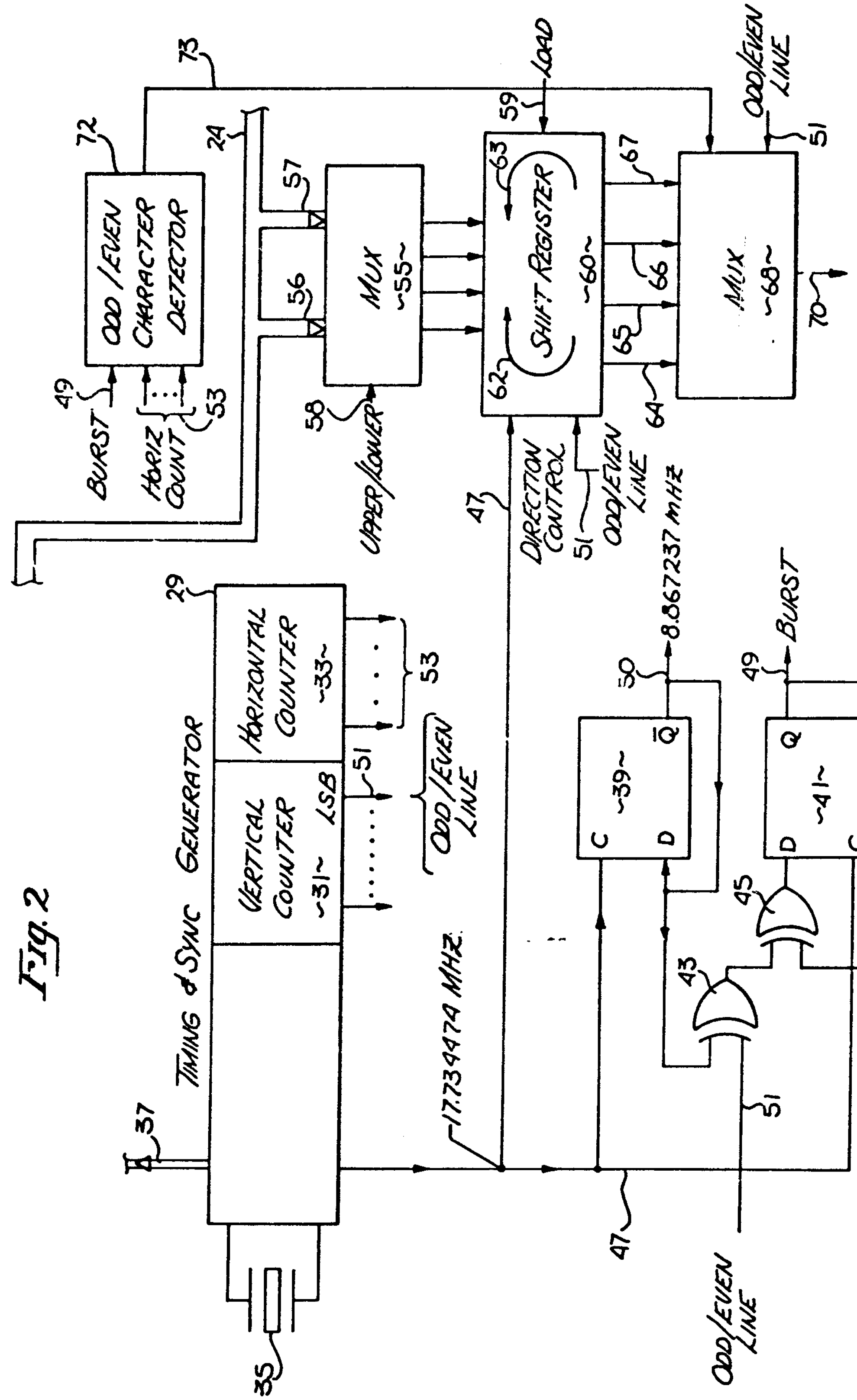


Fig. 3d



U.S. Patent Aug. 12, 1980 Sheet 2 of 2 4,217,604



APPARATUS FOR DIGITALLY CONTROLLING PAL COLOR DISPLAY

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention relates to the field of digitally controlling video displays.

2. Prior Art

With the reduced costs of large scale integrated circuits, it has become possible to provide low cost microcomputers suitable for home use. One such use which has flourished in recent years is the application of microcomputers to video displays for games, graphic displays and the like. Most often an ordinary television receiver is employed as the video display means. The standard, raster scanned cathode ray tubes employed in these receivers present unique problems in the interconnecting of these displays with the digital information provided by the microcomputer.

These receivers are designed to operate with one or more standard video broadcasting schemes such as PAL. In copending application Ser. No. now U.S. Pat. No. 4,136,359 786,197, filed Apr. 11, 1977, entitled "Microcomputer For Use With Video Display", and assigned to the assignee of this application, a digitally controlled display is described. This earlier filed application deals in part with the generation of color signals for the television broadcasting system employed in the United States and some other countries, referred to as the National Television Systems Committee (NTSC) standard.

In many parts of the world and particularly in Europe, a phase alternation line (PAL) system is employed for television broadcasting. The PAL raster scan displays employ different signals (e.g. different frequencies, format, etc.) than the NTSC standard. The subject of this application is a unique means for generating color signals in a digital manner which are compatible with a PAL display such as an ordinary PAL compatible television receiver.

SUMMARY OF THE INVENTION

An improved color generation means for a video display is described. The color generation means is particularly useful with a digital computer which provides digital signals for controlling a raster scan display where the display is adapted for functioning with a standardized video signal such as a PAL signal. The improved color generation means includes a circuit means for providing a signal representative of the odd and even lines on the display. A recirculating shift register means is employed which is capable of selectively recirculating digital signals in both directions. Coded digital signals when read from the register means provide a video color signal since shifting in the register means occurs at a frequency compatible with the display. The register means is coupled to the circuit means such that the signal representative of the odd and even lines selects the direction of the recirculation in the register. A multiplexing means is employed to select alternate stages of the register means. In this manner compensation for the color signal phase reversals of the coded digital signals is provided.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a general block diagram of the microcomputer which generates the digital signals used to control the video display.

FIG. 2 is a block diagram and circuit diagram of the improved color generation means of the present invention, the circuit employed to provide a continuous "burst" signal and the timing and synchronization generator.

FIGS. *3a*, *3b*, *3c*, and *3d* are a series of diagrams used to illustrate the effects of shifting digital signals in a register in different directions and of sensing the signals at different stages in the register. These diagrams are used to explain the operation of the color generation means of FIG. 2.

DETAILED DESCRIPTION OF THE INVENTION

An improved color generation means particularly suited for use with a phase alternation line (PAL) video display such as a PAL-compatible television receiver is described. In the following description numerous specific details such as specific frequencies and number of bits are set forth to provide a thorough understanding of the present invention. However, it will be obvious to one skilled in the art that the invention may be practiced without these specific details. In other instances, well-known circuits such as logic circuits and timing circuits have not been set forth in detail in order not to obscure the present invention in unnecessary detail.

Throughout this application, reference is made to the PAL broadcasting system or standard (or PAL signal). Since numerous details associated with PAL broadcasting are well-known to one skilled in the art, they are not set forth in the present application. Numerous texts and patents describe this broadcasting system in detail. For example, see *Receiving PAL Colours Television* by A. G. Priestley, published by Fountain Press, England, 1974.

In its presently preferred embodiment, the improved color generation means of the present invention is incorporated with a microcomputer which generates a PAL compatible video signal which may be directly coupled to a standard PAL television receiver. This microcomputer which is sold under the trademark "Apple" includes many modes of operation, however, only the generation of the color video signal and its related circuitry are part of the present invention. Thus, many aspects of this microcomputer are not discussed in the present application. It will be apparent to one skilled in the art that anyone of a plurality of other commercially available microcomputers may be used with the invented color generation means.

In FIG. 1 the microcomputer includes a microprocessor or central processing unit (CPU) 10. In the presently preferred embodiment, a commercially available microprocessor Part Number 6502, is used. The CPU 10 communicates through bidirectional tristate buffers (not illustrated) with a data bus 24. The microcomputer includes two memories; one memory is a 12 K (bytes) read-only memory (ROM) 12 which is coupled to the data bus 24. This memory is used for program storage. The second memory is used for general storage for the microcomputer and comprises the random-access memory 22. This memory, in the commercial embodiments of the microcomputer, contains between 4 K to 48 K (bytes) and consists of commercially available dynamic MOS memories.

The address decoder 18 receives address signals from the address bus 19 and decodes them in a well-known manner. The address decoder 18 is coupled to the ROM 12 and RAM 22. Address signals on the bus 19 are also coupled to the address multiplexer 20. This multiplexer couples address signals to the RAM 22.

The input/output (interface) unit 14 provides ports which allow the microcomputer to be electrically coupled to a cassette jack, floppy disc or to other electrical means. Known buffers and timing means are employed for this purpose.

The video generator 16 receives signals from the input/output means 14 and also from the RAM 22. This generator provides video signals for the display. A portion of this generator particularly that portion which generates the PAL-compatible color signal is discussed in detail in conjunction with FIG. 2.

In FIG. 2 a timing and synchronization generator 29 is illustrated which provides the timing signals for the computer on lines 37 and for a display coupled to the microcomputer. Ordinary timing means may be employed for the generator 29. A crystal 35 is used to provide the basic timing for the timing and synchronization generator 29. Among the frequencies provided by means 29 is the 17.734474 MHz timing frequency on line 47. This signal is used to generate a continuous "burst" signal on line 49, as will be discussed. This frequency is also used by the shift register 60.

A digital counter 33 which is part of the timing and synchronization generator 29 provides a digital count representative of horizontal beam location. The digital output of the counter 33, lines 53, are coupled to the odd/even character detector 72 in addition to other means within the computer. In the presently preferred embodiment, the 17.734474 MHz signal is divided first by 18 and then by 63 to provide a timing signal to drive the counter 33. This counter resets at a count of 1134, one count less than the standard PAL line count. The resetting of the counter 33 advances the digital counter 31. The digital counter 31 which provides a digital signal representation of a vertical beam location in the presently preferred embodiment, is reset at an even number, specifically 312. The least significant bit of this counter, which is coupled to line 51, provides an odd-/even line signal. That is, when a binary zero is present on line 51, an even numbered line is being scanned, whereas when a binary one is present on line 51, an odd numbered line is being scanned.

In FIG. 2 the circuit comprising the bistable circuits, flip-flops 39 and 41 and the exclusive OR gates 43 and 45 are used to generate a timing signal of 8.867237 MHz on line 50 and also a continuous "burst" signal on line 49. The signal on line 49 is a 4.43361875 MHz signal with phase alterations of ±45° for odd/even lines.

The timing signal on line 47 from generator 29 is coupled to the C terminals of the flip-flops 39 and 41. The $\bar{Q}$ output terminal of the flip-flop 39 (line 50) is coupled to the D terminal of this flip-flop and also to one input terminal of the gate 43. The other input terminal of gate 43 receives the odd/even line signal on line 51. The output of the gate 43 is coupled to one input terminal of the exclusive OR gate 45. The other input terminal to this gate is coupled to line 49. The output of the gate 45 is coupled to the D terminal of the flip-flop 41. An examination of the logic associated with these flip-flops and gates will reveal that the burst signal on line 49 is in fact a signal having a frequency ¼ that of the

signal on line 47 with a ±45° phase shift for odd/even lines.

As presently implemented, an odd number of characters, each of equal width, are displayable on the display in the horizontal direction. However, the width of each of these characters is not an integer number of color reference frequency cycles. For this reason, compensation from character-to-character for the video color signal (line 70) must be provided. A signal is developed on line 73 by the odd/even character detector 72 for this purpose. This detector receives the continuous "burst" signal (line 47) and also the horizontal line count (lines 53). At the beginning of each character as determined by the horizontal count (this is also the time at which color data is loaded into the shift register 60 from the multiplexer 55) the "burst" signal is examined. If the signal is in its high state or low state a signal of a first binary state is coupled to line 73. If the signal is in a transition, a signal of the second binary state is coupled to line 73. Obviously, ordinary logic means may be employed to implement the odd/even character detector 72.

The video color signal is primarily generated within the shift register 60 and is coupled to the line 70 through the multiplexer 68. In the presently preferred embodiment, a four bit shift register is employed. However, the principles described herein are applicable to a shift register with more than four stages. The shift register is of the recirculating type, thus it continually circulates the data which is shifted into the register in parallel form from the multiplexer 55. This data is shifted at the rate of 17.734474 MHz which signal frequency is coupled to register 60 via line 47. The register 60 is capable of shifting the data in two directions as indicated by directions 62 and 63. The direction control is determined by the odd/even line signal coupled to the register 60 on line 51. Also a load signal is coupled to the register 60 on line 59. As mentioned, loading occurs at the beginning of each character for the presently preferred embodiment, however, loading may occur at other times.

As presently implemented in the microcomputer, four lines 57 of the eight lines of the data bus 24 are employed to provide color data for the upper portion of each character, while four lines 56 of the data bus 24 are employed to provide color data for the lower portion of each character. The multiplexer 55 thus selects either the lines 56 and 57 and couples these lines to the register 60. The microcomputer generates a signal indicating whether the upper portion or lower portion of the character is being written and furnishes an appropriate signal to the multiplexer 55 on line 58. However, this particular coupling for obtaining the color data is not critical to the present invention.

Each stage of the register 60 is coupled to the multiplexer 68 so as to permit sensing of the data at any stage of the register as the data is recirculated. Thus the multiplexer permits selective coupling of one of the lines 64, 65, 66 or 67 to line 70. The particular tap or stage of the register which is selected is a function of the binary signals coupled to the multiplexer 68 on lines 51 and 73.

The operation of the improved color generation means shall first be described without the additional compensation required from character-to-character as provided by the signal on line 73. Referring to FIG. 3a, assume that the binary coded signal 1100 represents the color blue and that the this code is shifted into the shift register 60 from the multiplexer 55 (FIG. 2). Furthermore assume that an even line is being scanned and that the direction of circulation within the register 60 is shown by direction 62. For these conditions also assume that the first stage of the register is selected by the multiplexer 68 as indicated by line 64. The resultant signal on line 64 which is coupled to line 70 through the multiplexer is shown by the graph 74. This video signal provides a pure blue color on the display. When an odd line is being scanned, the video signal for blue does not shift in phase for standard PAL broadcasting. However, when an odd line is scanned the direction of recirculation within the register 60 as shown in FIG. 3*b* changes and as indicated by direction 63. For an odd line the second stage of the register 60 is selected (line 65). The selection of this stage is controlled by the signal coupled to the multiplexer 68 via line 51 (FIG. 2). The signal sensed on line 65 is shown by the graph 75. As may be seen this signal is identical with that shown in FIG. 3*a*. Thus even though the direction of recirculation has changed within the register, because of the different stage selection, the same signal results. This, of course, is the desired result since there is no difference in the blue color video signal from line-to-line

Referring now to FIG. 3*c*, assume that the binary code for a pure red signal is 0110 and that the digital signal for this color is shifted into the register 60. If an even line is scanned, again the digital signal is recirculated in direction 62 and the signal is sensed on line 64. This results in the signal shown in graph 76. As shown in FIG. 3*d*, for an odd line the signal 0110 is shifted in direction 63 and sensed on line 65. The resultant signal is shown on graph 77. Note that there is a 180° phase reversal between the signals of FIGS. 3*c* and 3*d* for the odd and even lines. This phase reversal corresponds to the red signal phase reversals implemented in PAL broadcasting.

While only a pure blue and pure red case have been illustrated above, appropriate video color signals are generated by register 60 for all possible colors obtainable with the four bit color signals employed in the preferred embodiment. Moreover, color coded signals with greater number bits (e.g. 8 bits) may be employed with larger shift registers (e.g. 8 stages) with the same result.

With the additional signal provided on line 73 (FIG. 2) the multiplexer 68 selects one of the lines 64, 65, 66 and 67 as a function of the signals on lines 51 and 73. Specifically, line 64 is selected for an odd character, even line; line 65 for an odd character, odd line; line 66 for an even character, even line, and; line 67 for an even character, odd line. This selection of lines provides the additional required character-to-character compensation.

Thus, by changing the direction of shifting in the register 60 as a function of even lines and odd lines and also by appropriately selecting different stages of the register, compensation is provided for the color signal phase reversals implemented in PAL broadcasting.

I claim:

1. In an apparatus for use with a phase alternation line video display adapted to receive color signals having a color subcarrier reference signal of frequency N, an improved color signal generation means comprising:

means for generating at least one digital word which corresponds to a predetermined color, said digital word comprising a plurality of bits;

storing means for storing said digital word;

circuit means for providing an odd/even signal representative of odd and even lines on said display;

sampling means coupled to said storing means for sequentially sampling each of said bits of said digital words at a predetermined sampling rate in a first sequence and a second sequence opposite to said first sequence, said first and second sequences being selected according to said odd-even signal and said predetermined sampling rate being selected such that a color signal is developed at an output of said sampling means which corresponds to said predetermined color and which has a frequency component at said frequency N;

whereby a color signal suitable for use with a phase alternation line video display is generated.

2. The color signal generation means of claim 1 wherein said sampling means includes bit selecting means for controlling which of said bits is sampled in the beginning of said first and second sampling sequences.

3. The color signal generation means of claim 2 wherein said sampling means is a recirculating shift register means having a plurality of stages for receiving said digital word from said storing means and for circulating said digital word at said predetermined sampling rate in a first direction which corresponds to said first sampling sequence, and a second direction which corresponds to said second sampling sequence.

4. The color signal generation of claim 3 wherein said bit selecting means comprises a multiplexing means for selectively coupling one of said stages of said shift register means to said output.

5. The color signal generation means of claim 4 wherein said selective coupling of said multiplexing means occurs in response to said odd-even signal.

6. The color signal generation means of claim 5 wherein said shift register means is comprised of P number of said stages and said predetermined sampling rate is at a frequency approximately equal to N×P.

7. The color signal generation means of claim 6 wherein said circuit means comprises a digital counter and said odd/even signal is derived from the least significant bit of said counter.

8. The color signal generation means defined by claim 6 wherein P is equal to four.

9. The color signal generation means of claim 8, wherein N is approximately 4.434 MHz and said predetermined sampling rate is approximately 17.734 MHz.

10. The color signal generation means of claim 9 wherein said at least one digital word and said corresponding predetermined color comprise the following:

Digital word	Corresponding Color
1100	Blue
0110	Red

11. In an apparatus for use with a phase alternation line video display adapted to receive color signals having a color subcarrier reference signal of frequency N and where said apparatus provides a predetermined number of characters at least in the horizontal direction on said display, an improved color signal generation means comprising:

means for generating at least one digital word which corresponds to a predetermined color, said digital word comprising P number of bits;

storing means for storing said digital word;

first circuit means for providing an odd/even line signal representative of odd and even lines on said display;

second circuit means for providing an odd/even character signal representative of odd and even characters on said display;

sampling means coupled to said storing means for sequentially sampling each of said bits of said digital word at a sampling rate approximately equal to N×P in a first sequence and in a second sequence opposite to said first sequence, said first and second sequences being selected according to said odd-/even signal, with said sampling means further including a bit selecting means responsive to said odd/even line signal and said odd/even character signal for controlling which of said bits is sampled in the beginning of said first and second sampling sequences;

whereby a color signal suitable for use with a phase alternation line video display is generated at an output of said sampling means.

12. The color signal generation means of claim 11 wherein said sampling means comprises a recirculating shifter register means having P number of stages for receiving said digital word from said storing means and for circulating said digital word at said predetermined sampling rate in a first direction which corresponds to said first sampling sequence and a second direction which corresponds to said second sampling sequence and wherein said bit selecting means comprises a multiplexer having a plurality of inputs coupled to said stages of said shift register means and a multiplexer output coupled to said sampling means output.

* * * * *

United States Patent [19]

Wozniak

[11] **4,278,972**

[45] **Jul. 14, 1981**

[54] **DIGITALLY-CONTROLLED COLOR SIGNAL GENERATION MEANS FOR USE WITH DISPLAY**

[75] Inventor: **Stephen G. Wozniak**, Cupertino, Calif.

[73] Assignee: **Apple Computer, Inc.**, Cupertino, Calif.

[21] Appl. No.: **110,409**

[22] Filed: **Jan. 8, 1980**

Related U.S. Application Data

[60] Continuation of Ser. No. 910,125, May 26, 1978, abandoned, which is a division of Ser. No. 786,197, Apr. 11, 1977, Pat. No. 4,136,359.

[51] **Int. Cl.**³ **G06F 3/14**

[52] **U.S. Cl.** **340/703**; 340/725; 340/744; 340/800; 340/814; 358/17

[58] **Field of Search** 358/17, 18, 28, 10; 340/703, 744, 725, 750, 800, 801, 814

[56] **References Cited**

U.S. PATENT DOCUMENTS

3,571,513	3/1971	Ward	358/17
3,624,634	11/1971	Clark	340/703
3,771,155	11/1973	Hayashi	340/703
3,877,009	4/1975	Kanie et al.	340/703
3,878,536	4/1975	Gilliam	340/728
4,093,960	6/1978	Estes	358/10
4,119,954	10/1978	Seitz et al.	340/728
4,119,956	10/1978	Murray	340/728

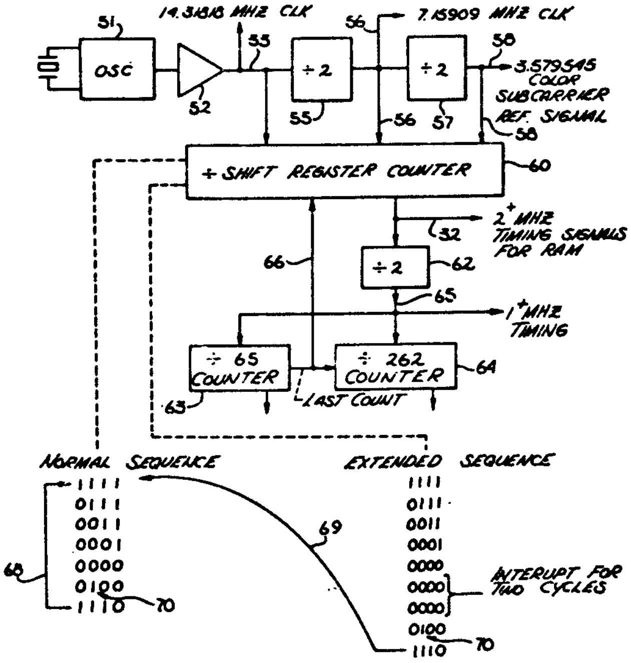
Primary Examiner—Marshall M. Curtis
Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman

[57] **ABSTRACT**

A microcomputer including a video generator and timing means which provides color and high resolution graphics on a standard, raster scanned, cathode ray tube is disclosed. A horizontal synchronization counter is synchronized at an odd-submultiple of the color subcarrier reference frequency. A "delayed" count is employed in the horizontal synchronization counter to compensate for color subcarrier phase reversals between lines. This permits vertically aligned color graphics without substantially altering the standard horizontal synchronization frequency. Video color signals are generated directly from digital signals by employing a recirculating shift register.

11 Claims, 4 Drawing Figures

Apple 2



"APPLE_PAT_4_278_972_01" 155 KB 2000-02-21 dpi: 300h x 300v pix: 1950h x 2872v

U.S. Patent Jul. 14, 1981 Sheet 1 of 2 4,278,972

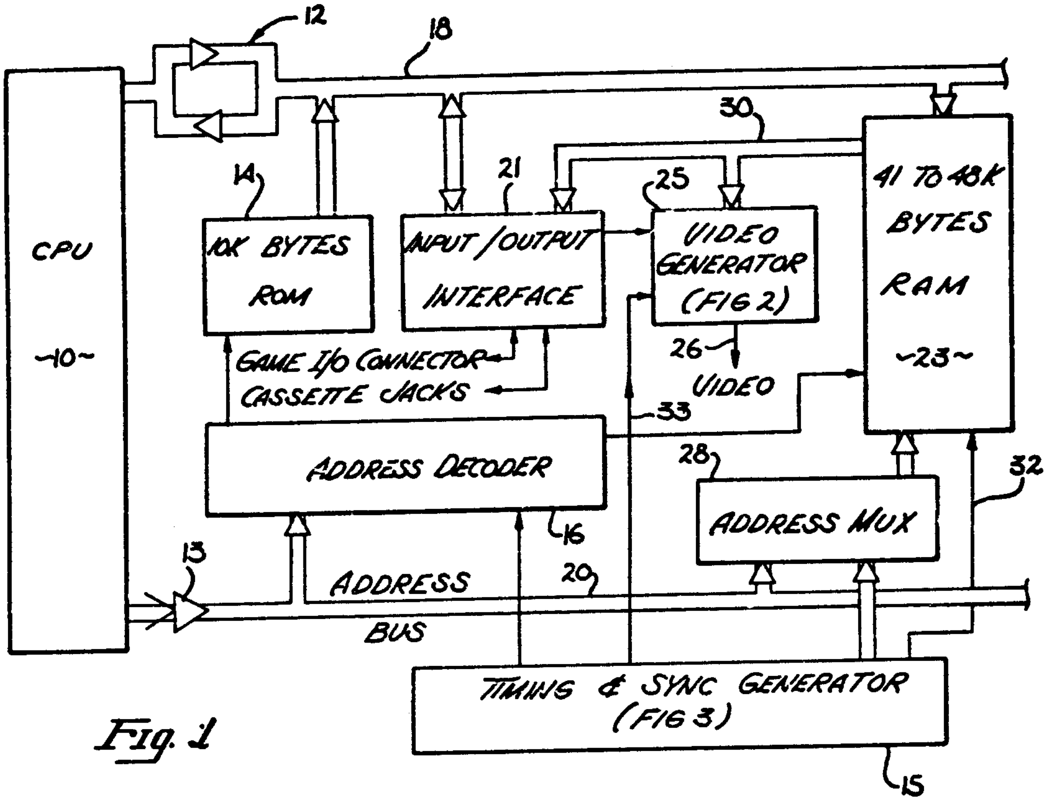


Fig. 1

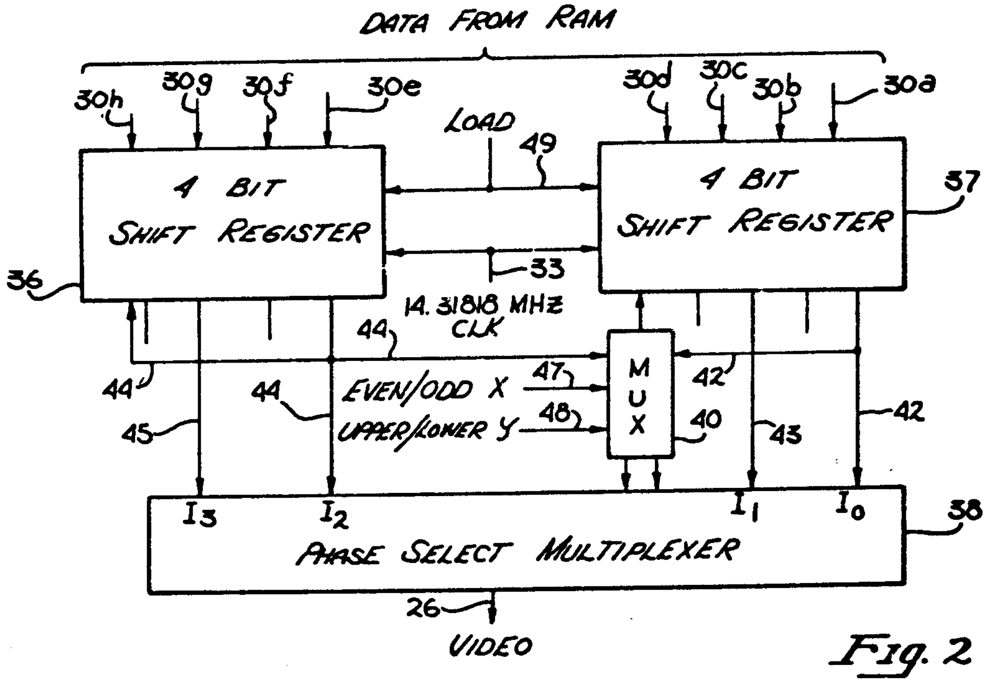


Fig. 2

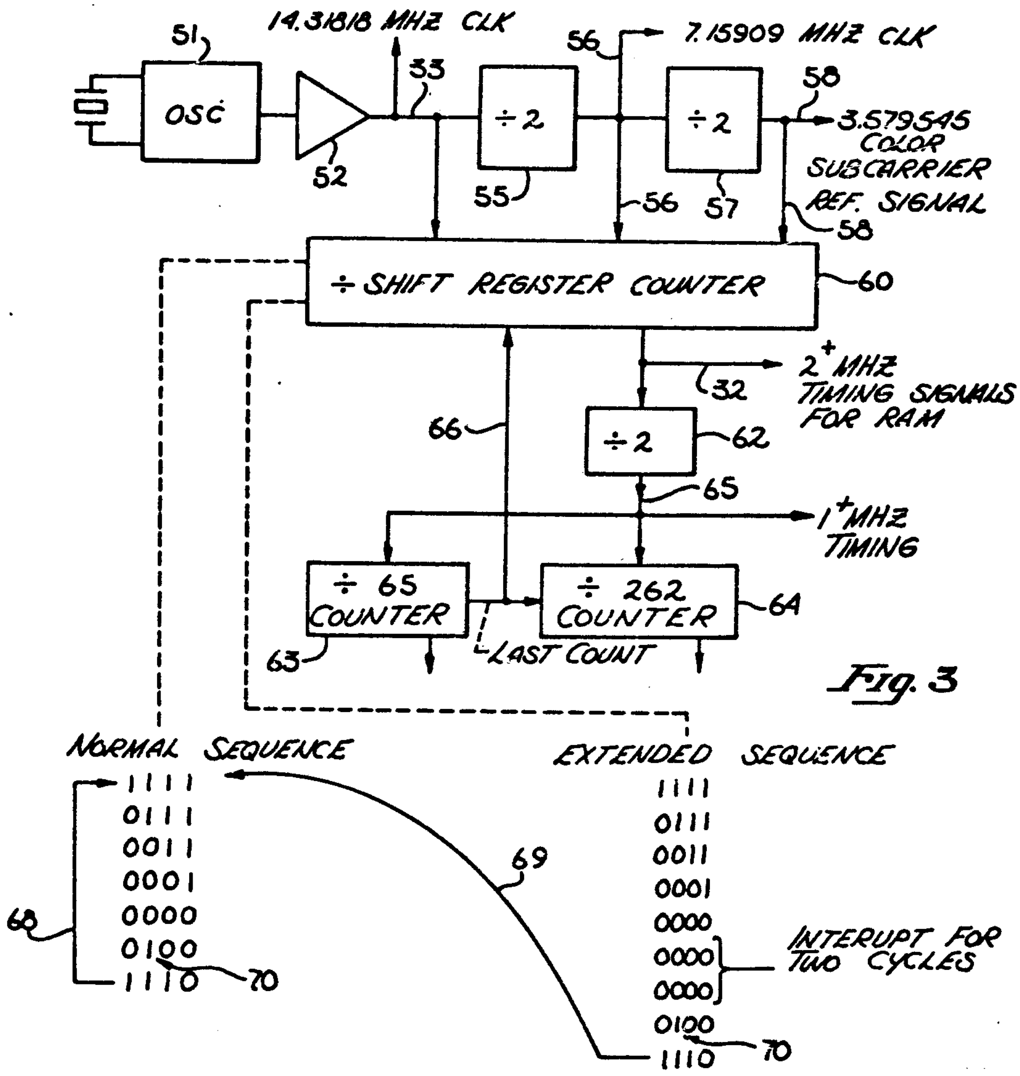


Fig. 3

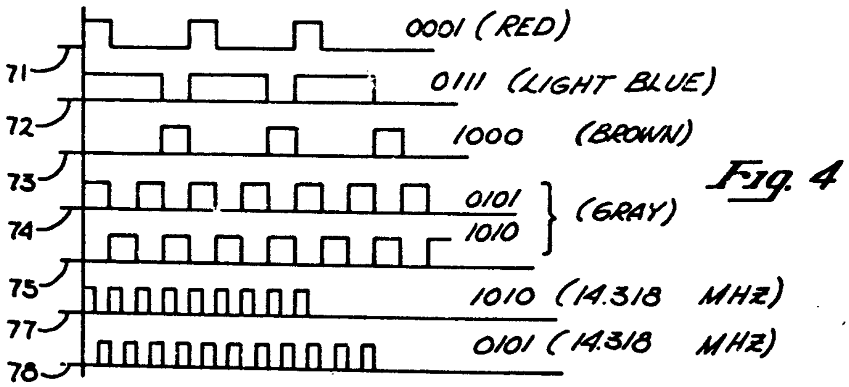


Fig. 4

“APPLE_PAT_4_278_972_03” 125 KB 2000-02-21 dpi: 300h x 300v pix: 1834h x 2842v

DIGITALLY-CONTROLLED COLOR SIGNAL GENERATION MEANS FOR USE WITH DISPLAY

This is a continuation of application Ser. No. 910,125, now abandoned, filed May 26, 1978, which is a division of application Ser. No. 786,197, filed on Apr. 11, 1977 which is now U.S. Pat. No. 4,136,359.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention is for the generation of signals for raster scanned video displays employing digital means, believed to be in Class 340-324.

2. Prior Art

With the reduced cost of large scale integrated circuits it has become possible to provide low-cost microcomputers suitable for home use. One such use which has flourished in recent years is the application of microcomputers in conjunction with video displays for games and graphic displays. Most often an ordinary television receiver is employed as the video display means. The standard, raster scanned, cathode ray tubes employed in these receivers and like displays, present unique problems in interfacing these displays with the digital information provided by the microcomputer.

In presenting color graphics it is, of course, desirable to provide high resolution lines and to avoid "ragged" lines. In a microcomputer controlled display, typically a single frequency reference source is employed to generate the color subcarrier reference signal of 3.579545 Mhz and the horizontal and vertical synchronization signals. If the frequency of the horizontal synchronization signals is to remain close to its normal frequency (i.e. 15,750 hz) the horizontal synchronization means must operate at an odd-submultiple of the color subcarrier frequency. When this occurs there is a phase reversal or phase shift of the color subcarrier reference signal when compared to color control signal between each of the lines of the display. This results in ragged vertical lines unless the color signals are changed for each line. One prior art solution to this problem has been to operate the horizontal synchronization counter at an even submultiple of the color subcarrier frequency (i.e. 15,980 hz). This deviation from the standard horizontal synchronization frequency typically requires manual adjustment of the receiver and for some receivers horizontal synchronization may be more difficult to maintain.

As will be described with the invented microcomputer, the horizontal counter operates close to its standard frequency (15,734 hz). Through use of a timing compensation means, counting in the horizontal synchronization counter is delayed to compensate for the fact that the counter operates at an odd-submultiple frequency of a color reference signal. In this manner, phase reversal of the color reference signal is eliminated and sharp graphic displays are provided without complex programming.

In many prior art microcomputer controlled displays, color information is stored as four digital bits which are used to designate green, red, blue, and high/low intensity. The color generation means generally includes a signal generator for generating the pure color signals (CW). These pure color signals are then gated and mixed in accordance with the binary state of the four bits to provide a color signal compatible with standard television receivers. Generation of the video color signal in this manner is complex and requires a substantial amount of circuitry.

The invented microprocessor includes a recirculating shift register which circulates four bits of information. In this manner video color signals are generated directly from digital information without the cumbersome generation techniques employed in the prior art.

SUMMARY OF THE INVENTION

A microprocessor for use with a video display is described. The microprocessor includes an improved timing apparatus which provides well-defined color graphics on a standard, raster scanned, cathode ray tube. A timing reference means is employed to provide a color reference signal for the video display. A horizontal synchronization means which is synchronized to the timing reference means provides horizontal synchronization signals for the display. These signals occur at a rate which is an odd-submultiple of the color reference signal frequency. The timing apparatus includes a compensation means which is coupled to both the timing reference means and the synchronization means for periodically adjusting the horizontal synchronization signals such that these signals remain in phase relationship with the color reference signal.

The microcomputer also includes a unique color signal generation means which uses a recirculating shift register. This register receives digital signals representative of color from memory and circulates this data at a predetermined rate. In this manner a color signal suitable for use with a video display is generated from the digital signals.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a general block diagram illustrating the invented microcomputer in its presently preferred embodiment.

FIG. 2 is a block diagram of the video generator employed in the microcomputer of FIG. 1.

FIG. 3 is a block diagram of the timing and synchronization generator employed in the computer of FIG. 1; and

FIG. 4 is graph illustrating several waveforms generated by the video generator of FIG. 2.

DETAILED DESCRIPTION OF THE INVENTION

A microcomputer is disclosed which is particularly suitable for controlling color graphics on a standard, raster scanned, cathode ray tube. The described microcomputer includes a video generator which generates color signals directly from digital information, and a timing means which provides well defined color graphics, particularly in the vertical direction, with complex programming.

In the following description, numerous well-known circuits are shown in block diagram form in order not to obscure the described inventive concepts in unnecessary detail. In other instances, very specific details such as frequencies, number of bits, specific codes, etc., are providing in order that these inventive concepts may be clearly understood. It will be apparent to one skilled in the art that the described inventive concepts may be employed without use of these specific details.

Referring now to FIG. 1, the microcomputer includes a central processing unit (CPU) or microprocessor 10. While any one of a plurality of commercially available microprocessors may be employed such as the

M6800 or 8080, in the presently preferred embodiment, a commercially available microprocessor, Part No. 6502, is employed. CPU 10 communicates with the data bus 18 through a bidirectional tri-state buffer 12. The CPU 10 is also coupled to the address bus 20 through a tri-state buffer 13.

The microcomputer, in its presently preferred embodiment, includes two memories. The first is a 12K (bytes) read-only memory (ROM) 14 which is coupled to the data bus 18. This ROM may be a mask programmable memory, E PROM or other read-only memory. The primary data storage for the computer comprises the random-access memory 23. In the presently preferred embodiment, this memory may contain 4K to 48K (bytes) and comprises commercially available dynamic MOS memories. The RAM 23 is coupled to the input/output interface means 21 via bus 30, the data bus 18 and the video generator 25.

The timing signals for the microcomputer are provided by the timing and synchronization generator 15. The novel portions of this generator shall be described, in detail, in conjunction with FIG. 3. This generator provides timing signals for the microcomputer, and additionally, synchronization signals for the video display. Among the signals provided by the generator 15 are 2^+ Mhz timing signals on lines 32 for the RAMs 23 and a 14.31818 Mhz signal on line 33 for the video generator 25. The timing and synchronization generator 15 also provides timing signals for the decoder 16 and for the address multiplexer 28.

The address decoder 16 receives address signals from the address bus 20 and decodes them in a well-known manner. The address decoder 16 is coupled to the ROM 14 and to the RAM 23. Address signals are also received from the bus 20 by the address multiplexer 28 which couples these signals to the RAM 23.

The input/output interface means 22 provides ports which allows the microprocessor to be electrically coupled to a cassette jack or to a connector used for receiving game input/output signals. Known buffers and timing means may be employed for this purpose.

The video generator 25 receives signals from the input/output interface means 21 and also from the RAM 23. This generator provides an output video signal on line 26. Video generator 25 shall be described, in detail, in conjunction with FIG. 2.

In the presently preferred embodiment, the entire microcomputer of FIG. 1 is fabricated on a single printed circuit board. This board includes connectors to allow the computer to be connected to a cassette playback means, or other devices. As will be appreciated, numerous well-known interconnections, driver means and other circuits employed in the microcomputer are not shown in FIG. 1. For a detailed description of circuits and interconnections which may be employed in the microprocessor of FIG. 1, including a transparent refresh cycle for the RAMs 23, see "*A CRT Terminal Using The M6800 Family*" by Roy & Morris, *Interface Age*, Volume 2, Issue 2, January 1977.

Referring now to FIG. 3, the timing and synchronization generator (timing means) includes a frequency reference source, crystal oscillator 51. The output of oscillator 51 is coupled to a buffer 52 which provides a 14.31818 Mhz signal on line 33 for the presently preferred embodiment. This signal is coupled to the video generator of FIG. 2 as will be described, and is also coupled to the shift register counter 60 and the divider 55. The divider 55 divides the 14.31818 Mhz signal by two, thereby providing a 7.15909 Mhz signal on line 56. This signal is employed by the microprocessor as a timing signal, and additionally, is employed by the shift register counter 60 as a feedback synchronization signal. The signal on line 56 is further divided by two, by divider 57, to provide the standard color subcarrier reference signal of 3.579545 Mhz on line 58. The signal on line 58 is used in an ordinary manner by the video display and also is used as a feedback synchronization signal by the shift register counter 60.

The 14.3 Mhz signal on line 33 is divided by seven, by the shift register counter 60 to provide a 2^+ Mhz signal on line 32. This signal is used by the RAMs 23 of FIG. 1. This 2^+ Mhz signal is further divided by divider 62 (divided by two) to provide a 1^+ Mhz timing signal on line 65. This 1^+ Mhz signal in addition to being employed elsewhere in the microprocessor is used by counters 63 and 64.

The "divide-by-65" counter 63 is used to provide the horizontal synchronization signals. When the maximum count is reached within the counter 63, a signal is provided on line 66 to shift register 60 and also to the vertical synchronization counter 64. The counter 64 is employed to divide this signal by 262 to provide vertical synchronization signals.

In the presently preferred embodiment, the display is divided into a 65×262 array. However, 25 of the 65 horizontal character positions are employed for blanking and 70 of the 262 lines are also employed for blanking.

It is apparent from FIG. 3 that the horizontal synchronization signals from counter 63 occur at a frequency of approximately 15,734 hz. This is very close to the standard horizontal synchronization rate of 15,750 hz. Each count of the counter 63 includes 3½ color cycle of the color subcarrier reference frequency; moreover, the total number of color cycles per line is a non-integer. As a result, the color subcarrier reference signal will be shifted 180° for each new line. Unless some corrective action is taken this will result in ragged vertical lines. As will now be described, compensation is provided by delaying the occurrence of the 1^+ Mhz timing signal once for each line by a period of time corresponding to ½ cycle of the 3.58 Mz subcarrier reference signal.

As shown in FIG. 3, the normal counting sequence for the shift counter 60 includes seven states. When the last stage of the four stage counter contains a binary-zero, a binary-one is loaded into the second stage (position 70). The first and second stages receive the output of the second stage when the last stage contains a binary-zero. Thus, the states become 1110 after the next shift, and finally the states become 1111 as indicated by path 68.

Each time a signal occurs on line 66 (every 65 cycles of the 1^+ Mhz signal) the normal sequencing within the counter 60 is altered as shown by the extended sequence of FIG. 3. When a signal occurs on line 66 and when the count of 0000 is reached, the loading of the binary-one into the second stage (position 70) is delayed for two cycles of the 14.318 Mhz clock. These two cycles correspond to 180° of the 3.58 Mhz signal. After these two cycles, a binary-one is then loaded into the second stage, followed by the loading of binary-ones into the first and third stages. As indicated by path 69, a normal counting sequence then occurs. By extending the count within counter 60 as described, compensation occurs which provides vertical color alignment from line-to-line.

Referring now to FIG. 2, the video generator 25 of FIG. 1 includes two, four bit shift registers 36 and 37. Each of these four bit shift registers is coupled to receive four bits of data on lines 30 from the RAM 23. The registers 36 and 37 receive a load signal on line 49 which causes the data on lines 30*a* through 30*h* to be shifted into the registers. The first stage of register 37 (I_0) is coupled to a multiplexer 38 by line 42. The third stage of register 37 (I_1) is also coupled to the multiplexer 38 by line 43. In a similar fashion, the first stage of the register 36 (I_2) is coupled by line 44 to the multiplexer 38, and the third stage of this register (I_3) is also coupled to the multiplexer 38 by line 45.

Line 44 is coupled to the fourth stage of register 36 in order that four bits of data within register 36 may be recirculated. (Registers 36 and 37 shift data from left to right, that is, toward their first stage). The line 42 may be selectively coupled to the fourth stage of register 37 through the multiplexer 40 in order that four bits of data within register 37 may be recirculated. Line 44 may be coupled through the multiplexer 40 to the fourth stage of the shift register 37. When this occurs, the shift registers 36 and 37 operate as a single eight bit shift register.

Control signals designated as even/odd X (line 47) and upper/lower Y (line 48) are used to control multiplexer 38. During the color graphics mode the registers 36 and 37 operate as separate registers and data is alternately selected for coupling to line 26 by multiplexer 38. The upper/lower Y signal, during the color graphics mode, allow selection of data from either register 36 or 37. The odd/even X signal then toggles the data from the selected register by alternating selecting I_0 or I_1 if register 37 is selected, or I_2 or I_3 if register 36 is selected.

During the color graphics mode as presently implemented, eight bits of color information are shifted (in parallel) into the registers 36 and 37 from the RAM 23 at a 1+ Mhz rate. This data is recirculated within registers 36 and 37 at a rate of 14.31818 Mhz by the clocking signal received on line 33. The circulation of the data bit within the registers 36 and 37 at this rate provides signals having a 3.58 Mhz component and as will be described, these signals may be readily employed for providing color signals for video display.

In the color graphics mode, as presently implemented, each of the display characters is divided into an upper and lower color rectangle. The RAM 23 provides the four bits of color data for the upper rectangles to registers 36 and for the lower rectangles to register 37. This color data for the presently preferred embodiment is coded as follows:

Red	0001	Medium Violet	0011
Pink	1011	Medium Blue	0110
Blue	0010	Medium Green	1100
Light Blue	0111	Orange	1001
Dark Green	0100	White	1111
Light Green	1110	Gray	1010
Brown	1000	Gray	0101
Yellow	1101		

When colors are coded in this manner and circulated at the rate of 14.318 Mhz in the registers, video color signals compatible with standard television receivers are produced. The resultant signal for red is shown on line 71 of FIG. 4, light blue on line 72, brown on line 73 and gray on lines 74 and 75.

Briefly referring again to FIG. 3, each count of the horizontal synchronization counter 63 corresponds to 3½ cycles of the subcarrier reference signal. Thus, a 180° phase shift occurs from character-to-character with respect to the color subcarrier reference signal. This means that the color signals must be shifted by 180° by the generator of FIG. 2, or the coding for these signals must be alternated for odd and even horizontal character positions. In the presently preferred embodiment, a 180° phase shift for the color signals is obtained by toggling between the first or third stages of the selected registers. For example, assume that the lower portion of a character is being displayed and that the color information is thus contained within register 37. Further assume that this information is being circulated, that is, line 42 couples stage 4 to stage 1 through the multiplexer 40. For even horizontal character positions, as indicated by the signal on line 47, the phase select multiplexer 38 couples the I_0 signal to line 26. For the odd positions, a 180° phase shift is obtained by selecting the I_1 signal.

During a second mode of operation the generator of FIG. 2 is used for providing high resolution graphics. In this case, eight bits of information are provided by the RAM 23 to the registers 36 and 37. For this high resolution mode line 42 is coupled to the video line 26 and the eight bits of data from RAM 23 are serially coupled to the video line 26 at the 14.318 Mhz rate. The multiplexer 40 couples line 44 to the fourth stage of register 37 to provide a single eight bit shift register. The resultant signals are shown on lines 77 and 78 of FIG. 4. The signals on lines 77 and 78 provide either a green or violet display. In the presently preferred embodiment, data changes are employed to obtain the compensation provided by the multiplexer 38 during the color graphics mode.

Thus, a microcomputer has been disclosed which is particularly suitable for controlling a color video display. The unique timing means provides well defined vertical color lines without complicated programming changes while allowing the generation of horizontal synchronization signals at close to the standard rate. The unique video generator allows the generation of color signals directly from digital signals without the complex circuitry often employed in the prior art.

I claim:

1. A digitally controlled color signal generation means for use with a color video display adapted to receive color signals having a color subcarrier reference signal of frequency N, said color signal generation means comprising:

means for generating at least one digital word which corresponds to a predetermined color, said digital word comprising a plurality of bits;

storing means for storing said digital word;

sampling means coupled to said storing means for sequentially sampling each of said bits of said digital word at a predetermined sampling rate, said sampling rate being selected such that a color signal is developed at an output of said sampling means which corresponds to said predetermined color and which has a frequency component at said frequency N,

whereby a color signal suitable for use with the video display is generated.

2. The color signal generation means of claim 1 wherein said sampling means comprises a recirculating shift register means for receiving said digital word from

said storing means and for circulating said digital word in said shift register means at said predetermined sampling rate.

3. The color signal generation means defined by claim 2 wherein said digital word is comprised of P number of bits, said shift register means comprises a recirculating shift register having P number of stages, and said predetermined sampling rate is at a frequency approximately equal to N×P.

4. The color signal generation means defined by claim 3 wherein P is equal to four.

5. The color signal generation means defined by claim 4 wherein N is approximately 3.58 MHz and said predetermined sampling rate is approximately 14.318 MHz.

6. The color signal generation means defined by claim 5 wherein said sampling means further includes phase shifting means for coupling different stages of said shift register to said output thereby allowing the selection of a phase shifted signal.

7. The color signal generation means defined by claim 6 wherein said digital word corresponding to the color red is 0001.

8. A digitally controlled color signal generation means for use with a color video display adapted to receive color signals having a color subcarrier reference signal of frequency N, said improved color signal generation means comprising:

means for generating at least one digital word which corresponds to a predetermined color, said digital word comprising P number of bits;

storing means for storing said at least one digital word;

sampling means coupled to said storing means for sequentially sampling each of said bits of said digital word at a sampling rate approximately equal to a frequency of N×P;

whereby a color signal suitable for use with the video display is developed at an output of said sampling means.

9. The color signal generation means defined by claim 8 wherein said sampling means includes phase shifting means for altering the sequence of said sequential sampling, thereby allowing the selection of a phase shifted signal at said output of said sampling means.

10. The color signal generation means defined by claim 9 wherein N is approximately equal to 3.58 MHz, P is equal to four and said sampling rate is at a frequency approximately equal to 14.318 MHz.

11. The color signal generation means defined by claim 10 wherein said sampling means comprises a recirculating shift register having four stages which receives said digital word from said storing means, with said shift register being clocked at a frequency approximately equal to 14.318 MHz and said phase shifting means is a means for coupling different stages of said shift register to said sampling means output.

* * * * *

United States Patent [19]

Sander

[11] **4,383,296**

[45] **May 10, 1983**

[54] **COMPUTER WITH A MEMORY SYSTEM FOR REMAPPING A MEMORY HAVING TWO MEMORY OUTPUT BUSES FOR HIGH RESOLUTION DISPLAY WITH SCROLLING OF THE DISPLAYED CHARACTERS**

[75] Inventor: **Wendell B. Sander**, San Jose, Calif.

[73] Assignee: **Apple Computer, Inc.**, Cupertino, Calif.

[21] Appl. No.: **150,630**

[22] Filed: **May 16, 1980**

[51] **Int. Cl.**^{3} **G06F 13/06**; G06F 3/14

[52] **U.S. Cl.** **364/200**; 340/726; 340/799

[58] **Field of Search** ... 364/200 MS File, 900 MS File; 340/726, 798, 799; 358/17

[56] **References Cited**

U.S. PATENT DOCUMENTS

3,821,730	6/1974	Carey et al.	340/799 X
3,893,075	7/1975	Orban et al.	340/799 X
3,903,510	9/1975	Zobel	340/726 X
3,980,992	9/1976	Levy et al.	364/200
4,136,359	1/1979	Wozniak	358/17
4,150,364	4/1979	Baltzer	364/900 X

FOREIGN PATENT DOCUMENTS

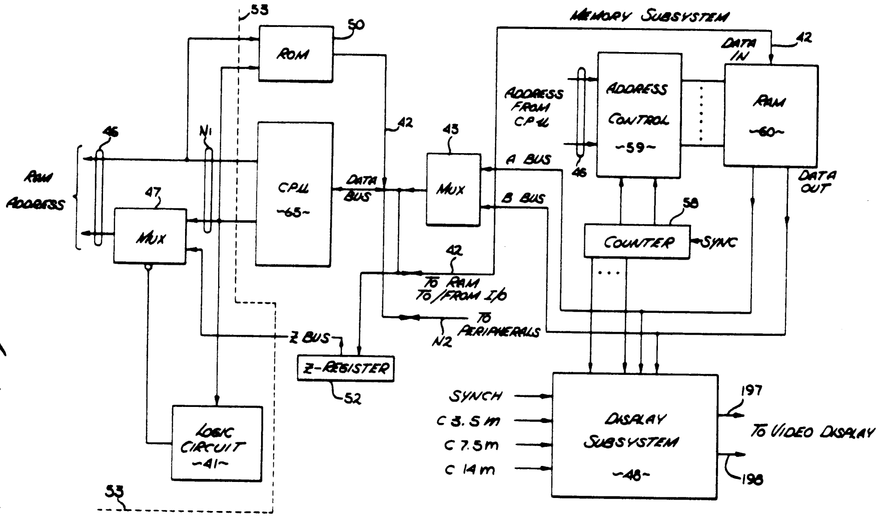
1351590	5/1974	United Kingdom .
1482819	8/1977	United Kingdom .
1496563	12/1977	United Kingdom .
1524873	9/1978	United Kingdom .

Primary Examiner—Raulfe B. Zache
Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman

[57] **ABSTRACT**

A microcomputer system with video display capability, particularly suited for small business applications and home use is described. The CPU performance is enhanced by permitting zero page data to be stored throughout the memory. The circuitry permitting this capability also provides a pointer for improved direct memory access. Through unique circuitry resembling "bank switching" improved memory mapping is obtained. Four-bit digital signals are converted to an AC chroma signal and a separate luminance signal for display modes. Display modes include high resolution modes, one of which displays 80 characters per line.

22 Claims, 9 Drawing Figures



CONTAINS BOOT ROM LISTING
(REVISION Ø)

Apple /// Computer

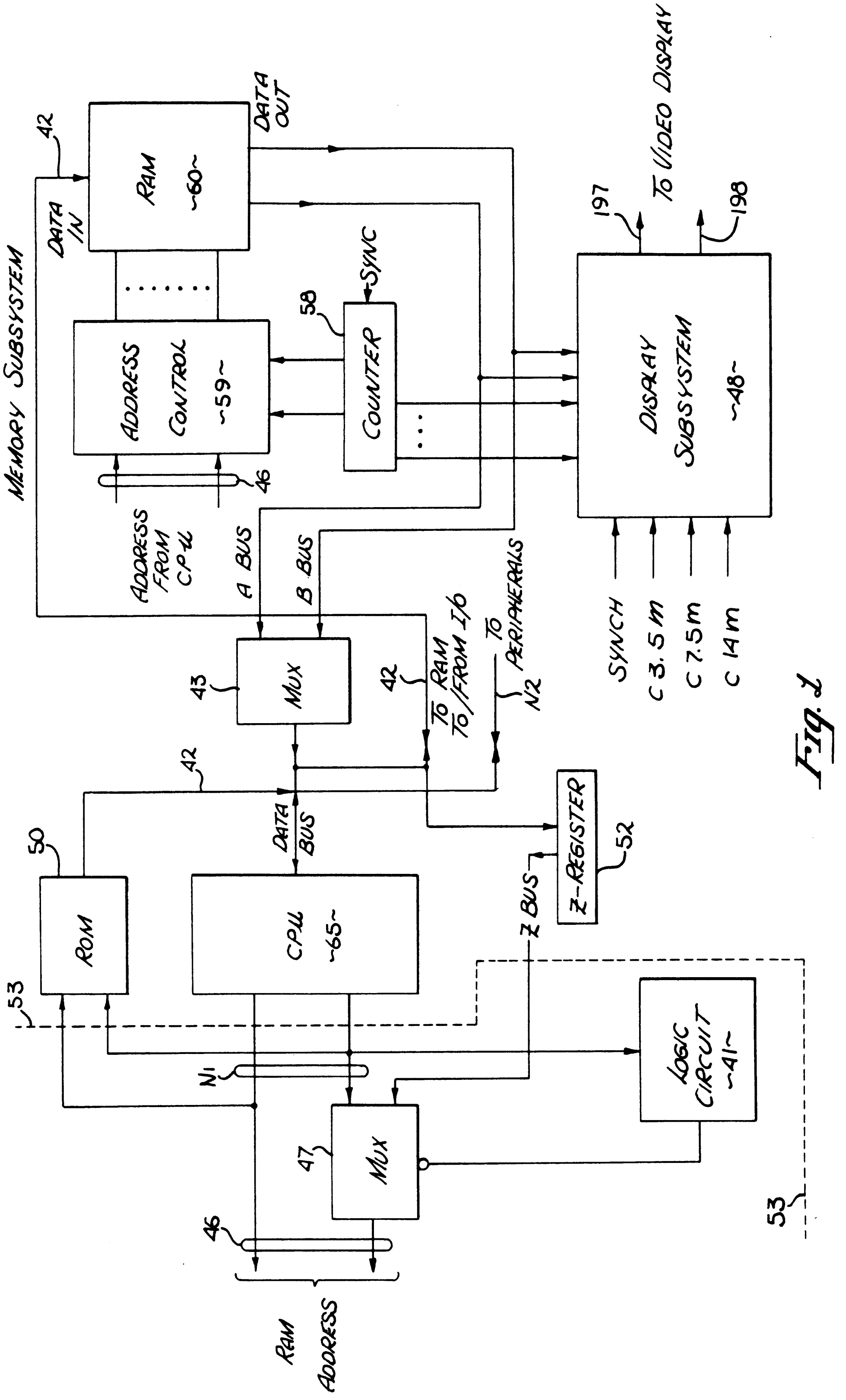

MEMORY SUBSYSTEM
42
DATA IN
RAM ~60~
DATA OUT
ADDRESS CONTROL ~59~
58
COUNTER
SYNC
46
ADDRESS FROM CPU
A BUS
B BUS
197
TO VIDEO DISPLAY
198
DISPLAY SUBSYSTEM ~48~
SYNCH
C 3.5 M
C 7.5 M
C 14 M
43
MUX
42
TO RAM
TO/FROM I/O
TO PERIPHERALS
N2
42
DATA BUS
50
ROM
CPU ~65~
Z-REGISTER
52
Z BUS
53
N1
LOGIC CIRCUIT ~41~
47
MUX
46
53
RAM ADDRESS

Fig. 1

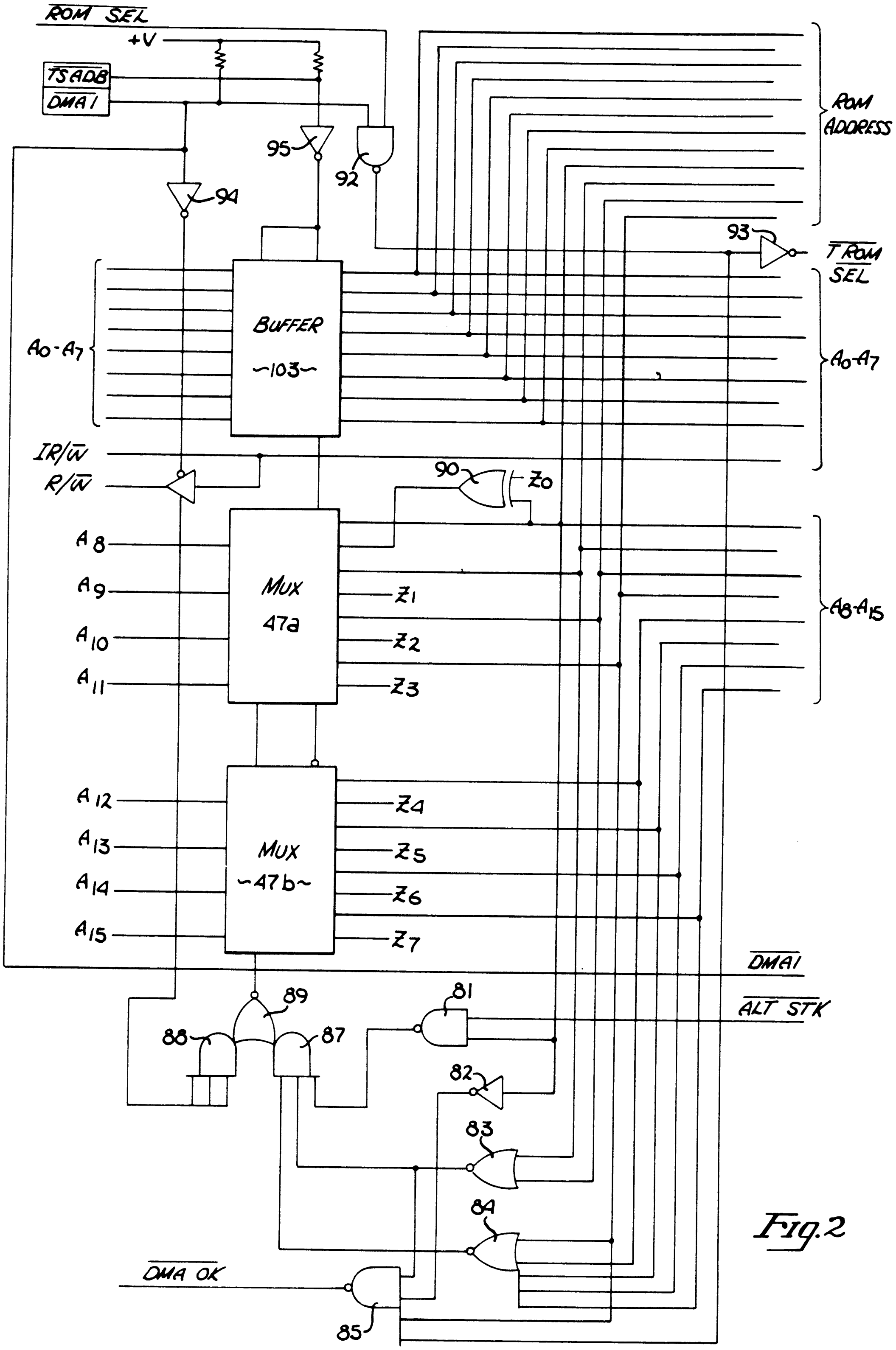

ROM SEL
+V
TSADB
DMA1
95
92
94
93
TROM SEL
ROM ADDRESS
A0-A7
BUFFER ~103~
A0-A7
IR/W
R/W
90
Z0
A8
A9
A10
A11
MUX 47a
Z1
Z2
Z3
A8-A15
A12
A13
A14
A15
MUX ~47b~
Z4
Z5
Z6
Z7
DMA1
89
81
ALT STK
88
87
82
83
84
DMA OK
85

Fig. 2

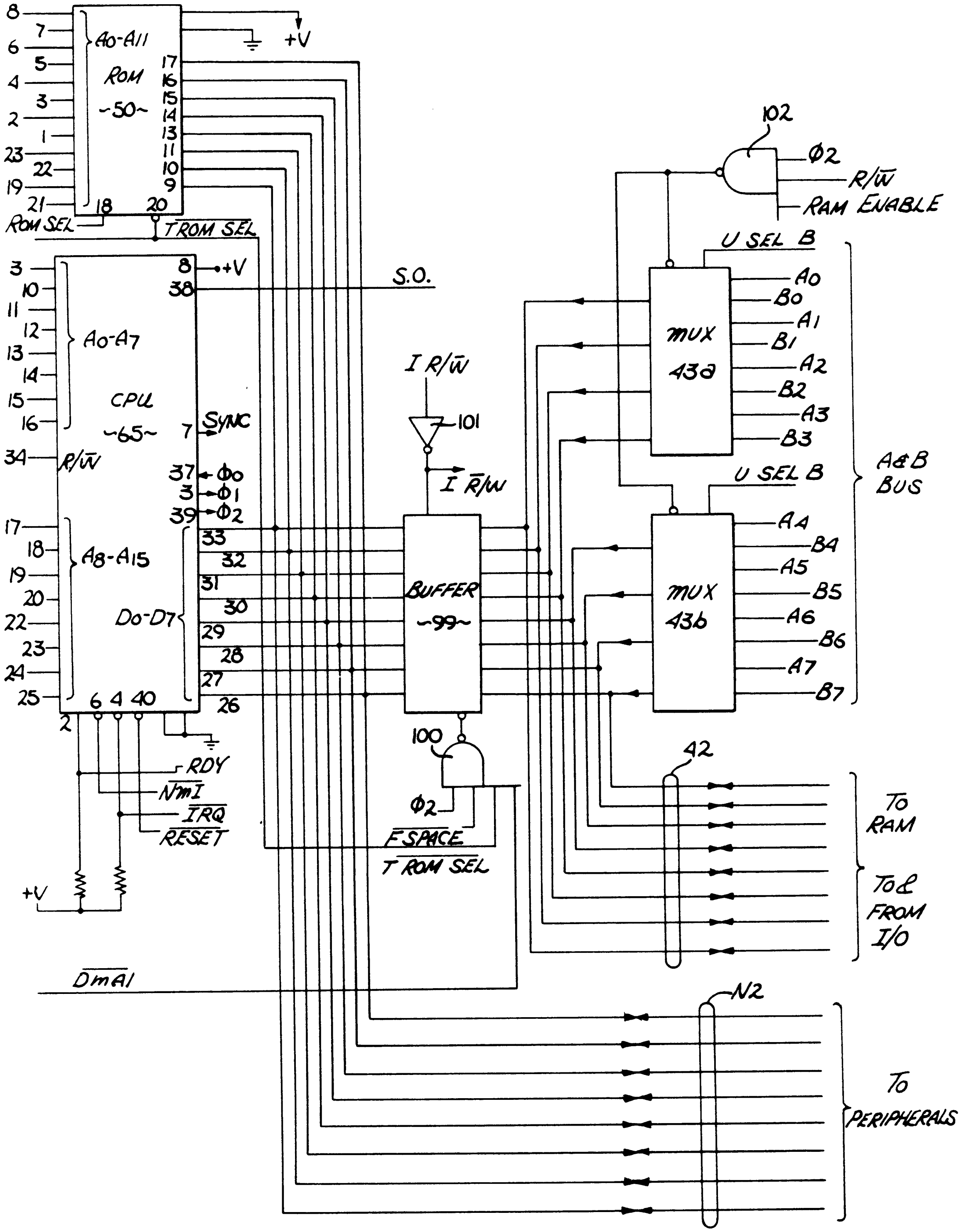

A0-A11
ROM
~50~
+V
17
16
15
14
13
11
10
9
18
20
ROM SEL
T ROM SEL
8
7
6
5
4
3
2
1
23
22
19
21
102
Φ2
R/W
RAM ENABLE
U SEL B
A0
B0
A1
B1
A2
B2
A3
B3
MUX
43a
MUX
43b
A4
B4
A5
B5
A6
B6
A7
B7
A&B BUS
S.O.
CPU
~65~
A0-A7
A8-A15
D0-D7
SYNC
R/W
34
37 Φ0
3 Φ1
39 Φ2
38
33
32
31
30
29
28
27
26
6 4 40
RDY
NMI
IRQ
RESET
+V
I R/W
101
BUFFER
~99~
100
FSPACE
T ROM SEL
42
To RAM
To & FROM I/O
DMA1
N2
To PERIPHERALS

Fig. 3

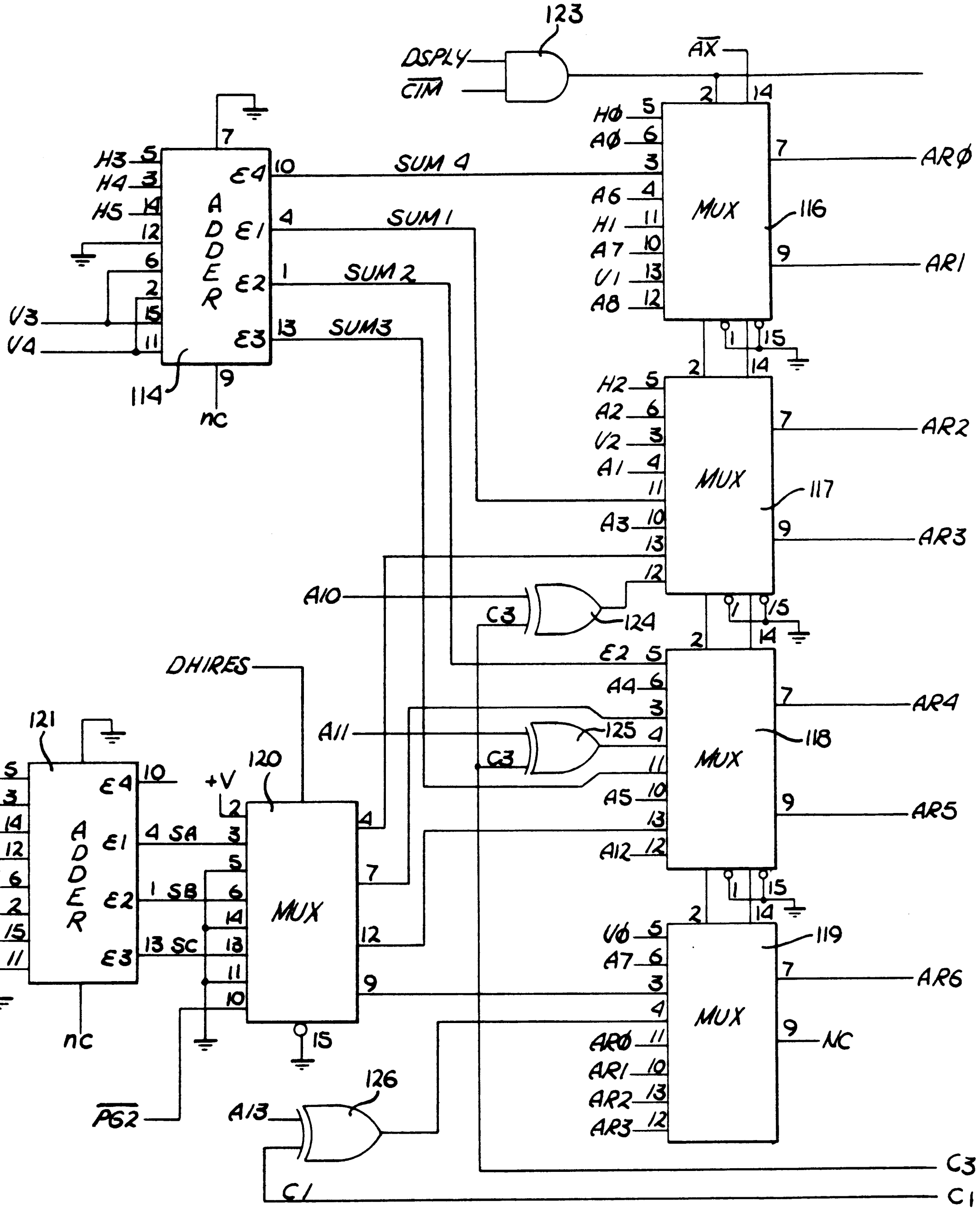
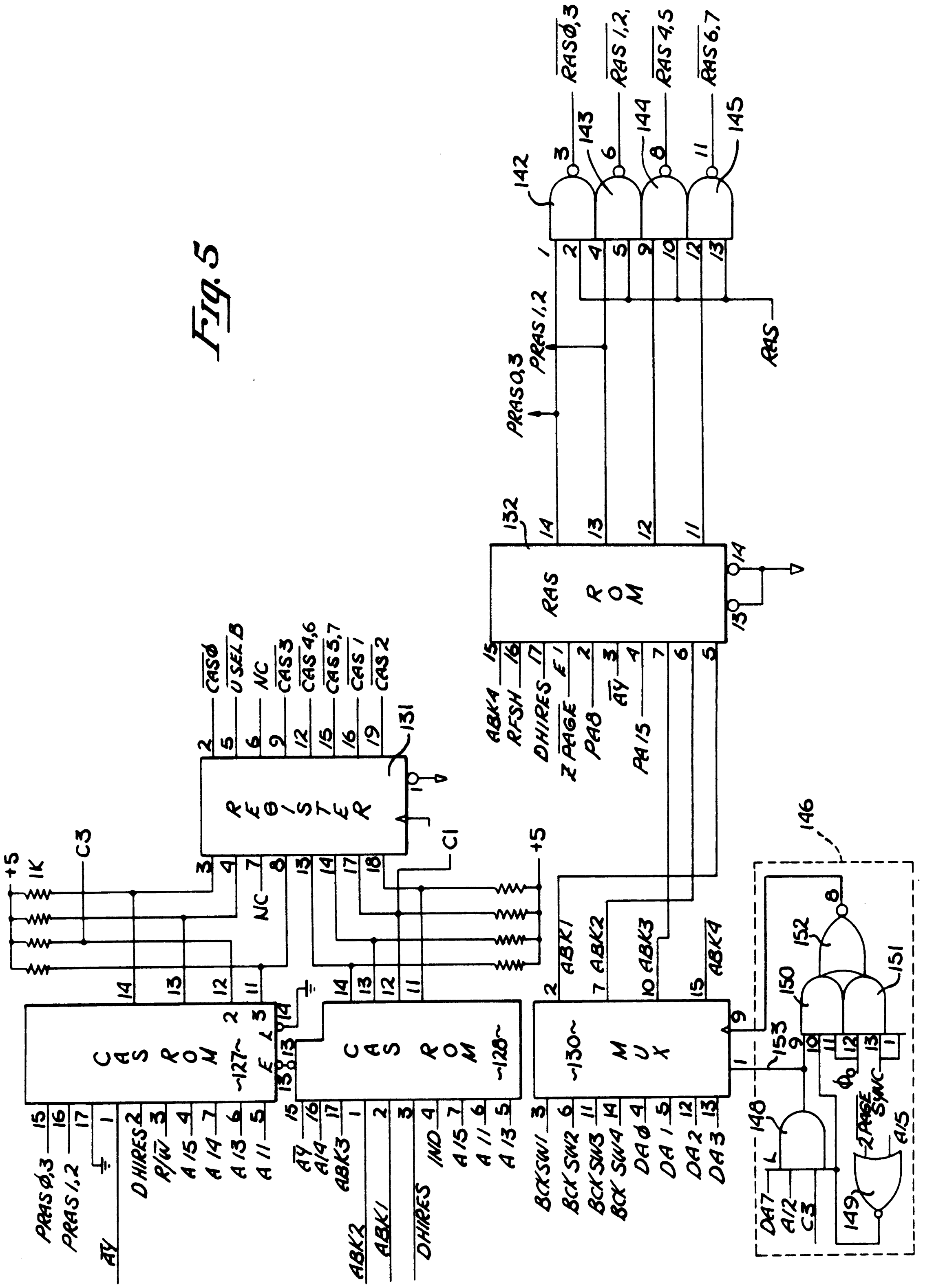

123
DSPLY
CIM
AX
2
14
H3
H4
H5
5
3
14
12
6
2
15
11
7
10
4
1
13
9
nc
ADDER
E4
E1
E2
E3
V3
V4
114
SUM 4
SUM 1
SUM 2
SUM3
H0
A0
A6
H1
A7
V1
A8
MUX
116
AR0
AR1
H2
A2
V2
A1
A3
117
AR2
AR3
A10
C3
124
E2
A4
A5
A12
A11
125
118
AR4
AR5
DHIRES
121
VA
VB
VC
VA1
VB1
VC1
+V
120
4 SA
1 SB
13 SC
PG2
A13
126
C1
V6
A7
AR0
AR1
AR2
AR3
119
AR6
NC
C3
C1

Fig. 4

U.S. Patent May 10, 1983 Sheet 5 of 8 4,383,296



"APPLE_PAT_4_383_296_06" 387 KB 2000-02-27 dpi: 600h x 600v pix: 3525h x 5214v

U.S. Patent May 10, 1983 Sheet 6 of 8 4,383,296

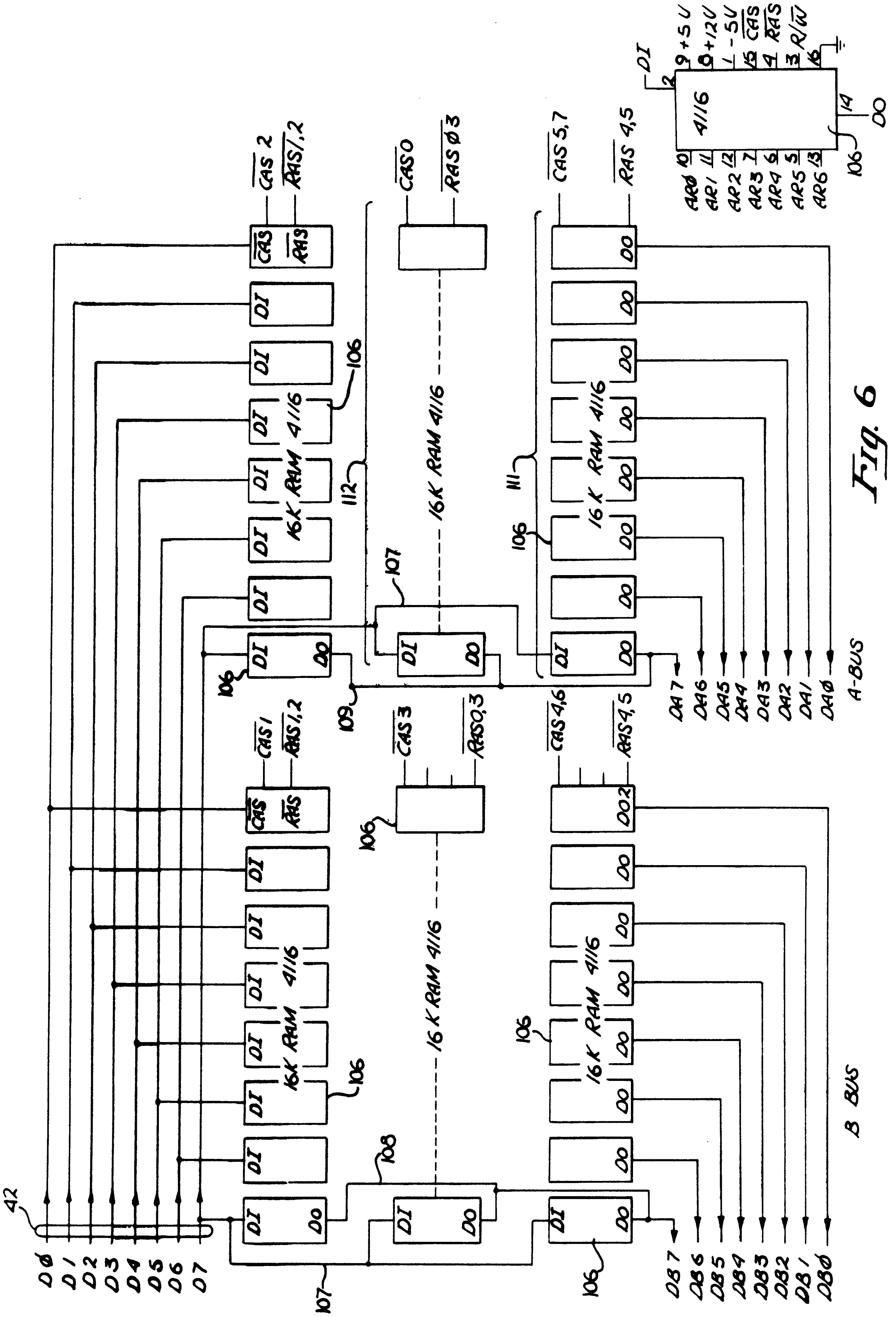


Fig. 6

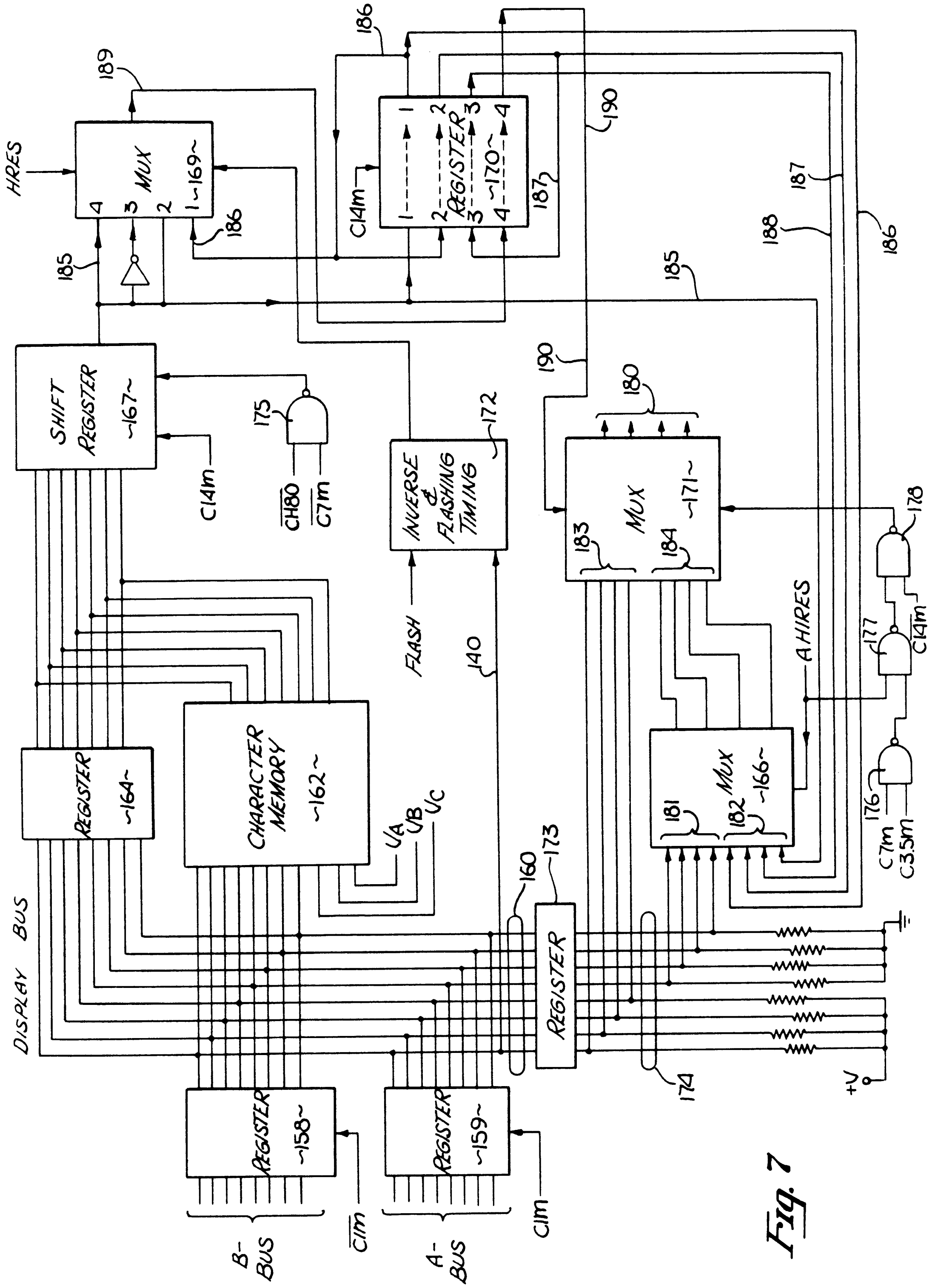

HRES
MUX
~169~
1
2
3
4
189
186
REGISTER
~170~
C14m
187
190
185
188
SHIFT REGISTER
~167~
C14m
175
$\overline{CH80}$
$\overline{C7m}$
INVERSE & FLASHING TIMING
172
MUX
~171~
180
183
184
178
AHIRES
177
C14m
MUX
~166~
181
182
176
C7m
C3.5m
CHARACTER MEMORY
~162~
VA
VB
VC
REGISTER
~164~
DISPLAY BUS
FLASH
140
160
173
REGISTER
174
+V
REGISTER
~158~
REGISTER
~159~
$\overline{C1m}$
C1m
B-BUS
A-BUS

Fig. 7

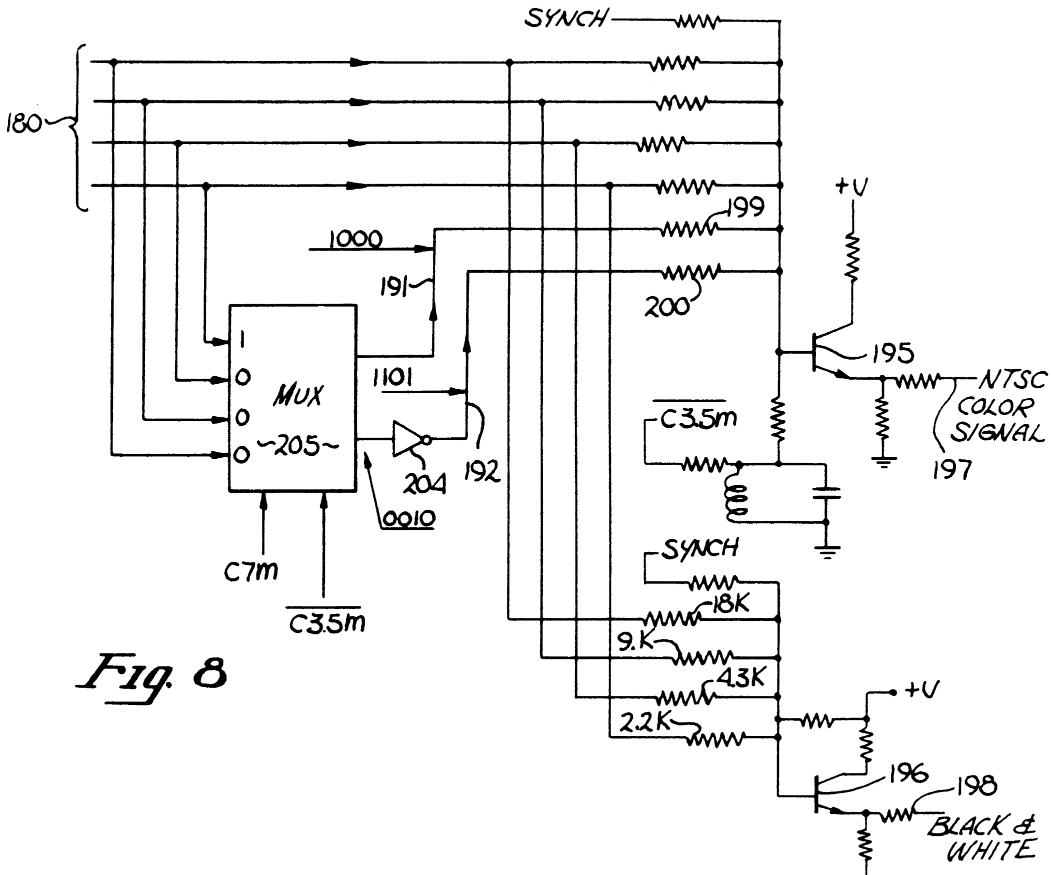

SYNCH
180
1000
199
191
200
+V
MUX
~205~
1
0
0
0
1101
195
NTSC
COLOR
SIGNAL
197
C3.5m
204
192
0010
C7m
C3.5m
SYNCH
18K
9.1K
4.3K
2.2K
+V
196
198
BLACK &
WHITE

Fig. 8

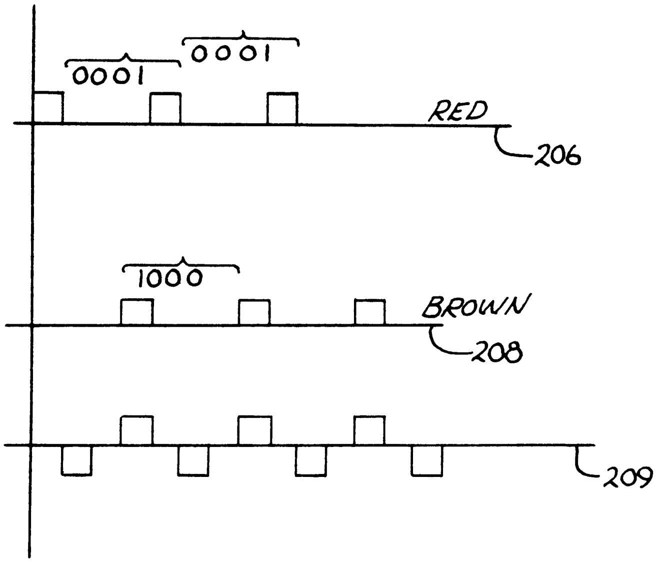

0001
0001
RED
206
1000
BROWN
208
209

Fig. 9

COMPUTER WITH A MEMORY SYSTEM FOR REMAPPING A MEMORY HAVING TWO MEMORY OUTPUT BUSES FOR HIGH RESOLUTION DISPLAY WITH SCROLLING OF THE DISPLAYED CHARACTERS

BACKGROUND OF THE INVENTION

The invention relates to the field of digital computers, particularly microcomputers, having video display capabilities.

Prior Art

In the last few years, there has been rapid growth in the use of digital computers in homes by hobbyists, for small business and for routine engineering and scientific application. For the most part, these needs have been met with self-contained, relatively inexpensive microcomputers or microprocessors with essential peripherals, including disc drives and with relatively easy to manage computer programs. The design for computers for these needs requires considerable ingenuity since each computer must meet a wide range of applications and because this market is particularly cost conscious.

A home or small business computer must, for example, operate with a number of different program languages, including those requiring relatively large memories, such as Pascal. The computer should interface with a standard raster scanned display and provide a wide range of display capabilities, such as high density alpha-numeric character displays needed for word processing in addition to high resolution graphics displays.

To meet these specialize computer needs, generally requires that a relatively inexpensive microprocessor be used and that the capability of the processor be enhanced through circuit techniques. This reduces the overall cost of the computer by reducing, for example, power needs, bus structures, etc. Another important consideration is that the new computers be capable of using programs developed for earlier models.

As will be seen, the presently described microcomputer is ideally suited for home and small business applications. It provides a wide range of capabilities including advanced display capabilities not found in comparable prior art computers.

The closest prior art computer known to applicant is commercially available under the trademark, Apple-II. Portions of that computer are described in U.S. Pat. No. 4,136,359.

SUMMARY OF THE INVENTION

A digital computer which includes a central processing unit (CPU) and a random-access memory (RAM) with interconnecting address bus and data bus is described. One aspect of the present invention involves the increased capability of the CPU by allowing base page or zero page data to be stored throughout the memory. Alternate stack locations and an improved direct memory access capability are also provided by the same circuitry. Detection means are used for detecting a predetermined address range such as the zero page. This detection means causes a special register (Z-register) to be coupled into the address bus. The contents of this Z-register provide, for example, a pointer during direct memory access, or alternate stack locations for storing data normally stored on page one.

The memory of the invented computer is organized in an unusual manner to provide compatibility with the 8-bit data bus and yet provide high data rates (16-bits/MHz) needed for high resolution displays. A first plurality of memory devices are connected to a first memory output bus; these memory devices are also connected to the data bus. The memory includes a second plurality of memory devices which are also connected to the data bus; however, the outputs of these second devices are coupled to a second output memory bus. First switching means permit the first and second memory buses to be connected to the display for high data rate transfers. Second switching means permit either one of the memory buses to be connected to the data bus during non-display modes.

The addressing capability of the memory is greatly enhanced not only through bank switching, but through a novel remapping which does not require the CPU control associated with bank switching. In effect, the "unused" bits from one of the first and second memory buses are used for remapping purposes. This mode of operation is particularly useful for providing toggling between two separate portions of the memory.

The display subsystem of the described computer generates video color signal in a unique manner. A 4-bit color code as used in the prior art, is also used with the described display subsystem. However, this code is used to generate an AC chrominance signal and a separate DC luminance signal. This provides enhanced color capability over similar prior art color displays.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a block diagram showing the major components and subsystems of the invented and described microcomputer system.

FIGS. **2** and **3** together show the central processing unit (CPU) and the architecture associated with this CPU, particularly the address bus and data bus.

FIG. **2** is a circuit diagram primarily showing the address bus and the logic means associated with this bus.

FIG. **3** is a circuit diagram primarily showing the data bus and its interconnection with the memory buses (A bus and B bus), bootstrap read-only memory, and input/output ports.

FIGS. **4**, **5** and **6** show the memory subsystem.

FIG. **4** is a circuit diagram primarily showing the circuitry for selecting between address signals from the address bus and display counter signals.

FIG. **5** is a circuit diagram primarily showing the generation of various "select" signals for the memory devices.

FIG. **6** is a circuit diagram showing the organization of the random-across memory and its interconnection with the data bus and memory output buses.

FIGS. **7** and **8** illustrate the display subsystem of the invented computer.

FIG. **7** is a circuit diagram showing the circuitry for generating the digital signals used for the video display.

FIG. **8** is a circuit diagram of the circuitry used to convert the digital signals to analog video signals.

FIG. **9** is a graph of several waveforms used to describe a prior art circuit and the circuit of FIG. **8**.

DETAILED DESCRIPTION OF THE INVENTION

A microcomputer system capable of driving a raster scanned video display is disclosed. In the following description, numerous specific details such as specific part numbers, clock rates, etc, are set forth to provide a thorough understanding of the present invention. However, it will be obvious to one skilled in the art that the inventive concepts described in this patent may be practiced without these specific details. In other instances, well-known circuits have been shown in block diagram form in order not to obscure the present invention in unnecessary detail.

Referring first to FIG. **1**, in general the described computer includes a central processing unit (CPU) **65**, its associated data bus **42**, address bus **46**, a memory subsystem and a display subsystem **58**.

The address bus **46** from the CPU is coupled to the memory subsystem to permit the selection of locations in memory. Some of the address signals pass through a multiplexer **47**. For some modes of operation, signals from a register **52** are coupled through the multiplexer **47** onto the bus **46**. The register **52** is identified as the Z-register and is coupled to the multiplexer **47** by the Z bus. The general description of the multiplexer **47** and its control by the logic circuit **41** are described in detail in conjunction with FIG. **2**. In general, the circuitry shown to the left of the dotted line **53** is included in FIG. **2** while the CPU **65**, memory **50**, data bus **42** and multiplexer **43** are shown in detail in FIG. **3**.

The address bus N1 is coupled to the read-only memory **50**. The output of this memory is coupled to the computer's data bus **42**. The read-only memory (ROM) **50**, as will be described, stores test routines, and other data of a general bootstrap nature for system initialization.

The data bus **42** couples data to the random-access memory (RAM) **60** and to and from I/O ports. This bus also couples data to the Z-register **52** and other commonly used registers not illustrated. The data bus **42** receives data from the RAM **60** through the A bus and B bus which are selected by multiplexer **43**. The peripheral Bus N2 is used, as is better illustrated in FIG. **3**, for coupling to peripherals.

The memory subsystem is shown in detail in FIGS. **4**, **5** and **6**. The address control means which receives addresses on bus **46**, makes the final selection of memory locations within the RAM **60**. Bank switching, addressing for display purposes, scrolling and other memory mapping is controlled by the address control means **59** as will be described in greater detail in conjunction with FIGS. **4** and **5**. The PAM **60** is shown in detail in FIG. **6**. The counter **58** which is sychronized with the horizontal and vertical display signals, provides signals both to the address control means **59** and to the display subsystem **48**.

The display subsystem receives data from the RAM **60** on the A bus and B bus and converts these digital signals to video signals which control a standard raster scanned display. A standard NTSC color signal is generated on line **197** and a black and white video signal on line **198**. The same signals used to generate these video signals can be used to generate separate red, green, blue (RGB) video signals. The display subsystem **48** receives numerous timing signals including the standard color reference signal shown as 3.5 MHz (C3.5M). This subsystem is described in detail in FIGS. **7** and **8**.

COMPUTER ARCHITECTURE

In the presently preferred embodiment, the CPU **65** (microprocessor) employed with the described computer is a commercially available component, the 6502A. This 8-bit processor (8-bit data bus) which has a 16-bit address bus is shown in FIG. **3** with its interconnections to the remainder of the computer. The pin number for each interconnection is shown adjacent to the corresponding line. In many cases, the nomenclature associated with the 6502A (CPU **65**) is used in this application. For example, pin **6** receives the nonmaskable interrupt signal ($\overline{NMI}$), and pin **4** is coupled to receive the interrupt request signal ($\overline{IRQ}$). Some of the signals employed with the CPU **65**, which are well-known in the art, and which are not necessary for the understanding of the present invention are not described in detail in this application, such as the various synchronization signals and clocking signals. The address signals from the CPU **65** are identified as A_0–A_7 and A_8–A_{15}. The data signals associated with the CPU **65** are shown as D_0–D_7. As will be apparent to one skilled in the art, the inventive concepts described in this application may be employed with other microprocessors.

Referring now to FIGS. **2** and **3**, the general architecture, particularly the architecture associated with the CPU **65** can best be seen. The address signals A_0–A_7 are coupled to a buffer **103** by the bus shown primarily in FIG. **2**. These address signals are also coupled to the ROM **50**. The signals A_0–A_7 after passing through the buffer **103** are coupled to the memory subsystem. The address signals A_8–A_{15} (higher order address bits) are coupled through lines shown in FIG. **2** to the multiplexers **47*a*** and **47*b***. The contents of the Z-register **52** of FIG. **1** is also connected to the multiplexers **47*a*** and **47*b*** through the Z-bus (Z_1–Z_7). The multiplexers **47*a*** and **47*b*** allow the selection of either the signals A_8–A_{15} from the CPU **65** or the contents of the Z-register (Z_1–Z_7) for addressing the RAM **60**. The output of these multiplexers are shown as A_8–A_{15}; this designation is used even when the Z-bus is selected. Note in the case of the Z_0 signal, this signal is coupled to the multiplexer **47*a*** through the exclusive OR gate **90** for reasons which are explained later. The address signals A_8–A_{11} are also coupled to the ROM **50**, thus the signals A_0–A_{11} are used for addressing the ROM **50**. The signals A_8–A_{15} are connected to the logic circuit shown in the lower left-hand corner of FIG. **2**; this logic circuit corresponds to the logic circuit **41** of FIG. **1**.

The input and output data signals from the CPu **65** are coupled by a bidirectional bus to the bidirectional buffer **99** (FIG. **3**). This buffer is selectively disabled by gate **100** to allow the output of ROM **50** to be communicated to CPU **65** and during other times not pertinent to the present discussion. The direction of flow through the buffer **99** is controlled by a read/write signal coupled to the buffer through inverter **101**. Data from the CPU **65** is coupled through the buffer **99** and bus **42** to the RAM **60** or to I/O ports. Data from the RAM **60** is communicated to CPU **65** or bus N2 from the A bus and B bus through the buffer **99**. The **4** lines of the A bus and 4 lines of the B bus are coupled to the multiplexer **43*a***. Similarly, the other 4 lines of the A and B buses are coupled to the multiplexer **43*b***. Multiplexers **43*a*** and **43*b*** select the 8 lines of the A bus or B bus and communicate the data through to buffer **99** and bus **42**. These multiplexers are selectively disabled (for example, during writing) by gate **102**. As will be described later, the 16 lines of the A bus and B bus permits the reading of 16-bits from the RAM at one time. This provides a data rate of 16-bits/MHz which is necessary, for example, for an 80 character per line display. The data is loaded into the RAM **60**, 8-bits at a time.

The ROM **50**, as mentioned, stores test programs, data needed to initialize various registers, character generation data (for RAM **162** of FIG. **7**) and other related data. Specific programs employed in the presently preferred embodiment of the computer are set forth in Table 1. The ROM **50** is selected by control signals coupled to its pins **18** and **20**, identified as signals ROM SEL and $\overline{\text{T ROM SEL}}$. Any one of a plurality of commercially available read-only memories may be used for the ROM **50**. In the presently preferred embodiment, commercially available Part No. SY2333 is used.

Referring now to this logic circuit (lower left-hand corner of FIG. **2**), the NAND gate **81** receives the address signal A_8 and also the alternate stack signal identified as $\overline{\text{ALT STK}}$. The output of this gate provides one input to the AND gate **87**. The A_8 signal is also coupled through the inverter **82** to one input terminal of the NAND gates **85** and **86**. The address signals A_9 and A_{10} are coupled to the input terminals of the NOR gate **83**. The output of this gate is coupled to one input terminal of the NAND gates **85** and **86** and the AND gate **87**. The address signals A_{11}-A_{15} are coupled to the input terminals of the NOR gate **84**. The signal A_{11} is also coupled to an input terminal of the NAND gate **85**.

The outputs of the AND gates **87** and **88** (through NOR gate **89**), controls the multiplexers **47***a* and **47***b*. When the output of gate **89** is low the Z-bus is selected, otherwise the address signals from the CPU **65** are selected.

The logic circuit above-described, along with the Z-bus and Z-register provide enhanced performance for the computer. First, this circuit permits the zero page or base page data to be stored throughout the RAM **60** rather than just on zero page. Secondly, this circuit enables addressing of alternate stack locations (other than page one). Lastly, this circuit through the Z-register provides a RAM pointer for direct memory access (DMA).

Assume for purposes of discussion that the CPU **65** is addressing the zero page of memory. That is, the higher order address bits A_8–A_{15} are all zeros. The zeros for A_9–A_{15} are detected by the gates **83** and **84**. If all the inputs to these gates are zeros, the outputs of these gates are high which condition is communicated to the gate **87**. A_8 which is also low, insures that the output of gate **81** will be high. Thus, all the inputs to gate **87** are high, causing the signal at the output of the gate **89** to drop. When this occurs, the Z-bus is selected. Instead of all the binary zeros from the CPU being coupled to the main memory (RAM **60**), the contents of the Z-register form part of the address for the memory. Therefore, even though the CPU **65** has selected the zero page, nonethelessdata may be written into or from any location of RAM **60** (including the zero page). This enhances the performance of the CPU, since for example, the time consumed in shifting data to and from a single zero page is minimized.

Normally, the CPU **65** selects page one for stack locations. This occurs when A_8 is high and A_9–A_{15} are low. Assume first that the alternate stack locations have not been selected. Both inputs to gate **81** are high and its output is low. The low input to the gate **87** prevents the selection of the Z-bus. Thus, for these conditions the address signals A_0–A_7 select stack locations on page one.

Next assume that page one has been selected by the CPU and that the $\overline{\text{ALT STK}}$ signal is low, indicating the alternate stack locations are to be selected. (A flag is set by the CPU to change the $\overline{\text{ALT STK}}$ signal). Since the $\overline{\text{ALT STK}}$ signal is low and A_8 is high, a high output occurs from the gate **81**. All the inputs to gates **83** and **84** are low, therefore, high outputs occur from both these gates. The conditions of gate **87** are met, causing a high output from this gate and lowering the output from the gate **89**. The Z-bus is thus selected by the multiplexers **47***a* and **47***b*. This allows the contents of the Z-register to be used as alternate locations. Nonzero page locations are assured by inverting A_8. The exclusive OR gate **90** acts as a selective inverter. If A_8 is high and Z_0 is low, then A_8 at the output of the multiplexer **47***a* will be low. Note that during zero page selection when A_8 is low, the Z_0 signal is directly communicated through gate **90** to the output of multiplexer **47***a*.

Thus, the logic circuits along with the $\overline{\text{ALT STK}}$ signal allows alternate stack locations to be selected through the Z-bus. This further enhances the performance of the CPU which would otherwise be limited to page one for stack locations.

The logic circuit of FIG. **2** is also used along with the Z-register to provide a pointer during direct memory access (DMA). Assume that direct access to the computer's memory is required by a peripheral apparatus. To initiate the DMA mode the CPU provides an address between **F800** and R8FF. Through a logic circuit not illustrated in FIGS. **2** and **3**, the $\overline{\text{ROM SEL}}$ signal is brought low for addresses between **F000** and FFFF. This signal is communicated to gate **93** and causes the output of gate **92** to rise ($\overline{\text{DMA 1}}$ is high at this time). This rise in potential is communicated to one input of the gate **85**. Additionally, gate **85** senses that the address bits A_8, A_9 and A_{10} are low. This information is coupled to gate **85** through the inverter **82** and the NOR gate **83** as high signals. Also the fact that A_{11} is high is directly communicated to gate **85**. Thus, with the address between **F800** and **F8FF** the $\overline{\text{DMA OK}}$ signal drops in potential. This is sensed by the peripheral apparatus which in turn causes the $\overline{\text{DMA 1}}$ signal to drop and provides a ready signal to the CPU **65**. With the completion of this handshake, data may begin to be transferred to the RAM.

The $\overline{\text{DMA 1}}$ signal through gate **93** and inverter **93** forces the $\overline{\text{T ROM SEL}}$ signal low. This signal in addition to being communicated to the ROM **50**, is coupled to the buffer **99** through gate **100**, disabling this buffer (during the reading of ROM **50**). Also, the ready signal causes the CPU to come to a hard stop. Importantly, the $\overline{\text{DMA 1}}$ signal, after passing through the inverter **94** and the gates **88** and **89**, assures the selection of the Z-register. The contents of the Z-register are fixed and provide a pointer to a page in the RAM.

Under the above conditions, the CPU increments the lower 8-bits of the address signal. The ROM **50** furnishes the instructions for incrementing the address, specifically SBC #1 and BEQ. The peripheral apparatus provides the data or receives the data in synchronization with the CPU operation. The peripheral also furnishes a read/write signal to indicate which operation is to occur. Data is then written into RAM via bus **N2** and bus **42**, or read from RAM via the A and B buses and bus **N2**.

Importantly, with the above DMA arrangement, addresses from the peripheral apparatus are not necessary and the Z-register is used to provide a pointer to a page in RAM **60**.

MEMORY SUBSYSTEM

The memory sybsystem shown in FIG. **1** as the address control means **59** and RAM **60** is illustrated in detail in FIGS. **4**, **5** and **6** as mentioned. In FIGS. **4** and **5**, the memory control means is shown, while in FIG. **6** the memory devices and their organization are illustrated. The address control means of FIGS. **4** and **5** receives the address signals from the CPU **65** (A_0–A_{15}), the count in the vertical and horizontal counters (counter **58** of FIG. **1**) which are used during display modes, control signals from the CPU and other signals. In genreal, this control means develops the address signals which are coupled to the RAM of FIG. **6** including the column address and row address signals, commonly referred to as $\overline{CAS}$ and $\overline{RAS}$. Other related functions are also shown in FIGS. **4** and **5**, such as the circuitry which provides display scrolling, indirect RAM addressing and memory mapping.

The CPU of FIG. **3** provides a 16-bit address for addressing the memory. Under ordinary circumstances this address limits the memory capacity to 64K bytes. This size memory is insufficient in many applications, as for example, to effectively use the Pascal program language. As will be described in greater detail, the address control means of FIGS. **4** and **5** enable the use of a memory having a 96K byte or 128K byte capacity. One well-known technique which is used with the present invention for increasing this capacity is bank switching; this switching occurs under the contol of the CPU. In addition, the address control means uses a unique indirect addressing mode which provides the benefits of bank switching, however, this mode does not require CPU control. This greately enhances CPU operation with the larger memory (as will be described) when compared to the CPU controlled bank switching.

Referring first to FIG. **6**, the RAM configuration is illustrated for a capacity of 96K bytes. The memory is organized into six rows, each of which includes eight 16K memory devices such as rows **111** and **112**. In the presently preferred embodiment, Part No. 4116 MOS dynamic RAMs are used. (The pin designations and signal designations refer to this memory device.) Obviously, other memory devices may be employed.

Input data to these memory devices **106** is provided from the bus **42**. Each line in the bus **42** is connected to the data input terminal of one device **106** in each row. The interconnection of this bus with each of the memory devices is not shown in FIG. **6** in order to overcomplicate this drawing. By way of example, however, line **107** connects the data bit D7 to the data input terminal of one of the memory devices in each of the six rows.

Three rows of devices **106** have their output terminals coupled to the A bus, and three rows are similarly coupled to the B bus. By way of example, line **108** connects three output terminals of devices **106** to the DB7 line of the B bus while line **109** connects three output terminals of the devices **106** to the DA7 line of the A bus.

The described memory devices **106** are each organized as a 16KX1 memory. Thus, each device receives a 14-bit address which is time multiplexed into two, 7-bit addresses. This multiplexing occurs under the control of the $\overline{CAS}$ and $\overline{RAS}$ signals as is well-known. The lines coupling the address signals to each of the devices in FIG. **6** are not illustrated. However, in the lower right-hand corner of FIG. **6**, the various signals applied to each device (including the address signals), along with the corresponding pin numbers are shown. Other circuitry not illustrated is the refresh control circuitry which operates in a well-known manner in conjunction with the $\overline{CAS}$, $\overline{RAS}$ and address signals to refresh the dynamic devices.

Each row of memory devices **106** receives a unique combination of $\overline{CAS}$ and $\overline{RAS}$ signals. For example, row **111** receives $\overline{CAS}$ **5**, **7** and RAS **4**, **5**; similarly, row **112** receives $\overline{CAS}$ **0** and $\overline{RAS}$ **0**, **3**. The generation of these $\overline{CAS}$ and $\overline{RAS}$ signals is described in conjunction with FIG. **5**. These signals (along with the 14-bit address signals) permit the selection of a single 8-bit location in the 96K byte memory (for writing) and also the selection (for reading) of 16-bit locations.

The memory of FIG. **6** may be expanded to a 128K byte memory by using 32K memory devices, such as Part No. 4132. In this case, four rows of eight, 32K memory devices are used with each row receiving two $\overline{CAS}$ and $\overline{RAS}$ signals.

Before reviewing FIG. **4**, a general understanding of the organization of the display is helpful. The display, during certain modes, is organized into 80 horizontal segments and 24 vertical segments for a total of 1920 blocks. 11-bits of the counter **58** of FIG. **1** are used as part of the address signals for the memory to access data for displaying during these modes. These counter signals are shown in FIG. **4** as H_0–H_5 and V_0–V_4. During other display modes each horizontal segment is further divided into 8 segments (e.g. for displaying 80 alpha numeric characters per line). This requires 3 additional vertical timing signals shown as V_A, V_B and V_C in FIGS. **4** and **7**.

Often in the prior art, two separate counters are used to supply the timing/address signals for accessing a memory when the data in the memory is displayed. The count in one counter represents the horizontal lines of the screen (vertical count) and the other the position along each line, (horizontal or dot count). In many prior art displays the most significant bit of the dot counter is used to increment the line counter. Data in memory intended for display is mapped with a one-to-one correlation to the counts in these counters. In another prior art system (implemented in the Apple-II computer sold by Apple Computer, Inc.) this one-to-one correlation is not used. Rather, to conserve on circuitry, a single counter is employed and a more dispersed mapping is used in the memory. (Note that where a maximum horizontal count of 80 is used, this number cannot be represented by all ones in a digital counter and thus the vertical counter cannot easily be incremented by the most significant bit in the horizontal counter.) Since this more dispersed mapping technique is part of the prior art and not critical to an understanding of the present invention, it shall not be described in detail. However, the manner in which it is implemented shall be discussed in conjunction with the adder **114** of FIG. **4**. For purposes of discussion, the signals from the counter **58** of FIG. **1** are designated as either vertical (V) or horizontal (H).

Referring now to FIG. **4**, the selection of either the counter signals on the address signals from the CPU is made by the multiplexers **116**, **117**, **118** and **119**. Each of these commercially available multiplexers (Part No. 153) couples one of four input lines to an output line. There are eight inputs to multiplexers **116**, **117** and **118** and the outputs of these multiplexers provide the address signals for the memories (AR0 through AR5). The multiplexer **119** has four inputs on its pins **3**, **4**, **5**, **6** and provides a single output on pin **7**, the AR6 address signal. (The signals supplied to pins **11**, **12** and **13** of multiplexer **119** are for clamping purposes only.)

The $\overline{AX}$ signal is applied to the pin **14** of each of the multiplexers. The signal on this line and the signal applied to pin **2**, determines which of the four inputs is coupled to each of the outputs of the multiplexers. The $\overline{AX}$ signal is a RAM timing signal for clocking the first 7 bits and second 7 bits of the multiplexed 14-bit address applied to each of the memory devices **106**. The other control signal to the multiplexers is developed through the AND gate **123**. The inputs to this gate are the display signal (DSPLY) which indicates that the computer is in a display mode and a clocking signal, specifically a 1MHz timing signal ($\overline{C1M}$). The output of the AND gate **123** determines whether the address signals from the CPU or the signals associated with the counter **58** of FIG. **1** are selected.

Assume for purposes of discussion that the display has not been selected, and thus, the output of gate **123** is low. The $\overline{AX}$ signal then selects for pin **7** of multiplexer **116** first the address signal A_0 and then A_6. Likewise, each of the multiplexers selects an address signal (except for those associated with exclusive OR gates **124** and **125** which shall be discussed). If the display signal is high and an output is present from the gate **123**, then, by way of example, the $\overline{AX}$ signal first causes the H_1 signal and then the V_1 signal to be connected to the AR1 address line. Similarly, signals corresponding to the vertical and horizontal count are coupled to the other address lines during display modes.

The adder **114** is an ordinary digital adder for adding two 4-bit digital nibbles and for providing a digital sum signal. A commercially available adder (Part No. 283) is employed. The carry-in terminal (pin **7**) is grounded and no carry-outs occur since one of the inputs (pin **12**) is grounded. The adder sums the digital signal corresponding to H_3, H_4 and H_5 with the digital signal corresponding to V_3, V_4, V_3, V_4. The resultant sum signal is coupled to the multiplexers **116**, **117** and **118** as illustrated. the summing of these horizontal and vertical counter signals is used to provide the more dispersed mapping as previously discussed.

The adder **121** is identical to adder **114** and is coupled to sum the three least significant vertical counter bits from the counter **58** (FIG. **2**) with the signals VA1, VB1 and VC1. The sum is selected by the multiplexer **120** during the high resolution display modes and also during scrolling as will be described. These sum signals are coupled to the multiplexers **117**, **118** and **119**. During the low resolution display modes, the multiplexer **120** couples ground signals or the page **2** signal ($\overline{PG2}$) to the multiplexers **117**, **118** and **119**. (The $\overline{PG2}$ signal is used for special mapping purposes, not pertinent to the present invention.) During the high resolution modes when the display is not being scrolled, the VA1, VB2 and VB3 signals are at ground potential and thus no summing occurs within adder **121** and the VA, VB and VC signals are coupled directly to the multiplexers **117**, **118** and **119**.

The address signals A_{10}, A_{11}, and A_{13} from the CPU are coupled to the multiplexers **117**, **118** and **119**, respectively, through exclusive OR gates **124**, **125**, and **126**, respectively. The other input terminals to gates **124** and **125** receive the C_3 signal, while the other input terminal of the gate **126** receives the C_1 signal. (The development of the C_1 and C_3 signals is illustrated in FIG. **5**.) The gates **124**, **125** and **126** provide mapping compensation within the memory. As the computer and memory are presently implemented, the sequence in which the various portions of the display are generated is not the same as the sequence in which the data is removed from memory for display. These gates provide compensating addresses and, in effect, cause a remapping so that the proper sequence is maintained when data is read from the memory for the display. These gates are shown to provide a complete disclosure of the presently preferred embodiment, however, they are not critical to the present invention.

In operation, the circuitry of FIG. **4**, as mentioned, selects the address signals which are applied to each of the memory devices, either from the CPU or counter if the display mode is selected. It should be noted that not all of the address bits from the CPU are coupled to the multiplexers **116** through **119**. Some of these address bits, as will be described in conjunction with FIG. **5**, are used to develop the various $\overline{CAS}$ and $\overline{RAS}$ signals and thus select different rows within the memory of FIG. **6**.

The scrolling operation which is used is somewhat unusual in that each line of the display is separately moved up (line-by-line) with one line of data in memory being moved for each frame. This technique provides a uniform, esthetically pleasing, scroll. Scrolling the screen one line per frame can be achieved by moving all the data in the memory into a new position for each frame. This would be very time consuming and impractical. With the described technique, only one-eighth of the data in the memory is moved for each new frame.

Referring to the adder **121**, as mentioned, the signals V_A, V_B V_C are the three least significant vertical counter bits from the counter **58**. These bits or counts, by way of example, represent the 8 horizontal lines of each character. In adder **12**, a 3-bit digital signal, VA1, VB1 and VC1, is added to the count from counter **58**. This 3-bit signal is constant during each frame, however, it is incremented for each new frame.

During a first frame, 000 is added to the vertical count. During a second frame, 001 is added; and during a third frame, 010 is added, and so on. By adding this digital signal to the count from counter **58**, the addresses to the memory are changed in the vertical sense. During the first frame when 000 is added, the display remains unaffected. During the next frame, when 001 is added to the vertical count, instead of first displaying the first line of a character, the second line of each character is displayed at the top of each character space and each subsequent line of the character is likewise moved up one line. If data in memory is not moved, the first line of the character would appear at the bottom of each character. Note when 001 is added to 111 from the counter, 000 results. Thus, the first line of characters would be addressed when the beam is scanning the eighth line of characters. To prevent this, the data corresponding to the first line of each character is moved in memory for this frame. The first line of one character is moved up and becomes the bottom line of the character directly above it. When 010 is added, the process is again repeated. For example, the third line of each character is first displayed in each character space and the second line of each character is moved up to become the bottom line of the character directly above it. This process is repeated to scroll the data. The movement of data in memory is controlled by the CPU in a well-known manner.

Thus, through use of adder **121**, an even, continuous scroll is obtained without moving all the data in memory for each frame. Rather, only $\frac{1}{8}$th of the data is moved for each frame.

Referring now to FIG. **5**, the circuitry used to extend the addressing from the CPU is illustrated. In general, the $\overline{\text{CAS}}$ signals are generated by the ROMs **127** and **128**. The $\overline{\text{RAS}}$ signals are generated by the ROM **132**. The multiplexer **130** allows the selection of either the bank switching signals, or the unique indirect addressing mode when "bank switching" occurs without direct commands from the CPU.

The CAS ROM **127** receives as an address the following signals: PRAS,ϕ**3**, PRAS **1**,**2** $\overline{\text{AY}}$, DHIRES, R/$\overline{\text{W}}$, A_{11}, A_{13}, A_{14}, and A_{15}. As the PRASϕ, **3** and PRAS **1**, **2** represent the RAS signals being used. These signals are high when the respective RAS signal is active.

As previously mentioned, the AY signal is high for display modes and the DHIRES signal is high for high resolution display modes. The CAS ROM **128** receives as address signals the ABK1, ABK2, and ABK3 signals and also DHIRES, $\overline{\text{AY}}$, IND, A_{11}, A_{13}, A_{14}, and A_{15}.

The ROMS **127** and **128** are programmed to implement the following equations.

$$\begin{array}{l}\overline{PCASO}=(PRASO,3\cdot(\overline{DHIRES}\cdot\overline{AY}+AY\cdot(\overline{A15}\cdot\overline{A1}-\\ \overline{4}\cdot\overline{A13}\cdot\overline{A11}\cdot\overline{R/WN}+\overline{A15}\cdot\overline{A14}\cdot\overline{A13}\cdot R/WN+A1-\\ 5\cdot\overline{A14}\cdot A13+A15\cdot A14\cdot A13\cdot\overline{A11})))\end{array}\qquad(1)$$

$$\begin{array}{l}\overline{PCAS2}=(DHIRES\cdot\overline{AY}+AY\cdot(\overline{ABK1}\cdot\overline{ABK2}\cdot\overline{ABK}-\\ \overline{3}\cdot\overline{IND}+ABK1\cdot ABK2\cdot ABK3)\cdot(\overline{A15}\cdot A14-\\ )+AY\cdot IND\cdot\overline{ABK1}\cdot\overline{ABK2}\cdot\overline{ABK3}\cdot\overline{A15}\cdot(A14\cdot A13-\\ +A14\cdot\overline{A13}))\end{array}\qquad(2)$$

$$\begin{array}{l}PCAS3=PRASO.\\ 3\cdot(\overline{DHIRES}\cdot\overline{AY}+AY\cdot(\overline{A15}\cdot\overline{A14}\cdot\overline{A13}\cdot A11+A1-\\ 5\cdot A14\cdot\overline{A13}\cdot\overline{A11}+A15\cdot A14\cdot\overline{A13})))\end{array}\qquad(3)$$

$$\begin{array}{l}\overline{PCAS4,6}=(AY\cdot\overline{IND}\cdot\overline{ABK3}\cdot\overline{A15}\cdot(ABK1\cdot\overline{ABK}-\\ \overline{2}+ABK1)\cdot ABK2)\\ \cdot(\overline{A14}\cdot A13+A14\cdot\overline{A13})+AY\cdot IND\cdot\overline{ABK3}\cdot(\overline{ABK}-\\ \overline{2}\cdot\overline{ABK1}\cdot A15+\overline{ABK2}\cdot ABK1+ABK2\cdot\overline{ABK}-\\ \overline{1}\cdot\overline{A15})\cdot\overline{A14}+AY\cdot\overline{IND}\cdot ABK1\cdot ABK2\cdot\overline{ABK3}\cdot(\overline{A1}-\\ \overline{5}\cdot\overline{A14}\cdot A13+A15\\ \cdot\overline{A14}\cdot\overline{A13})+AY\cdot IND\cdot\overline{ABK3}\cdot ABK2\cdot(\overline{A15}\cdot ABK-\\ 1+A15\cdot\overline{ABK1})\cdot(\overline{A14}\cdot\overline{A13}+A14\cdot\overline{A13}))\end{array}\qquad(4)$$

$$\begin{array}{l}PCAS5,\\ 7,=(AY\cdot\overline{IND}\cdot\overline{ABK3}\cdot(ABK1\cdot\overline{ABK2}+\overline{ABK1}\cdot\\ ABK2)\cdot(\overline{A15}\cdot A14\cdot A13+A15\cdot\overline{A14}\cdot\overline{A13})+AY\cdot I-\\ ND\cdot\overline{ABK3}\cdot(\overline{ABK2}\cdot\overline{ABK1}\cdot A15+\overline{ABK2}\cdot ABK-\\ 1+ABK2\cdot\overline{ABK1}\cdot\overline{A15})\cdot A14+AY\cdot\overline{IND}\cdot ABK-\\ 1\cdot ABK2\cdot\overline{ABK3}\cdot(\overline{A15}\cdot A14)+AY\cdot IND\cdot\overline{ABK}-\\ \overline{3}\cdot ABK2\cdot(\overline{A15}\cdot ABK1+A15\cdot\overline{ABK1})\cdot(\overline{A14}\cdot A13-\\ +A14\cdot\overline{A13}))\end{array}\qquad(5)$$

In effect, these ROMs are programmed to allow selection of predetermined rows in the memory, based on the address signals A_{10}, A_{13}, A_{14} and A_{15}, (ignoring for a moment the contribution of the $\overline{\text{RAS}}$ signals and the other signals appearing in the equations).

The outputs of the $\overline{\text{CAS}}$ ROMs **127** and **128** are coupled to the register **131**. Register **131** is a commercially available register which permits the enabling of output signals (Part No. 374). During accessing of the memory the various $\overline{\text{CAS}}$ signals ($\overline{\text{CAS 0}}$ through $\overline{\text{CAS 7}}$) are coupled to the memory of FIG. **6** to permit selection of the appropriate memory devices. The signal USELB from $\overline{\text{CAS}}$ ROM **127** through register **131** selects either the A bus or B bus. This signal is coupled to the multiplexers **43***a* and **43***b* of FIG. **3**.

During normal operation, the multiplexer **130** selects the bank switching signals BCKSW **1** through BCKSW **4**. These four signals (or alternatively four signals from the A bus) provide four of the inputs (address signals) to the ROM **132**. The other inputs to this ROM are the DHIRES, Z PAGE, PA**8**, PA**15**, RFSH (refresh), and $\overline{\text{AY}}$ signals. These address signals select the RAS **0**, **3**; RAS **1**, **2**; RAS **4**, **5** and RAS **6**, **7** signals. The ROM **132** is programmed to implement the following four equations.

$$\begin{array}{l}PRAS0,3=\overline{AY}\cdot(\overline{DHIRES}+RFSH)+(ABK4\cdot(Z\\ Page\cdot\overline{PA8}))+ABK1\cdot ABK2\cdot ABK3)\cdot AY\end{array}\qquad(6)$$

$$\begin{array}{l}PRAS1,2=\overline{AY}\cdot(DHIRES+RFSH)+AY\cdot(\overline{ABK}-\\ \overline{1}\cdot\overline{ABK2}\cdot\overline{ABK3}\cdot(ABK4\cdot(ZPAGE\cdot\overline{PA8})\cdot\overline{PA15}-\\ )+ABK1\cdot ABK2\cdot ABK3)+AY\cdot\overline{ABK3}\cdot(\overline{ABK}-\\ \overline{1}\cdot ABK2\cdot ABK4\cdot(ZPAGE\cdot\overline{PA8})\cdot PA15+ABK-\\ 1\cdot ABK2\cdot(ABK4\cdot(ZPAGE\cdot\overline{PA8})\cdot\overline{PA15})\end{array}\qquad(7)$$

$$\begin{array}{l}PRAS4,5=RFSH\cdot\overline{AY}+AY\cdot\overline{ABK2}\cdot\overline{ABK3}\cdot(\overline{ABK}-\\ \overline{1}\cdot ABK4\cdot(ZPAGE\cdot\overline{PA8})\cdot PA15+ABK1\cdot(ABK-\\ 4\cdot(ZPAGE\cdot\overline{PA8})\cdot\overline{PA15})\end{array}\qquad(8)$$

$$\begin{array}{l}PRAS6,7=RFSH\cdot\overline{AY}+AY\cdot\overline{ABK3}\cdot(ABK1\cdot\overline{ABK}-\\ \overline{2}\cdot ABK4\cdot(ZPAGE\cdot\overline{PA8})\cdot PA15+\overline{ABK1}\cdot ABK-\\ 2\cdot(ABK4\cdot(ZPAGE\cdot\overline{PA8})\cdot\overline{PA15})\end{array}$$

Thus, the bank switching signals (along with the other input signals to ROM **132**) select predetermined rows in memory in conjunction with the $\overline{\text{CAS}}$ signals.

The output signals of the ROM **132** are coupled through the NAND gates **142**, **143**, **144** and **145** to the memory. The other input terminals of these gates receive the RAS timing signal. In this manner, the output signals of the ROM **132** are clocked through the gates **142** through **145** to provide the $\overline{\text{RAS}}$ signals shown in FIGS. **5** and **6**.

An important feature to the presently described computer is provided by the circuitry shown within the dotted line **146**. The AND gate **148** receives, at its input terminals, the DA7, A_{12}, and C_3 signals. The NOR gate **149** receives the zero page and A_{15} signal. The output of gate **149** provides one input to the gate **148** and also one input to the AND gate **150**. The output of gate **148** provides another input signal to gate **150** and this signal (line **153**) is one of the two control signals coupled to the multiplexer **130**. The AND gates **150** and **151** also receive a SYNC signal and the ϕ_0 signal. The output of the gates **150** and **151** are coupled to a NOR gate **152** with the output of the gate **152** (line **154**) coupled to the other control terminal of the multiplexer **130**.

The gates **150**, **151** and **152** effectively form a clock for multiplexer/register **130** (multiplexer **130** is a commercial part, Part No. **399**, which effectively is a register/multiplexer). This selects the lower four input lines to the multiplexer **130**. However, because of the synchronization signal applied to gate **151**, the multiplexer **130** selects the bank switching signals each time an OP code is fetched by the CPU.

To understand the operation of the circuit shown within the dotted line **146** it should be recalled that the memory of FIG. **6** provides a 16-bit output. As mentioned, during certain display modes, 16-bits/msec. are needed for display purposes. In nondisplay modes, only 8-bits are required, particularly for interaction with the CPU. When the memory is addressed by the CPU during the indirect addressing modes the data on the A bus is not ordinarily used. However, with the circuitry shown within the dotted line **146**, this otherwise "unused" data is put to use to provide the equivalent of the bank switching signals through multiplexer **130**.

Whenever the CPU selects a predetermined range of addresses, the multiplexer **130** selects the equivalent of the bank switching signals from the A bus provided DA7 is high. (This occurs when addressing as zero page the address space -**1800** through **1**FFF.) Once the signal on line **153** is high it is latched through gates **150**, **151** and **152** causing the multiplexer **130** to select the four bits from the A bus (assuming the timing signals are high). Even if the next reference from the CPU is not to this special address range, the multiplexer **130** nonetheless remains latched with the four bits from the data bus. Once the SYN pulse drops, however, which is an indication that an OP code is being fetched, the signal on line **154** rises in potential, causing the multiplexer to switch back to the bank switching signals.

Effectively, what occurs is that when the CPU selects this special address range, (and provided DA7 is high) the bits DA0 through DA3 which are stored in memory, cause a remapping, that is, the address from the CPU accesses a different part of the memory. With the fetching of each OP code, the mapping automatically returns to the bank switching signals. Importantly, the remapping, which occurs is controlled by the bits stored in the RAM (DAϕ through DA3). Thus, with the remapping information stored in RAM, toggling can occur between different portions of the memory without requiring bank switching signals, or the like from the CPU. This enhances the CPU's performance since CPU time is not used for remapping. Additionally, it provides an easy tool for programming.

For some program languages it is desirable to separate data and the program into separate portions of the memory. For example, the 128K memory can be divided into two 64K memories, one for program and one for data. Switching can occur between these memory portions without the generation of bank switching signals by the CPU with the above described circuit. This arrangement is particularly useful when using the Pascal program language.

DISPLAY SUBSYSTEM

The display subsystem **48** of FIG. **1** receives data from the A bus and B bus and converts the data into video signals which may be used for displaying alphanumeric characters or other images on a standard raster scanned cathode ray tube display. The display subsystem **48** specifically generates on line **197**, a standard NTSC color video signal and a video black and white video signal on line **198** (FIG. **8**). This display subsystem, in addition to other inputs, receives a synchronization signal, and several clocking signals. For sake of simplicity, the standard color reference signal of 3.579545 MHz is shown as **C3.5M**. Twice this frequency and four times this frequency are shown as **C7M** and **C14M**, respectively.

Before describing the details of the display subsystem **48**, a discussion of a prior art display system will be helpful in understanding the present display subsystem. In U.S. Pat. No. 4,136,359, a video display system is described which is implemented in a commercially available computer, Apple-II, sold by Apple Computer, Inc., of Cupertino, Calif. In this system, 4-bit digital words are shifted in parallel into a shift register. These words are then circulated in the shift register at 14 MHz to define a waveform having components at 3.5 MHz. Referring to FIG. **9**, line **206**, assume that the digital word 0001 is placed in the shift register and circulated at a rate of 14 MHz. The resultant signal which has a component of 3.5 MHz is shown on line **206**. The phase relationship of this component to the 3.5 MHz reference signal determines the color of the resultant video signal. This relationship is changed by changing the 4-bit word placed in the shift register. As explained in the above-referenced patent, if the signal 1000 is placed in the register and circulated, the resultant phase relationship of the 3.5 MHz component results in the color brown, this signal is shown on line **208**. With this prior art technique, the luminance was determined by the DC component of the signals such as shown on lines **206** and **208**.

The display subsystem **48** of FIG. **1** also uses 4-bit words to generate the various color signals in a manner somewhat similar to the above-described system. Referring to FIG. **8**, 4-bit words representative of colors (16 possible colors) are coupled to the bus **180**. (The generation of these words shall be described in detail in conjunction with FIG. 7.) Instead of using a shift register which circulates the 4-bit work, the same result is achieved by using a multiplexer **205** which sequentially selects each of the lines of the bus **180**. The signals on bus **180** also provide a luminance signal and a black and white video signal with a gray scale.

The 4 lines of the bus **180** are coupled to multiplexer **205**; this multiplexer also receives the C7M and the $\overline{\text{C3.5M}}$ timing signals. These two timing signals cause each of the four lines to be sequentially selected and coupled to line **191**. (Note that the order in which each of the lines of the bus **180** is selected does not change.)

In effect, the multiplexer operates to serialize the parallel signal from bus **180**. Assume for sake of explanation that the digital signals on bus **180** are **1000** as indicated in FIG. **8**. The signal on line **191** will then be 10001000 . . . . The output of the multiplexer **205** coupled to the input of the inverter **204** also receives in a sequential order, the signals from bus **180**, however, in a different order. For the example shown, the input to inverter **204** is 00100010 . . . . After inversion, this results in the signal 11011101 . . . on line **192**. Effectively, the signals on lines **191** and **192** are added by resistors **199** and **200**. The resultant waveform is an AC signal (no DC component) shown in FIG. **9** on line **209**. Thus, with the described circuit, a chroma signal is generated, having a predetermined phase relationship to the 3.5 MHz color reference signal. This phase relationship which is varied by changing the signals on bus **180** determines the color of the video signal on line **197**.

In the prior art display discussed above, the DC component of the color signal determines the luminance. In the present invention, the signals on bus **180** are coupled to the base of transistor **195**, consists of an AC signal from resistors **199** and **200**, and the luminance level also determined by the signals on bus **180**. These inputs to transistor **195**, along with the $\overline{\text{C3.5M}}$ signal, generate a NTSC color signal on line **197** of improved quality when compared to the discussed prior art system.

In some cases, the signals on bus **180** are all binary ones or all binary zeros. When this occurs, there is no AC component from resistors **199** and **200** (no color signal) and the resultant signal on line **197** is either "black" or "white."

The lines of bus **180** are also coupled through resistors to the base of a transistor **196**. Each of these resistors have a different value to provide a "weighting" to the binary signal.

This weighting is used for non-color displays to provide "gray" shades as opposed to having a display with only black and white. The binary signals on bus **180** drive the transistor **196** to provide a video signal on line **198**. RGB is generated with weighted sums of these same five signals.

Referring now to FIG. **7**, data from memory is coupled from the A bus and B bus to registers **159** and **158**, respectively. These registers are clocked by the 1 MHz clocking signal and its complement, thus permitting the sequential transfer of 8-bit words every 0.5 msec. As will be described, in some display modes the data is transferred at the 2 MHz rate, and in other display modes, at a 1 MHz rate.

The registers **158** and **159** are coupled to an 8 line display bus **160**. This display bus transfers data to registers **164** and **173**, and also addresses to a memory **162**. The registers **164** and **173** and memory **162** are enabled during specific display modes as will be apparent.

The character memory **162**, in the presently preferred embodiment, is a random-access memory which stores patterns representative of alpha-numeric characters. Each time the computer is powered up, the character information is transferred from the ROM **50** into the character memory **162** during an initialization period. During character display modes, the signals from the display bus **160** are addresses, identifying particular alpha-numeric characters stored within the character memory **160**. The vertical counter signals V_A, V_B, and V_C(previously discussed in conjunction with adder **121** of FIG. **4**) identify the particular line in each character which is to be displayed. Thus, the generation of the digital signals representative of each of the characters occurs in an ordinary manner. The 7-bit signal representative of each line of each character (memory output) is coupled to the shift register **167**. Through timing signals not shown, either the register **164** or the character memory **162** is selected to allow the shift register **167** to receive either data directly from the A bus or B bus, or alpha-numeric character information from the memory **162**.

The 7-bits of information from either memory **162** or register **164** are serialized by the shift register **167** either at a 7 MHz rate or 14 MHz rate, depending upon the display mode. The serialized data is coupled by line **185** to the multiplexer **169**, pins **1** and **4**. The inverse of this data is also coupled to multiplexer **169**, pin **3**. Line **185** is also coupled as one input to the multiplexer **166** and to the register **170** (input **1**).

The output **1** of register **170** (line **186**) is coupled to the multiplexer **169**, pin **1**; to register **170** (input **2**); and to multiplexer **166**. Output **2** of register **170** (line **187**) is coupled to input **3** of register **170** and also to multiplexer **166**. Output **3** of register **170** (line **187**) provides a third input to the multiplexer **166**. Input **4** of the register **170** receives the output of the multiplexer **169** (line **189**). Output **4** of register **120** (line **190**) provides one control signal for the multiplexer **171**.

The multiplexer **171** selects either the four lines of bus **183** or the four lines of bus **184**. The output of multiplexer **171**, bus **180**, provides the 4-bit signal discussed in conjunction with FIG. **8**. During one of the high resolution display modes (AHIRES), the multiplexer **171** is controlled by a timing signal from the output of the gate **178**.

The multiplexer **166** selects either the lines of bus **181** or bus **182**. The output of this multiplexer provides the signals for the bus **184**. In all but the AHIRES display

mode, multiplexer **166** selects bus **181**. Thus, typically, the multiplexer **171** receives the signals from bus **174**.

For purposes of description above, and also for purposes of explaining for some of the display modes below a simplifying assumption has been made. The signals coupled to the bus **180** by multiplexer **171**, for most modes, are controlled by the serialized signal on line **190**. This serialized signal is in sychronization with the C7M or C14M clocking signals. The multiplexer **205** of FIG. **8**, which as described above, does the "spinning" for the parallel digital signal on bus **180**, operates in sychronization with the multiplexer **171**. In the description above, and except when otherwise noted below, it is assumed that, by way of example, if the multiplexer **171** is coupling all binary ones and zeros onto bus **180**, the signal on line **191** will be either ones or zeros. Also for this condition the signal on line **192** will be all binary zeros or ones, and thus, no AC signal is generated at the base of transistor **195**. However, as actually implemented, there is a "phase" difference between the clocking of the multiplexer **171** when compared to the sampling of the signals from bus **180** by the multiplexer **205**. This results in a first constant AC signal on the gate of transistor **195** even when it appears that all binary ones are on bus **180**, and a second constant AC signal when all binary zeros are on the bus **180**. Thus, in this specification, when it states that "black" or "white" signals are being generated, instead, as currently implemented, two constant colors are generated on a color display. Where a true black and white is desired, color suppression is introduced such as through the color burst signal.

The circuit of FIG. **7**, along with the circuit of FIG. **8**, provides the capability for several distinct display modes. The first of these modes provides a display consisting of **40** characters (or spaces) per horizontal line. This requires a data rate of 8-bits/MHz or half the data rate the memory is capable of delivering. In this mode, data is loaded from the A bus during every other 0.5 μsec period. (B bus is not used during this mode.) This data addresses the character memory **162**, and along with the signals V_A, V_B and V_C, provides the appropriate character line (7-bits) to the shift register **167**. During this mode, registers **164** and **173** are disabled. The shift register **167** for this mode shifts the data at a data rate of 7 MHz (note $\overline{\text{CH80}}$ is high, allowing the 7 MHz signal from gate **175** to control the shift register **167**). Each 7-bit signal is shifted serially onto line **185** and then to line **189** since multiplexer **169** selects pin **4**. The data is shifted through the register **170** onto line **190**. The serial binary signal on line **190** causes the selection of buses **183** or **184**.

The four lines of bus **183** during this mode are coupled to +V (register **173** is disabled); therefore the selection of bus **184** provides four binary ones. The selection of bus **184** provides four binary zeros through bus **181**. Thus, the serial binary signal on line **190** provides either all binary ones or all binary zeros to bus **180**. As discussed, the circuit of FIG. **8** will provide a black and white display with 40 characters per line.

If the inverse and flashing timing means **172** is selected, each time the shift register **167** is loaded, multiplexer **169** shifts between pins **3** and **4**. This causes the characters to change from white characters on a black background to black characters on a white background, and so on.

During the 80 character per line display mode, the registers **158** and **159** are each loaded during sequential

0.5 μsec periods (this utilizes the 2 MHz cycle rate previously discussed). The shift register **167** shifts the character data from memory **162** at a 14 MHz rate. The serialized data at the 14 MHz rate is shifted through the register **170** and again controls the multiplexer **171** as previously described. (Note that register **170** is always clocked at the 14 MHz rate.) Flashing again can be obtained as previously discussed.

In another alpha-numeric character display mode, the background of each character may be in one color and the character itself (foreground) in another color. This mode provides 40 characters per line. The character identification (address for RAM **162**), is furnished on the A bus to register **159** at a frequency of 1 MHz. The color information (background color and foreground color) is furnished on the B bus as two 4-bit words to register **158**. In the manner previously described, the address from register **159** selects the appropriate character from memory **162** and provides this information to shift register **167**. The color information from the B bus is transferred to register **173**. For purposes of explanation, assume that the 4-bits identifying the color red for the background are on bus **184** (from register **173** and multiplexer **166**) and that 4-bits representing the color blue for the foreground are on bus **183**. (Note that when register **173** is enabled, the signals from the register override the binary ones and zeros which otherwise appear on the lines of bus **174**.) The serial binary signal representative of the character itself on line **190**, selects either the color blue from bus **183** for the character itself or the color red from bus **184** for the background. The digital signals representative of these colors are transferred to bus **180** and provide the color data to the circuit of FIG. **8**. For black and white displays, a "gray" scale is provided through the weighting circuit associated with transistor **196** of FIG. **8**. Again, the multiplexer **169** may, through the timing means **172**, alternate between the signal of line **185** and its inverse, which will have the effect of interchanging the foreground and background colors.

During the high resolution graphics modes, the character memory **162** is not used, but rather, data from the memory directly provides pattern information for display. This requires more mapping of data from within the main memory since new data is required for each line of the display. (Note that when characters are displayed, the character memory **162** provides the different signals required for the 8 lines of each character row.) During these high resolution modes, the register **164** is enabled and the character memory **162** is disabled. Thus, the data from the A bus and B bus is shifted into the shift register **167**. In these modes, the "HRES" signal to multiplexer **169** causes this multiplexer to select between pins **1** and **2**. Pin **2** provides the signal directly from the shift register **167** while the signal on pin **1** is effectively the signal on line **185** delayed by one period of the C14M signal. This delay occurs through the register **170** from input **2** to output **2** since register **170** is clocked at C14M.

During a first graphics mode, data from the display bus **160** is loaded into shift register **167** at the rate of 7-bits/MHz. The data is serialized on line **185** and in the manner previously described for displaying characters, controls the selection of all binary ones and all binary zeros through the multiplexer **171**. Note, as mentioned before, in the presently preferred embodiment, unless color suppression is used, this will not result in a black and white display, but rather a two-color display. If a high bit is present on line **140** of the display bus, the inverse and flashing timing means **172** causes the multiplexer **169** to alternate between pins **1** and **2**. This switching occurs at a 1 MHz rate and provides a phase shift for every other 7-bits of data coupled to the multiplexer **171** on line **190**. This results in an additional color being generated on the display for every other 7-bits of data.

For the above-described graphics modes when shift register **161** is shifting at a 7 MHz rate, 8-bits may be coupled to the bus **160** during each period. Specifically, as in the case of the differing background and foreground colors for the 40 character per line display mode, two 4-bit color words are shifted into register **173** at a rate of 1 MHz. Then, the multiplexer **171** selects between two predetermined colors on buses **183** and **184**. Note these colors can be changed at a 1 MHz rate.

In an additional color mode identified as "AHIRES," multiplexer **171** operates under the control of gates **176**, **177** and **178**. In effect, multiplexer **171** selects bus **184** and latches the signals on this bus every four cycles of the C14M clock. Data is shifted into the shift register **167** from the A bus and B bus every 0.5 μ sec the register **167** operates under the control of the C14M signal. Each data bit on line **185** is shifted first to line **186**, then to line **187** and finally to line **188**. These lines are coupled to the multiplexer **171** through multiplexer **166** which selects bus **182** since AHIRES is high. In effect, what occurs is that 4-bit color words are serialized onto line **185** and then brought back into parallel on bus **182**. Since multiplexer **171** latches the signals on bus **184** every four cycles of the C14M signal, a new color word is generated at a 3.5 MHz rate on the bus **180**. The resultant display is **140** by **192** colored blocks wherein each block can be any one of 16 colors.

In the last display mode, typically used with color suppression, data is shifted into the shift register **167** from the display bus at the rate of 14-bits/MHz. The data is serialized onto line **185** and controls the selection of either all binary ones or all zeros through multiplexer **171**. This provides the highest resolution graphics display for the system.

Thus, a microcomputer with video display capability has been described. The computer is fabricated from commercially available parts and provides high utilization of these parts. Numerous existing programs including many of those which operate on the Apple-II computer, may be employed in the above-described computer.

TABLE I

```
F000:   ****************************
F000:   *     CRITICAL TIMING        *
F000:   *   REQUIRES PAGE BOUND      *
F000:   *   CONSIDERATIONS FOR       *
F000:   *      CODE AND DATA         *
F000:   *      -----CODE-----        *
F000:   *   VIRTUALLY THE ENTIRE     *
F000:   *     'WRITE' ROUTINE        *
F000:   *      MUST NOT CROSS        *
F000:   *     PAGE BOUNDARIES        *
F000:   *  CRITICAL BRANCHES IN      *
F000:   *  THE 'WRITE', 'READ',      *
F000:   *  AND 'READ ADR' SUBRS      *
F000:   *  WHICH MUST NOT CROSS      *
F000:   *  PAGE BOUNDARIES ARE       *
F000:   *  NOTED IN COMMENTS         *
F000:   *                            *
F000:   ****************************
F000:   *                            *
F000:   *         EQUATES            *
F000:   *                            *
0200:   NBUF1    EQU   $200
0302:   NBUF2    EQU   $302          ;(ZERO PAGE AT $300)
F000:   *
0080:   HRDERRS  EQU   $80
00E0:   DVMOT    EQU   $E0
F000:   *
0081:   IBSLOT   EQU   $81
0082:   IBDRVN   EQU   IBSLOT+1
0083:   IBTRK    EQU   IBSLOT+2
0084:   IBSECT   EQU   IBSLOT+3
0085:   IBBUFP   EQU   IBSLOT+4    ;&5
0087:   IBCMD    EQU   IBSLOT+6
0088:   IBSTAT   EQU   IBSLOT+7
0089:   IBSMOD   EQU   IBSLOT+8
0089:   CSUM     EQU   IBSMOD      ;USED ALSO FOR ADDRESS HEADER CKSUM
008A:   IOBPDN   EQU   IBSLOT+9
008B:   IMASK    EQU   IBSLOT+$A
008C:   CURTRK   EQU   IBSLOT+$B
0085:   DRVOTRK  EQU   CURTRK-7
F000:   ;SLOT 4, DRIVE 1
F000:   ;SLOT 4, DRIVE 2
F000:   ;SLOT 5, DRIVE 1
F000:   ;SLOT 5, DRIVE 2
F000:   ;SLOT 6, DRIVE 1
F000:   ;SLOT 6, DRIVE 2
0093:   RETRYCNT EQU   IBSLOT+$12
0094:   SEEKCNT  EQU   IBSLOT+$13
009B:   BUF      EQU   IBSLOT+$1A
009F:   ENVTEMP  EQU   IBSLOT+$1E
F000:   *IBSLOT+$1F NOT USED
F000:   *
F000:   ************************
F000:   *                      *
F000:   *     ----READADR----  *
F000:   *                      *
F000:   ************************
0095:   COUNT    EQU   IBSLOT+$14 ;'MUST FIND' COUNT.
0095:   LAST     EQU   IBSLOT+$14 ;'ODD BIT' NIBLS.
0096:   CKSUM    EQU   IBSLOT+$15 ;CHECKSUM BYTE.
0097:   CSSTV    EQU   IBSLOT+$16 ;FOUR BYTES,
F000:   *        CHECKSUM, SECTOR, TRACK, AND VOLUME.
F000:   *
F000:   ************************
F000:   *                      *
```

APPLE /// BOOT ROM LISTING
REVISION Ø ROM (See addr F189)

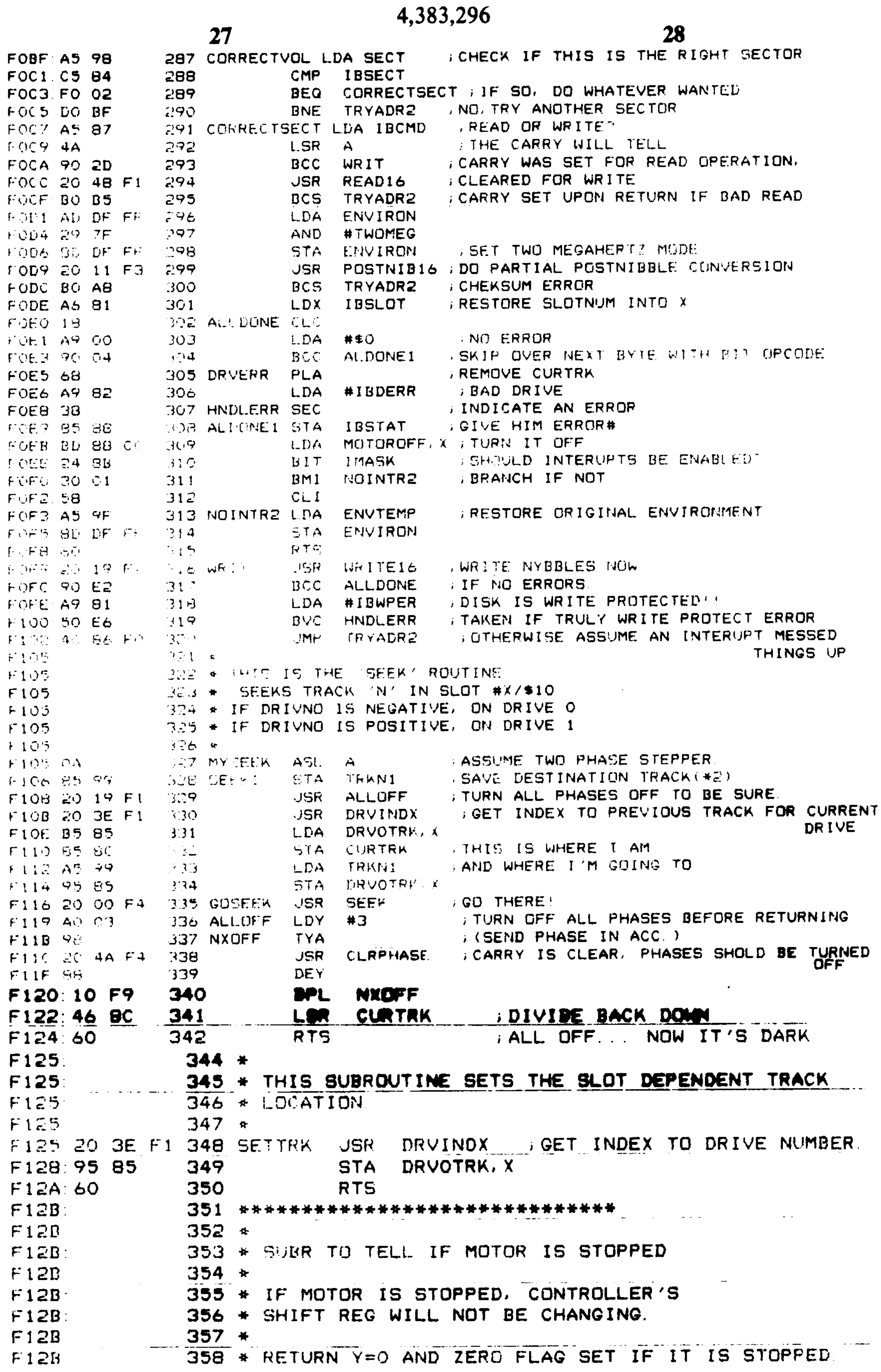
```
F000:      *     ----WRITE----      *
F000:      *                        *
F000:      *     USES ALL NBUFS     *
F000:      *      AND 32-BYTE       *
F000:      *   DATA TABLE 'NIBL'    *
F000:      *                        *
F000:      **************************
F000:      *
F000:      **************************
F000:      *                        *
F000:      *      -----READ----     *
F000:      *                        *
F000:      *     USES ALL NBUFS     *
F000:      *  USES LAST 54 BYTES    *
F000:      *  OF A CODE PAGE FOR    *
F000:      *  SIGNIFICANT BYTES     *
F000:      *  OF DNIBL TABLE.       *
F000:      *                        *
F000:      **************************
F000:      *
F000:      **************************
F000:      *                        *
F000:      *     ---- SEEK ----     *
F000:      *                        *
F000:      **************************
0095       TRKCNT   EQU  COUNT      ;HALFTRKS MOVED COUNT
009D.      PRIOR    EQU  IBSLOT+$1C
009E       TRKN     EQU  IBSLOT+$1D
F000:      *
F000:      **************************
F000:      *                        *
F000       *    ---- MSWAIT ----    *
F000       *                        *
F000       **************************
0099       MONTIMEL EQU CSSTV+2     ;MOTOR-ON TIME
009A:      MONTIMEH EQU MONTIMEL+1  ;COUNTERS.
F000.      *
F000:      **************************
F000:      *                        *
F000:      *    DEVICE ADDRESS      *
F000:      *     ASSIGNMENTS        *
F000:      *                        *
F000:      **************************
C080:      PHASEOFF EQU  $C080      ;STEPPER PHASE OFF.
C081:      PHASEON  EQU  $C081      ;STEPPER PHASE ON.
C08C:      Q6L      EQU  $C08C      ;Q7L,Q6L=READ
C08D:      Q6H      EQU  $C08D      ;Q7L,Q6H=SENSE WPROT
C08E:      Q7L      EQU  $C08E      ;Q7H,Q6L=WRITE
C08F:      Q7H      EQU  $C08F      ;Q7H,Q6H=WRITE STORE
FFEF:      INTERUPT EQU  $FFEF
FFDF:      ENVIRON  EQU  $FFDF
0080:      ONEMEG   EQU  $80
007F:      TWOMEG   EQU  $7F
F000:      **********************************
F000:      *
F000:      * EQUATES FOR RWTS AND BLOCK
F000:      *
F000:      **********************************
C088:      MOTOROFF EQU $C088
```

```
C089:     MOTORON EQU   $C089
C08A:     DRVOEN  EQU   $C08A
C08B:     DRV1EN  EQU   $C08B
C081:     PHASON  EQU   $C081
C080:     PHSOFF  EQU   $C080
0097      TEMP    EQU   CSSTV      ;PUT ADDRESS INFO HERE
0097      CSUM1   EQU   TEMP
0098      SECT    EQU   CSUM1+1
0099      TRACK   EQU   SECT+1
0099:     TRKN1   EQU   TRACK
009A:     VOLUME  EQU   TRACK+1
0083      IBRERR  EQU   HRDERRS+3
0082      IBDERR  EQU   HRDERRS+2
0081      IBWPER  EQU   HRDERRS+1
0080:     IBNODRV EQU   HRDERRS
F000           ***************************
F000:          *                         *
F000           *      READ WRITE A       *
F000           *    TRACK AND SECTOR     *
F000           *                         *
F000           ***************************
F000           *
F000 A0 01     REGRWTS LDY   #1            ;RETRY COUNT
F002 A6 81             LDX   IBSLOT        ;GET SLOT # FOR THIS OPERATION
F004 84 94             STY   SEEKCNT       ;ONLY ONE RECALIBRATE PER CALL
F006 08                PHP                 ;DETERMINE INTERUPT STATUS
F007 68                PLA
F008 6A                ROR   A
F009 6A                ROR   A             ;GET INTERUPT FLAG INTO BIT 7
F00A 6A                ROR   A
F00B 6A                ROR   A
F00C 85 9B             STA   IMASK
F00E AD DF FF          LDA   ENVIRON       ;PRESERVE ENVIRONMENT
F011 85 9F             STA   ENVTEMP
F013           *
F013           * NOW CHECK IF THE MOTOR IS ON, THEN START IT
F013           *
F013 20 2B F1          JSR   CHKDRV        ;SET ZERO FLAG IF MOTOR STOPPED
F016 08                PHP                 ;SAVE TEST RESULTS
F017 A5 85             LDA   IBBUFP        ;MOVE OUT POINTER TO BUFFER INTO ZPAGE
F019 85 9B             STA   BUF
F01B A5 86             LDA   IBBUFP+1
F01D 85 9C             STA   BUF+1
F01F A9 E0             LDA   #DVMOT
F021 85 9A             STA   MONTIMEH
F023 A5 82             LDA   IBDRVN        ;DETERMINE DRIVE ONE OR TWO
F025 C5 8A             CMP   IOBPDN        ;SAME DRIVE USED BEFORE
F027 85 8A             STA   IOBPDN        ;SAVE IT FOR NEXT TIME
F029 08                PHP                 ;KEEP RESULTS OF COMPARE
F02A 6A                ROR   A             ;GET DRIVE NUMBER INTO CARRY
F02B BD 89 C0          LDA   MOTORON,X     ;TURN ON THE DRIVE
F02E 90 01             BCC   DRIVSEL       ;BRANCH IF DRIVE 1 SELECTED
F030 E8                INX                 ;SELECT DRIVE 2
F031 BD 8A C0  DRIVSEL LDA   DRVOEN,X
F034 20 40 F7          JSR   SET1MEG       ;INSURE ONE MEGAHERTZ OPERATION
F037 28                PLP                 ;WAS IT SAME DRIVE?
F038 F0 0A             BEQ   OK
F03A 28                PLP                 ;MUST INDICATE DRIVE OFF BY SETTING ZERO
F03B A0 07             LDY   #7            ;DELAY 150 MS BEFORE STEPPING      FLAG)
F03D 20 56 F4  DRVWAIT JSR   MSWAIT        ;(ON RETURN A=0)
F040 88                DEY
F041 D0 FA             BNE   DRVWAIT
F043 08                PHP                 ;NOW ZERO FLAG SET
F044 A5 83     OK      LDA   IBTRK         ;GET DESTINATION TRACK
F046 A6 81             LDX   IBSLOT        ;RESTORE PROPER X (SLOT*16)
F048 20 05 F1          JSR   MYSEEK        ;AND GO TO IT
F04B           *NOW AT THE DESIRED TRACK   WAS THE MOTOR
F04B           * ON TO START WITH?
F04B 28                PLP                 ;WAS MOTOR ON?
F04C D0 17             BNE   TRYTRK        ;IF SO, DON'T DELAY, GET IT TODAY!
F04E           *
```

"APPLE_PAT_4_383_296_21" 151 KB 2000-02-27 dpi: 300h x 300v pix: 1836h x 2780v

```
F04E                 211 * MOTOR WAS OFF, WAIT FOR IT TO SPEED UP
F04E                 212 *
F04E: A0 12          213 MOTOF    LDY   #$12         ;WAIT EXACTLY 100 US FOR EACH COUNT
F050: 88             214 CONWAIT  DEY                                      IN MONTIME
F051  D0 FD          215          BNE   CONWAIT
F053  E6 99          216          INC   MONTIMEL     ;COUNT UP TO 0000
F055  D0 F7          217          BNE   MOTOF
F057  E6 9A          218          INC   MONTIMEH
F059  D0 F3          219          BNE   MOTOF
F05B:                221 ********************************
F05B:                222 *
F05B                 223 * MOTOR SHOULD BE UP TO SPEED
F05B                 224 * IF IT STILL LOOKS STOPPED THEN
F05B                 225 * THE DRIVE IS NOT PRESENT
F05B                 226 *
F05B                 227 ********************************
F05B: 20 2B F1       228          JSR   CHKDRV       ;IS DRIVE PRESENT?
F05E: D0 05          229          BNE   TRYTRK       ;YES, CONTINUE
F060  A9 80          230 NODRIVERR LDA #IBNODRV      ;NO, GET TELL EM NO DRIVE!
F062  4C EB F        231          JMP   HNDLERR
F065                 232 *
F065                 233 * NOW CHECK  IF IT IS NOT THE FORMAT DISK COMMAND.
F065                 234 *  LOCATE THE CORRECT SECTOR FOR THIS OPERATION
F065                 235 *
F065  A5 87          236 TRYTRK   LDA   IBCMD        ;GET COMMAND CODE #
F067  F0 77          237          BEQ   ALLDONE      ;IF NULL COMMAND, GO HOME TO BED
F069  C9 03          238          CMP   #3           ;COMMAND IN RANGE?
F06B  B0 73          239          BCS   ALLDONE      ;NO, DO NOTHING!
F06D  6A             240          ROR   A            ;SET CARRY=1 FOR READ, 0 FOR WRITE
F06E  B0 0B          241          BCS   TRYTRK2      ;MUST PRENIBBLIZE FOR WRITE
F070  AD DF FF       242          LDA   ENVIRON
F073  29 7F          243          AND   #TWOMEG      ;SHIFT TO HIGH SPEED!
F075  8D DF FF       244          STA   ENVIRON
F078. 20 C6 F2       245          JSR   PRENIB16
F07B  A0 7F          246 TRYTRK2  LDY   #$7F         ;ONLY 127 RETRIES OF ANY KIND
F07D  84 93          247          STY   RETRYCNT
F07F  A6 B1          248 TRYADR   LDX   IBSLOT       ;GET SLOT NUM INTO X-REG
F081  20 BD F        249          JSR   RDADR16      ;READ NEXT ADDRESS FIELD
F084: 90 21          250          BCC   RDRIGHT      ;IF READ IT RIGHT, HURRAH!
F086  24 BB          251 TRYADR2  BIT   IMASK        ;SHOULD INTERUPTS BE ALLOWED?
F088  30 01          252          BMI   NOINTR1      ;NO. DON'T ALLOW THEM.
F08A  58             253          CLI                ;RE-ENABLED AFTER READ/READADR 'WRIT'
                                                                              FAILURE
F08B  C6 93          254 NOINTR1  DEC   RETRYCNT     ;ANOTHER MISTAEK!!
F08D  10 F0          255          BPL   TRYADR       ; WELL, LET IT GO THIS TIME.
F08F  A5 8C          256          LDA   CURTRK
F091  48             257          PHA                ;SAVE TRACK WE REALLY WANT
F092  C6 94          258          DEC   SEEKCNT      ;ONLY RECALIBRATE ONCE!
F094  D0 4F          259          BNE   DRVERR       ;TRIED TO RECALIBRATE A SECOND TIME, ERROR!
F096  A9 60          260          LDA   #$60         ;RECALIBRATE ALL OVER AGAIN!
F098  20 25 F1       261          JSR   SETTRK       ;PRETEND TO BE ON TRACK 80
F09B  A9 00          262          LDA   #$00
F09D: 20 05 F1       263          JSR   MYSEEK       ;MOVE TO TRACK 00
F0A0  68             264 GOCAL1   PLA
F0A1  20 05 F1       265 GOCAL    JSR   MYSEEK       ;GO TO CORRECT TRACK THIS TIME!
F0A4  4C 7B F        266          JMP   TRYTRK2      ;LOOP BACK, TRY AGAIN ON THIS TRACK
F0A7                 267
F0A7.                268 * HAVE NOW READ AN ADDRESS FIELD CORRECTLY.
F0A7                 269 * MAKE SURE THIS IS THE TRACK, SECTOR, AND VOLUME DESIRED.
F0A7  A4 99          270 RDRIGHT  LDY   TRACK        ;ON THE RIGHT TRACK?
F0A9  C4 8C          271          CPY   CURTRK
F0AB  F0 0E          272          BEQ   RTTRK        ;IF SO, GOOD
F0AD                 273 * RECALIBRATING FROM THIS TRACK
F0AD: A5 8C          274          LDA   CURTRK       ;PRESERVE DESTINATION TRACK
F0AF. 48             275          PHA
F0B0: 98             276          TYA
F0B1  20 25 F1       277          JSR   SETTRK
F0B4. 68             278          PLA
F0B5: 20 05 F1       279          JSR   MYSEEK
F0B8: 4C 86 F0       280          JMP   TRYADR2      ;GO AHEAD AND RECALIBRATE
F0BB:                282 *
F0BB:                283 * DRIVE IS ON RIGHT TRACK, CHECK VOLUME MISMATCH
F0BB                 284 *
F0BB  A5 9A          285 RTTRK    LDA   VOLUME       ;GET ACTUAL VOLUME HERE
F0BD: 85 89          286          STA   IBSMOD       ;TELL OPSYS WHAT VOLUME WAS THERE
```

“APPLE_PAT_4_383_296_22” 165 KB 2000-02-27 dpi: 300h x 300v pix: 1837h x 2792v



```
F0BF A5 98     CORRECTVOL LDA SECT      ;CHECK IF THIS IS THE RIGHT SECTOR
F0C1 C5 84                CMP  IBSECT
F0C3 F0 02                BEQ  CORRECTSECT ;IF SO, DO WHATEVER WANTED
F0C5 D0 BF                BNE  TRYADR2   ;NO, TRY ANOTHER SECTOR
F0C7 A5 87     CORRECTSECT LDA IBCMD     ;READ OR WRITE?
F0C9 4A                   LSR  A         ;THE CARRY WILL TELL
F0CA 90 2D                BCC  WRIT      ;CARRY WAS SET FOR READ OPERATION,
F0CC 20 48 F1             JSR  READ16    ;CLEARED FOR WRITE
F0CF B0 B5                BCS  TRYADR2   ;CARRY SET UPON RETURN IF BAD READ
F0D1 AD DF FF             LDA  ENVIRON
F0D4 29 7F                AND  #TWOMEG
F0D6 8D DF FF             STA  ENVIRON   ;SET TWO MEGAHERTZ MODE
F0D9 20 11 F3             JSR  POSTNIB16 ;DO PARTIAL POSTNIBBLE CONVERSION
F0DC B0 A8                BCS  TRYADR2   ;CHEKSUM ERROR
F0DE A6 81                LDX  IBSLOT    ;RESTORE SLOTNUM INTO X
F0E0 18        ALLDONE    CLC
F0E1 A9 00                LDA  #$0       ;NO ERROR
F0E3 90 04                BCC  ALDONE1   ;SKIP OVER NEXT BYTE WITH BIT OPCODE
F0E5 68        DRVERR     PLA            ;REMOVE CURTRK
F0E6 A9 82                LDA  #IBDERR   ;BAD DRIVE
F0E8 38        HNDLERR    SEC            ;INDICATE AN ERROR
F0E9 85 88     ALDONE1    STA  IBSTAT    ;GIVE HIM ERROR#
F0EB BD 88 C0             LDA  MOTOROFF,X ;TURN IT OFF
F0EE 24 8B                BIT  IMASK     ;SHOULD INTERUPTS BE ENABLED?
F0F0 30 01                BMI  NOINTR2   ;BRANCH IF NOT
F0F2 58                   CLI
F0F3 A5 9F     NOINTR2    LDA  ENVTEMP   ;RESTORE ORIGINAL ENVIRONMENT
F0F5 8D DF FF             STA  ENVIRON
F0F8 60                   RTS
F0F9 20 19 F   WRIT       JSR  WRITE16   ;WRITE NYBBLES NOW
F0FC 90 E2                BCC  ALLDONE   ;IF NO ERRORS.
F0FE A9 81                LDA  #IBWPER   ;DISK IS WRITE PROTECTED!!
F100 50 E6                BVC  HNDLERR   ;TAKEN IF TRULY WRITE PROTECT ERROR
F102 4C 86 F0             JMP  TRYADR2   ;OTHERWISE ASSUME AN INTERUPT MESSED
                                                                    THINGS UP
F105                *
F105                * THIS IS THE 'SEEK' ROUTINE
F105                *  SEEKS TRACK 'N' IN SLOT #X/$10
F105                * IF DRIVNO IS NEGATIVE, ON DRIVE 0
F105                * IF DRIVNO IS POSITIVE, ON DRIVE 1
F105                *
F105 0A        MYSEEK     ASL  A         ;ASSUME TWO PHASE STEPPER
F106 85 99     SEEK1      STA  TRKN1     ;SAVE DESTINATION TRACK(*2)
F108 20 19 F1             JSR  ALLOFF    ;TURN ALL PHASES OFF TO BE SURE.
F10B 20 3E F1             JSR  DRVINDX   ;GET INDEX TO PREVIOUS TRACK FOR CURRENT
                                                                     DRIVE
F10E B5 85                LDA  DRVOTRK,X
F110 85 8C                STA  CURTRK    ;THIS IS WHERE I AM
F112 A5 99                LDA  TRKN1     ;AND WHERE I'M GOING TO
F114 95 85                STA  DRVOTRK,X
F116 20 00 F4  GOSEEK     JSR  SEEK      ;GO THERE!
F119 A0 03     ALLOFF     LDY  #3        ;TURN OFF ALL PHASES BEFORE RETURNING
F11B 98        NXOFF      TYA            ;(SEND PHASE IN ACC.)
F11C 20 4A F4             JSR  CLRPHASE  ;CARRY IS CLEAR, PHASES SHOLD BE TURNED
                                                                       OFF
F11F 88                   DEY
F120: 10 F9               BPL  NXOFF
F122: 46 8C               LSR  CURTRK         ;DIVIDE BACK DOWN
F124: 60                  RTS                 ;ALL OFF... NOW IT'S DARK
F125:                 *
F125:                 * THIS SUBROUTINE SETS THE SLOT DEPENDENT TRACK
F125:                 * LOCATION
F125                  *
F125 20 3E F1   SETTRK      JSR  DRVINDX   ;GET INDEX TO DRIVE NUMBER.
F128: 95 85                 STA  DRVOTRK,X
F12A: 60                    RTS
F12B:                 ******************************
F12B                  *
F12B:                 * SUBR TO TELL IF MOTOR IS STOPPED
F12B                  *
F12B:                 * IF MOTOR IS STOPPED, CONTROLLER'S
F12B:                 * SHIFT REG WILL NOT BE CHANGING.
F12B                  *
F12B                  * RETURN Y=0 AND ZERO FLAG SET IF IT IS STOPPED.
```

"APPLE_PAT_4_383_296_23" 175 KB 2000-02-27 dpi: 300h x 300v pix: 1824h x 2767v

```
F12B              359 *
F12B:             360 ******************************
F12B: A0 00       361 CHKDRV  LDY  #0         ;INIT LOOP COUNTER
F12D: BD 8C C0    362 CHKDRV1 LDA  Q6L,X      ;READ THE SHIFT REG
F130: 20 3D F1    363         JSR  CKDRTS     ;DELAY
F133  48          364         PHA
F134  68          365         PLA             ;MORE DELAY
F135  DD 8C C0    366         CMP  Q6L,X      ;HAS SHIFT REG CHANGED?
F138  D0 03       367         BNE  CKDRTS     ;YES, MOTOR IS MOVING
F13A  88          368         DEY             ;NO,DEC RETRY COUNTER
F13B  D0 F0       369         BNE  CHKDRV1    ;AND TRY 256 TIMES
F13D  60          370 CKDRTS  RTS             ;THEN RETURN
F13E              371 *
F13E  48          372 DRVINDX PHA              ;PRESERVE ACC
F13F  8A          373         TXA              ;GET SLOT(*$10)/8
F140  4A          374         LSR  A
F141  4A          375         LSR  A
F142  4A          376         LSR  A
F143  05 82       377         ORA  IBDRVN      ;FOR DRIVE 0 OR 1
F145  AA          378         TAX              ;INTO X FOR INDEX TO TABLE
F146  68          379         PLA              ;RESTORE ACC
F147  60          380         RTS
F148              381 ******************************
F148              382 *
F148              383 * NOTE: FORMATTING ROUTINES
F148              384 *       NOT INCLUDED FOR SOS
F148              385 *
F148              386 ******************************
F148:             388 **************************
F148:             389 *                        *
F148:             390 *     READ SUBROUTINE    *
F148:             391 *   (16-SECTOR FORMAT)   *
F148:             392 *                        *
F148:             393 **************************
F148:             394 *                        *
F148:             395 *   READS ENCODED BYTES  *
F148:             396 *  INTO NBUF1 AND NBUF2  *
F148:             397 *                        *
F148:             398 *  FIRST READS NBUF2     *
F148:             399 *          HIGH TO LOW,  *
F148:             400 *  THEN READS NBUF1      *
F148:             401 *          LOW TO HIGH.  *
F148:             402 *                        *
F148:             403 *   ---- ON ENTRY ----   *
F148:             404 *                        *
F148:             405 *  X-REG: SLOTNUM        *
F148:             406 *         TIMES $10.     *
F148:             407 *                        *
F148:             408 *  READ MODE (Q6L, Q7L)  *
F148:             409 *                        *
F148:             410 *   ---- ON EXIT -----   *
F148:             411 *                        *
F148:             412 *  CARRY SET IF ERROR.   *
F148:             413 *                        *
F148:             414 *  IF NO ERROR:          *
F148:             415 *     A-REG HOLDS $AA.   *
F148:             416 *     X-REG UNCHANGED.   *
F148:             417 *     Y-REG HOLDS $00.   *
F148:             418 *     CARRY CLEAR.       *
F148:             419 *   ---- CAUTION -----   *
F148:             420 *                        *
```

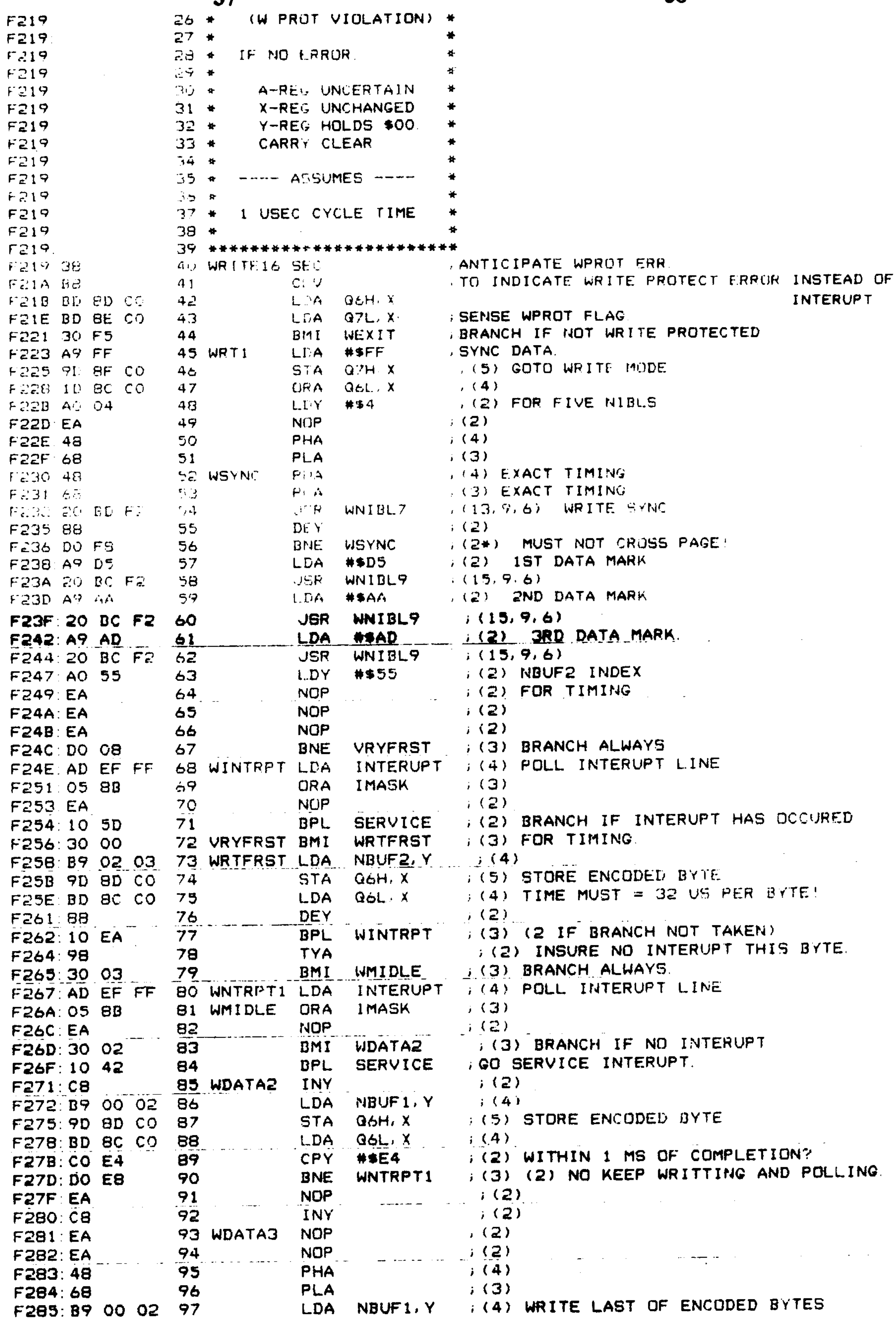
```
F148:          421 *       OBSERVE        *
F148:          422 *   'NO PAGE CROSS'    *
F148:          423 *     WARNINGS ON      *
F148:          424 *   SOME BRANCHES!!    *
F148:          425 *                      *
F148:          426 *  ---- ASSUMES ----   *
F148:          427 *                      *
F148:          428 *   1 USEC CYCLE TIME  *
F148:          429 *                      *
F148:          430 ************************
F148 A0 20     431 READ16  LDY  #$20      ;'MUST FIND' COUNT.
F14A 88        432 RSYNC   DEY            ;IF CAN'T FIND MARKS
F14B F0 6B     433         BEQ  RDERR     ;THEN EXIT WITH CARRY SET
F14D BD 8C C0  434 RD1     LDA  Q6L,X     ;READ NIBL.
F150:10 FB     435         BPL  RD1       ;*** NO PAGE CROSS! ***
F152:49 D5     436 RSYNC1  EOR  #$D5      ;DATA MARK 1?
F154:D0 F4     437         BNE  RSYNC     ;LOOP IF NOT.
F156 EA        438         NOP            ;DELAY BETWEEN NIBLS.
F157 BD 8C C0  439 RD2     LDA  Q6L,X
F15A 10 FB     440         BPL  RD2       ;*** NO PAGE CROSS! ***
F15C:C9 AA     441         CMP  #$AA      ;DATA MARK 2?
F15E:D0 F2     442         BNE  RSYNC1    ;(IF NOT, IS IT DM1?)
F160:A0 55     443         LDY  #$55      ;INIT NBUF2 INDEX.
F162           444 *            (ADDED NIBL DELAY)
F162 BD 8C C0  445 RD3     LDA  Q6L,X
F165 10 FB     446         BPL  RD3       ;*** NO PAGE CROSS! ***
F167 C9 AD     447         CMP  #$AD      ;DATA MARK 3?
F169 D0 E7     448         BNE  RSYNC1    ;(IF NOT, IS IT DM1?)
F16B,          449 *        (CARRY SET IF DM3!)
F16B BD 8C C0  450 RD4     LDA  Q6L,X
F16E 10 FB     451         BPL  RD4       ;*** NO PAGE CROSS! ***
F170 99 02 03  452         STA  NBUF2,Y   ;STORE BYTES DIRECTLY
F173 AD EF FF  453         LDA  INTERUPT  ;POLL INTERUPT LINE
F176 05 8B     454         ORA  IMASK     ;(THIS MAY BE USED TO INVALIDATE POLL)
F178 10 40     455         BPL  GOSERV
F17A 88        456         DEY            ;INDEX TO NEXT
F17B 10 EE     457         BPL  RD4
F17D C8        458 RD5     INY            ;(FIRST TIME Y=0)
F17E BD 8C C0  459 RD5A    LDA  Q6L,X     GET ENCODED BYTES OF NBUF1
F181 10 FB     460         BPL  RD5A
F183 99 00 02  461         STA  NBUF1,Y
F186 AD EF FF  462         LDA  INTERUPT  ;POLL INTERUPT LINE
F189 05 8B     463         ORA  IMASK     ;(THIS MAY BE USED TO INVALIDATE POLL)
F18B 10 2D     464         BPL  GOSERV
F18D C0 E4     465         CPY  #$E4      WITHIN 1 MS OF COMPLETION?
F18F D0 EC     466         BNE  RD5
F191 C8        467         INY
F192 BD 8C C0  468 RD6     LDA  Q6L,X     ;NO POLL FROM NOW ON
F195 10 FB     469         BPL  RD6
F197 99 00 02  470         STA  NBUF1,Y
F19A C8        471         INY            ;FINISH OUT NBUF1 PAGE
F19B D0 F5     472         BNE  RD6
F19D BD 8C C0  473 RDCKSUM LDA  Q6L,X     ;GET CHECKSUM BYTE.
F1A0 10 FB     474         BPL  RDCKSUM
F1A2 85 96     475         STA  CKSUM
F1A4 EA        476         NOP            ;EXTRA DELAY BETWEEN BYTES
F1A5 BD 8C C0  477 RD7     LDA  Q6L,X
F1A8 10 FB     478         BPL  RD7       ;*** NO PAGE CROSS! ***
F1AA C9 DE     479         CMP  #$DE      ;FIRST BIT SLIP MARK?
F1AC D0 0A     480         BNE  RDERR     ;(ERR IF NOT)
F1AE EA        481         NOP            ;DELAY BETWEEN NIBLS.
F1AF BD 8C C0  482 RD8     LDA  Q6L,X
F1B2 10 FB     483         BPL  RD8       ;*** NO PAGE CROSS! ***
F1B4 C9 AA     484         CMP  #$AA      ;SECOND BIT SLIP MARK?
F1B6 F0 5F     485         BEQ  RDEXIT    ;(DONE IF IT IS)
F1B8 38        486 RDERR   SEC            ;INDICATE 'ERROR EXIT'
F1B9 60        487         RTS            ;RETURN FROM READ16 OR RDADR16.
F1BA           488 *
F1BA 4C B3 F2  489 GOSERV  JMP  SERVICE   ;GO SERVICE INTERUPT
```

ADDRESS F1B9 SPECIFIES
ROM REVISION:
REV Ø — 6Ø
REV I — AØ

“APPLE_PAT_4_383_296_25” 174 KB 2000-02-27 dpi: 300h x 300v pix: 1984h x 2860v

```
F1BD:          491 ****************************
F1BD:          492 *                          *
F1BD:          493 *   READ ADDRESS FIELD     *
F1BD:          494 *       SUBROUTINE         *
F1BD:          495 *   (16-SECTOR FORMAT)     *
F1BD:          496 *                          *
F1BD:          497 ****************************
F1BD:          498 *                          *
F1BD:          499 *   READS VOLUME, TRACK    *
F1BD:          500 *       AND SECTOR         *
F1BD:          501 *                          *
F1BD:          502 *  ---- ON ENTRY ----      *
F1BD:          503 *                          *
F1BD:          504 * XREG: SLOTNUM TIMES $10  *
F1BD:          505 *                          *
F1BD:          506 * READ MODE (Q6L, Q7L)     *
F1BD:          507 *                          *
F1BD:          508 *  ---- ON EXIT -----      *
F1BD:          509 *                          *
F1BD:          510 * CARRY SET IF ERROR       *
F1BD:          511 *                          *
F1BD:          512 * IF NO ERROR              *
F1BD:          513 *   A-REG HOLDS $AA        *
F1BD:          514 *   Y-REG HOLDS $00.       *
F1BD:          515 *   X-REG UNCHANGED.       *
F1BD:          516 *   CARRY CLEAR.           *
F1BD:          517 *                          *
F1BD:          518 *   CSSTV HOLDS CHKSUM,    *
F1BD:          519 *     SECTOR, TRACK, AND   *
F1BD:          520 *     VOLUME READ.         *
F1BD:          521 *                          *
F1BD:          522 *   USES TEMPS COUNT,      *
F1BD:          523 *     LAST, CSUM, AND      *
F1BD:          524 *     4 BYTES AT CSSTV     *
F1BD:          525 *                          *
F1BD:          526 *   ---- EXPECTS ----      *
F1BD:          527 *                          *
F1BD:          528 *  ORIGINAL 10-SECTOR      *
F1BD:          529 * NORMAL DENSITY NIBLS     *
F1BD:          530 *  (4-BIT), ODD BITS,      *
F1BD:          531 *  THEN EVEN               *
F1BD:          532 *                          *
F1BD:          533 *   ---- CAUTION ----      *
F1BD:          534 *                          *
F1BD:          535 *        OBSERVE           *
F1BD:          536 *   'NO PAGE CROSS'        *
F1BD:          537 *     WARNINGS ON          *
F1BD:          538 *   SOME BRANCHES!!        *
F1BD:          539 *                          *
F1BD:          540 *   ---- ASSUMES ----      *
F1BD:          541 *                          *
F1BD:          542 *    1 USEC CYCLE TIME     *
F1BD:          543 *                          *
F1BD:          544 ****************************
F1BD: A0 FC    545 RDADR16 LDY   #$FC
F1BF: 84 95    546         STY   COUNT      ;'MUST FIND' COUNT.
```

```
F1C1 C8           RDASYN  INY
F1C2 D0 04                BNE   RDA1         ;LOW ORDER OF COUNT
F1C4 E6 95                INC   COUNT        ;(2K NIBLS TO FIND
F1C6 F0 F0                BEQ   RDERR        ; ADR MARK, ELSE ERR)
F1C8 BD 8C C0     RDA1    LDA   Q6L,X        ;READ NIBL.
F1CB 10 FB                BPL   RDA1         ;*** NO PAGE CROSS! ***
F1CD C9 D5        RDASN1  CMP   #$D5         ;ADR MARK 1?
F1CF D0 F0                BNE   RDASYN       ; (LOOP IF NOT)
F1D1 EA                   NOP                ;ADDED NIBL DELAY
F1D2 BD 8C C0     RDA2    LDA   Q6L,X
F1D5 10 FB                BPL   RDA2         ;*** NO PAGE CROSS! ***
F1D7 C9 AA                CMP   #$AA         ;ADR MARK 2?
F1D9 D0 F2                BNE   RDASN1       ; (IF NOT, IS IT AM1?)
F1DB A0 03                LDY   #$3          ;INDEX FOR 4-BYTE READ
F1DD              *             (ADDED NIBL DELAY)
F1DD BD 8C C0     RDA3    LDA   Q6L,X
F1E0 10 FB                BPL   RDA3         ;*** NO PAGE CROSS! ***
F1E2 C9 96                CMP   #$96         ;ADR MARK 3?
F1E4 D0 E7                BNE   RDASN1       ; (IF NOT, IS IT AM1?)
F1E6              *       (LEAVES CARRY SET!)
F1E6 A9 00                LDA   #$0          ;INIT CHECKSUM
F1E8 85 89        RDAFLD  STA   CSUM
F1EA BD 8C C0     RDA4    LDA   Q6L,X        ;READ 'ODD BIT' NIBL
F1ED 10 FB                BPL   RDA4         ;*** NO PAGE CROSS! ***
F1EF 2A                   ROL   A            ;ALIGN ODD BITS, '1' INTO LSB
F1F0 85 95                STA   LAST         ; (SAVE THEM)
F1F2 BD 8C C0     RDA5    LDA   Q6L,X        ;READ 'EVEN BIT' NIBL
F1F5 10 FB                RPL   RDA5         ;*** NO PAGE CROSS! ***
F1F7 25 95                AND   LAST         ;MERGE ODD AND EVEN BITS
F1F9 99 97 00             STA   CSSTV,Y      ;STORE DATA BYTE
F1FC 45 89                EOR   CSUM
F1FE 88                   DEY
F1FF 10 E7                BPL   RDAFLD       ;LOOP ON 4 DATA BYTES
F201 A8                   TAY                ;IF FINAL CHECKSUM
F202 D0 B4                BNE   RDERR        ;NONZERO, THEN ERROR
F204 BD 8C C0     RDA6    LDA   Q6L,X        ;FIRST BIT-SLIP NIBL
F207 10 FB                BPL   RDA6         ;*** NO PAGE CROSS! ***
F209 C9 DE                CMP   #$DE
F20B D0 AB                BNE   RDERR        ;ERROR IF NONMATCH
F20D 78                   SET                ;DELAY (NO INTERUPTS FROM NOW ON)
F20E BD 8C C0     RDA7    LDA   Q6L,X        ;SECOND BIT-SLIP NIBL
F211 10 FB                BPL   RDA7         ;*** NO PAGE CROSS! ***
F213 C9 AA                CMP   #$AA
F215 D0 A1                BNE   RDERR        ;ERROR IF NONMATCH
F217 18           RDEXIT  CLC                ;CLEAR CARRY ON
F218 60           WEXIT   RTS                ; NORMAL READ EXITS
F219                      CHN   RWTS2
F219:             ************************
F219:             *                      *
F219              *      WRITE SUBR      *
F219              *  (16-SECTOR FORMAT)  *
F219              *                      *
F219              ************************
F219              *                      *
F219              *   WRITES DATA FROM   *
F219              *    NBUF1 AND NBUF2   *
F219              *                      *
F219              *  FIRST NBUF2,        *
F219              *      HIGH TO LOW.    *
F219              *  THEN NBUF1,         *
F219              *      LOW TO HIGH     *
F219              *                      *
F219              *  ---- ON ENTRY ----  *
F219              *                      *
F219              *   X-REG  SLOTNUM     *
F219              *        TIMES $10.    *
F219              *                      *
F219              *  ---- ON EXIT -----  *
F219              *                      *
F219              *  CARRY SET IF ERROR. *
```



```
F219               *   (W PROT VIOLATION)  *
F219               *                       *
F219               *  IF NO ERROR.         *
F219               *                       *
F219               *     A-REG UNCERTAIN   *
F219               *     X-REG UNCHANGED   *
F219               *     Y-REG HOLDS $00   *
F219               *     CARRY CLEAR       *
F219               *                       *
F219               *  ---- ASSUMES ----    *
F219               *                       *
F219               *  1 USEC CYCLE TIME    *
F219               *                       *
F219               *************************
F219 38            WRITE16 SEC             ;ANTICIPATE WPROT ERR.
F21A B8                    CLV             ;TO INDICATE WRITE PROTECT ERROR INSTEAD OF
                                                                             INTERUPT
F21B BD 8D C0              LDA  Q6H,X
F21E BD 8E C0              LDA  Q7L,X      ;SENSE WPROT FLAG
F221 30 F5                 BMI  WEXIT      ;BRANCH IF NOT WRITE PROTECTED
F223 A9 FF         WRT1    LDA  #$FF       ;SYNC DATA.
F225 9D 8F C0              STA  Q7H,X       ;(5) GOTO WRITE MODE
F228 1D 8C C0              ORA  Q6L,X       ;(4)
F22B A0 04                 LDY  #$4         ;(2) FOR FIVE NIBLS
F22D EA                    NOP             ;(2)
F22E 48                    PHA             ;(4)
F22F 68                    PLA             ;(3)
F230 48            WSYNC   PHA             ;(4) EXACT TIMING
F231 68                    PLA             ;(3) EXACT TIMING
F232 20 BD F2              JSR  WNIBL7     ;(13,9,6)  WRITE SYNC
F235 88                    DEY             ;(2)
F236 D0 F8                 BNE  WSYNC      ;(2*)  MUST NOT CROSS PAGE!
F238 A9 D5                 LDA  #$D5       ;(2)  1ST DATA MARK
F23A 20 BC F2              JSR  WNIBL9     ;(15,9,6)
F23D A9 AA                 LDA  #$AA       ;(2)  2ND DATA MARK
F23F: 20 BC F2             JSR  WNIBL9      ;(15,9,6)
F242: A9 AD                LDA  #$AD        ;(2)  3RD DATA MARK.
F244: 20 BC F2             JSR  WNIBL9      ;(15,9,6)
F247: A0 55                LDY  #$55        ;(2) NBUF2 INDEX
F249: EA                   NOP              ;(2) FOR TIMING
F24A: EA                   NOP              ;(2)
F24B: EA                   NOP              ;(2)
F24C: D0 08                BNE  VRYFRST     ;(3) BRANCH ALWAYS
F24E: AD EF FF     WINTRPT LDA  INTERUPT    ;(4) POLL INTERUPT LINE
F251: 05 8B                ORA  IMASK       ;(3)
F253: EA                   NOP              ;(2)
F254: 10 5D                BPL  SERVICE     ;(2) BRANCH IF INTERUPT HAS OCCURED
F256: 30 00        VRYFRST BMI  WRTFRST     ;(3) FOR TIMING.
F258: B9 02 03     WRTFRST LDA  NBUF2,Y      ;(4)
F25B: 9D 8D C0             STA  Q6H,X       ;(5) STORE ENCODED BYTE
F25E: BD 8C C0             LDA  Q6L,X       ;(4) TIME MUST = 32 US PER BYTE!
F261: 88                   DEY              ;(2)
F262: 10 EA                BPL  WINTRPT     ;(3) (2 IF BRANCH NOT TAKEN)
F264: 98                   TYA               ;(2) INSURE NO INTERUPT THIS BYTE.
F265: 30 03                BMI  WMIDLE      ;(3) BRANCH ALWAYS.
F267: AD EF FF     WNTRPT1 LDA  INTERUPT    ;(4) POLL INTERUPT LINE
F26A: 05 8B        WMIDLE  ORA  IMASK       ;(3)
F26C: EA                   NOP              ;(2)
F26D: 30 02                BMI  WDATA2       ;(3) BRANCH IF NO INTERUPT
F26F: 10 42                BPL  SERVICE     ;GO SERVICE INTERUPT.
F271: C8           WDATA2  INY               ;(2)
F272: B9 00 02             LDA  NBUF1,Y      ;(4)
F275: 9D 8D C0             STA  Q6H,X       ;(5) STORE ENCODED BYTE
F278: BD 8C C0             LDA  Q6L,X       ;(4)
F27B: C0 E4                CPY  #$E4        ;(2) WITHIN 1 MS OF COMPLETION?
F27D: D0 E8                BNE  WNTRPT1     ;(3) (2) NO KEEP WRITTING AND POLLING.
F27F: EA                   NOP               ;(2)
F280: C8                   INY               ;(2)
F281: EA           WDATA3  NOP              ;(2)
F282: EA                   NOP              ;(2)
F283: 48                   PHA              ;(4)
F284: 68                   PLA              ;(3)
F285: B9 00 02             LDA  NBUF1,Y     ;(4) WRITE LAST OF ENCODED BYTES
```

"APPLE_PAT_4_383_296_28" 158 KB 2000-02-27 dpi: 300h x 300v pix: 1880h x 2774v

```
F288 9D 8D C0              STA   Q6H,X      ;(5)  WITHOUT POLLING INTERUPTS.
F28B BD 8C C0              LDA   Q6L,X      ;(4)
F28E A5 96                 LDA   CKSUM      ;(3) NORMALLY FOR TIMING
F290:C8                    INY              ;(2)
F291 D0 EE                 BNE   WDATA3     ;(3) (2)
F293:F0 00                 BEQ   WRCKSUM    ;(3) BRANCH ALWAYS
F295 20 BD F2     WRCKSUM  JSR   WNIBL7     ;(13,9,6) GO WRITE CHECK SUM..
F298 A9 DE                 LDA   #$DE       ;(2)  DM4, BIT SLIP MARK.
F29A 20 BC F2              JSR   WNIBL9     ;(15,9,6)    WRITE IT
F29D:A9 AA                 LDA   #$AA       ;(2)  DM5, BIT SLIP MARK
F29F 20 BC F2              JSR   WNIBL9     ;(15,9,6)    WRITE IT.
F2A2 A9 EB                 LDA   #$EB       ;(2)  DM6, BIT SLIP MARK.
F2A4 20 BC F2              JSR   WNIBL9     ;(15,9,6)    WRITE IT.
F2A7.A9 FF                 LDA   #$FF       ;(2) TURN-OFF BYTE
F2A9 20 BC F2              JSR   WNIBL9     ;(15,9,9)  WRITE IT
F2AC:BD 8E C0     NOWRITE  LDA   Q7L,X      ;OUT OF WRITE MODF
F2AF:BD 8C C0              LDA   Q6L,X      ;TO READ MODE.
F2B2:60                    RTS              ;RETURN FROM WRITE.
F2B3:             *
F2B3 38           SERVICE  SEC              ;TREAT INTERUPTION AS ERROR
F2B4:2C 54 F3              BIT   SEV        ;SET VFLAG TO INDICATE INTERUPT
F2B7:20 AC F2              JSR   NOWRITE    ;TAKE IT OUT OF WRITE MODE'
F2BA 58                    CLI              ;COULD NOT HAVE GOT HERE WITHOUT CLI OK
F2BB 60                    RTS
F2BC              ******************************
F2BC              *                            *
F2BC              *    7-BIT NIBL WRITE SUBRS  *
F2BC              *                            *
F2BC              *    A-REG OR'D PRIOR EXIT   *
F2BC              *         CARRY CLEARED      *
F2BC              *                            *
F2BC              ******************************
F2BC 18           WNIBL9   CLC              ;(2)  9 CYCLES, THEN WRITE
F2BD 48           WNIBL7   PHA              ;(3)  7 CYCLES, THEN WRITE
F2BE 68                    PLA              ;(4)
F2BF 9D 8D C0     WNIBL    STA   Q6H,X      ;(5)  NIBL WRITE SUB
F2C2 1D 8C C0              ORA   Q6L,X      ;(4)  CLOBBERS ACC  NOT CARRY
F2C5 60                    RTS
F2C6              *
F2C6              ******************************
F2C6              *                            *
F2C6              *     PRENIBLIZE SUBR        *
F2C6              *    (16-SECTOR FORMAT)      *
F2C6              *                            *
F2C6              ******************************
F2C6              *                            *
F2C6              *  CONVERTS 256 BYTES OF     *
F2C6              *  USER DATA IN (BUF) INTO   *
F2C6              *  ENCODED BYTES TO BE       *
F2C6              *  WRITEN DIRECTLY TO DISK   *
F2C6              *  ENCODED CHECK SUM IN      *
F2C6              *  ZERO PAGE 'CKSUM'         *
F2C6              *                            *
F2C6              *     ---- ON ENTRY ----     *
F2C6              *                            *
F2C6              *  BUF IS 2-BYTE POINTER     *
F2C6              *    TO 256 BYTES OF USER    *
F2C6              *    DATA.                   *
F2C6              *                            *
F2C6              *     ----- ON EXIT -----    *
F2C6              *                            *
F2C6              *  A-REG CHECK SUM.          *
F2C6              *  X-REG UNCERTAIN           *
F2C6              *  Y-REG HOLDS 0.            *
F2C6              *  CARRY SET.                *
F2C6              *                            *
F2C6              ******************************
F2C6 A2 02        PRENIB16 LDX   #$2        ;START NBUF2 INDEX.
F2C8 A0 00                 LDY   #0         ;START USER BUF INDEX.
F2CA:88           PRENIB1  DEY              ;NEXT USER BYTE
F2CB B1 9B                 LDA   (BUF),Y
F2CD 4A                    LSR   A          ;SHIFT TWO BITS OF
F2CE 3E 01 03              ROL   NBUF2-1,X  ;CURRENT USER BYTE
```

```
F2D1: 4A            172          LSR  A           ;INTO CURRENT NBUF2
F2D2: 3E 01 03      173          ROL  NBUF2-1,X   ;BYTE.
F2D5: 99 01 02      174          STA  NBUF1+1,Y   ;(6 BITS LEFT)
F2D8: E8            175          INX              ;FROM 0 TO $55
F2D9: E0 56         176          CPX  #$56
F2DB: 90 ED         177          BCC  PRENIB1     ;BR IF NO WRAPAROUND
F2DD: A2 00         178          LDX  #0          ;RESET NBUF2 INDEX
F2DF: 98            179          TYA              ;USER BUF INDEX
F2E0: D0 E8         180          BNE  PRENIB1     ;(DONE IF ZERO)
F2E2: A0 56         181          LDY  #$56        ;(ACC=0 FOR CHECK SUM)
F2E4: 59 00 03      182 PRENIB3  EOR  NBUF2-2,Y   ;COMBINE WITH PREVIOUS
F2E7: 29 3F         183 PRENIB2  AND  #$3F        ;STRIP GARBAGE BITS
F2E9: AA            184          TAX              ; TO FORM RUNNING CHECK SUM
F2EA: BD 55 F3      185          LDA  NIBL,X      ;GET ENCODED EQUIV
F2ED: 99 01 03      186          STA  NBUF2-1,Y   ;REPLACE PREVIOUS
F2F0: B9 00 03      187          LDA  NBUF2-2,Y   ;RESTORE ACTUAL PREVIOUS
F2F3: 88            188          DEY
F2F4: D0 EE         189          BNE  PRENIB3     ;LOOP UNTIL ALL OF NBUF2 IS CONVERTED
F2F6: 29 3F         190          AND  #$3F
F2F8: 59 01 02      191 PRENIB4  EOR  NBUF1+1,Y   ;NOW DO THE SAME FOR
F2FB: AA            192          TAX              ;NIBBLE BUFFER 1
F2FC: BD 55 F3      193          LDA  NIBL,X      ;TO DO ANY BACK TRACKING (NBUF1-1)
F2FF: 99 00 02      194          STA  NBUF1,Y
F302: B9 01 02      195          LDA  NBUF1+1,Y   ;RECOVER THAT WHICH IS NOW 'PREVIOUS'
F305: C8            196          INY
F306: D0 F0         197          BNE  PRENIB4
F308: AA            198          TAX              ;USE LAST AS CHECK SUM
F309: BD 55 F3      199          LDA  NIBL,X
F30C: 85 96         200          STA  CKSUM
F30E: 4C 4C F3      201          JMP  SET1MEG     ;ALL DONE.
F311:               203 **************************
F311:               204 *                        *
F311:               205 *    POSTNIBLIZE SUBR    *
F311:               206 *    16-SECTOR FORMAT    *
F311:               207 *                        *
F311:               208 **************************
F311:               209 *
F311: A0 55         210 POSTNIB16 LDY #$55        ;FIRST CONVERT TO 6 BIT NIBBLES
F313: A9 00         211          LDA  #$0         ;INIT CHECK SUM
F315: BE 02 03      212 PNIBL1   LDX  NBUF2,Y     ;GET ENCODED BYTE
F318: 5D 00 F3      213          EOR  DNIBL,X
F31B: 99 02 03      214          STA  NBUF2,Y     ;REPLACE WITH 6 BIT EQUIV
F31E: 88            215          DEY
F31F: 10 F4         216          BPL  PNIBL1      ;LOOP UNTIL DONE WITH NIBBLE BUFFER 2
F321: C8            217          INY              ;NOW Y=0
F322: BE 00 02      218 PNIBL2   LDX  NBUF1,Y     ;DO THE SAME WITH
F325: 5D 00 F3      219          EOR  DNIBL,X
F328: 99 00 02      220          STA  NBUF1,Y     ; NIBBLE BUFFER 1
F32B: C8            221          INY              ;DO ALL 256 BYTES
F32C: D0 F4         222          BNE  PNIBL2
F32E: A6 96         223          LDX  CKSUM       ;MAKE SURE CHECK SUM MATCHES
F330: 5D 00 F3      224          EOR  DNIBL,X     ;BETTER BE ZERO!
F333: 38            225          SEC              ;ANTICIPATE ERROR
F334: D0 16         226          BNE  POSTERR     ;BRANCH IF IT IS
F336: A2 56         227 POST1    LDX  #$56        ;INIT NBUF2 INDEX
F338: CA            228 POST2    DEX              ;NBUF IDX $55 TO $0.
F339: 30 FB         229          BMI  POST1       ;WRAPAROUND IF NEG
F33B: B9 00 02      230          LDA  NBUF1,Y
F33E: 5E 02 03      231          LSR  NBUF2,X     ;SHIFT 2 BITS FROM
F341: 2A            232          ROL  A           ;CURRENT NBUF2 NIBL
F342: 5E 02 03      233          LSR  NBUF2,X     ;INTO CURRENT NBUF1
F345: 2A            234          ROL  A           ;NIBL
F346: 91 9B         235          STA  (BUF),Y     ;BYTE OF USER DATA
F348: C8            236          INY              ;NEXT USER BYTE
F349: D0 ED         237          BNE  POST2
F34B: 18            238          CLC              ;GOOD DATA
F34C:               239 POSTERR  EQU  *
F34C: AD DF FF      240 SET1MEG  LDA  ENVIRON
F34F: 09 80         241          ORA  #ONEMEG     ;SET TO ONE MEGAHERTZ CLOCK RATE
F351: 8D DF FF      242          STA  ENVIRON
F354: 60            243 SEV      RTS              ;(SEV USED TO SET VFLAG)
```

```
F355:             **************************
F355:             *                        *
F355:             *     6-BIT TO 7-BIT     *
F355:             *  NIBL CONVERSION TABLE *
F355:             *                        *
F355:             **************************
F355:             *                        *
F355:             *   CODES WITH MORE THAN *
F355:             *   ONE PAIR OF ADJACENT *
F355:             *    ZEROES OR WITH NO   *
F355:             *   ADJACENT ONES (EXCEPT *
F355:             *     B7) ARE EXCLUDED.  *
F355:             *                        *
F355:             **************************
F355:96 97 9A     NIBL     DFB  $96,$97,$9A
F358:9B 9D 9E              DFB  $9B,$9D,$9E
F35B:9F A6 A7              DFB  $9F,$A6,$A7
F35E:AB AC AD              DFB  $AB,$AC,$AD
F361:AE AF B2              DFB  $AE,$AF,$B2
F364:B3 B4 B5              DFB  $B3,$B4,$B5
F367:B6 B7 B9              DFB  $B6,$B7,$B9
F36A:BA BB BC              DFB  $BA,$BB,$BC
F36D:BD BE BF              DFB  $BD,$BE,$BF
F370:CB CD CE              DFB  $CB,$CD,$CE
F373:CF D3 D6              DFB  $CF,$D3,$D6
F376:D7 D9 DA              DFB  $D7,$D9,$DA
F379:DB DC DD              DFB  $DB,$DC,$DD
F37C:DE DF E5              DFB  $DE,$DF,$E5
F37F:E6 E7 E9              DFB  $E6,$E7,$E9
F382:EA EB EC              DFB  $EA,$EB,$EC
F385:ED EE EF              DFB  $ED,$EE,$EF
F388:F2 F3 F4              DFB  $F2,$F3,$F4
F38B:F5 F6 F7              DFB  $F5,$F6,$F7
F38E:F9 FA FB              DFB  $F9,$FA,$FB
F391:FC FD FE              DFB  $FC,$FD,$FE
F394:FF                    DFB  $FF
F395:             **************************
F395:             *                        *
F395:             *     7-BIT TO 6-BIT     *
F395:             *    'DENIBLIZE' TABL    *
F395:             *   (16-SECTOR FORMAT)   *
F395:             *                        *
F395:             *      VALID CODES       *
F395:             *    $96 TO $FF ONLY.    *
F395:             *                        *
F395:             *                        *
F395:             *  CODES WITH MORE THAN  *
F395:             *  ONE PAIR OF ADJACENT  *
F395:             *   ZEROES OR WITH NO    *
F395:             *  ADJACENT ONES (EXCEPT *
F395:             *   BIT 7) ARE EXCLUDED  *
F395:             **************************
F395:00                    BRK                  ONE BYET LEFT OVER
F300:             DNIBL    EQU  REGRWTS+$300
F396:00 01 98              DFB  $00,$01,$98
F399:99 02 03              DFB  $99,$02,$03
```

```
F39C: 9C 04 05          DFB   $9C,$04,$05
F39F: 06 A0 A1          DFB   $06,$A0,$A1
F3A2: A2 A3 A4          DFB   $A2,$A3,$A4
F3A5: A5 07 08          DFB   $A5,$07,$08
F3A8: A8 A9 AA          DFB   $A8,$A9,$AA
F3AB: 09 0A 0B          DFB   $09,$0A,$0B
F3AE: 0C 0D B0          DFB   $0C,$0D,$B0
F3B1: B1 0E 0F          DFB   $B1,$0E,$0F
F3B4: 10 11 12          DFB   $10,$11,$12
F3B7: 13 B8 14          DFB   $13,$B8,$14
F3BA: 15 16 17          DFB   $15,$16,$17
F3BD: 18 19 1A          DFB   $18,$19,$1A
F3C0: C0 C1 C2          DFB   $C0,$C1,$C2
F3C3: C3 C4 C5          DFB   $C3,$C4,$C5
F3C6: C6 C7 C8          DFB   $C6,$C7,$C8
F3C9: C9 CA 1B          DFB   $C9,$CA,$1B
F3CC: CC 1C 1D          DFB   $CC,$1C,$1D
F3CF: 1E D0 D1          DFB   $1E,$D0,$D1
F3D2: D2 1F D4          DFB   $D2,$1F,$D4
F3D5: D5 20 21          DFB   $D5,$20,$21
F3D8: D8 22 23          DFB   $D8,$22,$23
F3DB: 24 25 26          DFB   $24,$25,$26
F3DE: 27 28 E0          DFB   $27,$28,$E0
F3E1: E1 E2 E3          DFB   $E1,$E2,$E3
F3E4: E4 29 2A          DFB   $E4,$29,$2A
F3E7: 2B E8 2C          DFB   $2B,$E8,$2C
F3EA: 2D 2E 2F          DFB   $2D,$2E,$2F
F3ED: 30 31 32          DFB   $30,$31,$32
F3F0: F0 F1 33          DFB   $F0,$F1,$33
F3F3: 34 35 36          DFB   $34,$35,$36
F3F6: 37 38 F8          DFB   $37,$38,$F8
F3F9: 39 3A 3B          DFB   $39,$3A,$3B
F3FC: 3C 3D 3E          DFB   $3C,$3D,$3E
F3FF: 3F                DFB   $3F
F400:             **************************
F400:             *                        *
F400:             *  FAST SEEK SUBROUTINE  *
F400:             *                        *
F400:             **************************
F400:             *                        *
F400:             *   ---- ON ENTRY ----   *
F400:             *                        *
F400:             *  X-REG HOLDS SLOTNUM   *
F400:             *        TIMES $10.      *
F400:             *                        *
F400:             *  A-REG HOLDS DESIRED   *
F400:             *        HALFTRACK.      *
F400:             *        (SINGLE PHASE)  *
F400:             *                        *
F400:             *  CURTRK HOLDS CURRENT  *
F400:             *          HALFTRACK.    *
F400:             *                        *
F400:             *   ---- ON EXIT -----   *
F400:             *                        *
F400:             *  A-REG UNCERTAIN.      *
F400:             *  Y-REG UNCERTAIN.      *
F400:             *  X-REG UNDISTURBED.    *
F400:             *                        *
F400:             *  CURTRK AND TRKN HOLD  *
F400:             *      FINAL HALFTRACK.  *
```

“APPLE_PAT_4_383_296_32” 144 KB 2000-02-27 dpi: 300h x 300v pix: 1460h x 2755v

```
F400:            *                          *
F400:            *  PRIOR HOLDS PRIOR       *
F400:            *     HALFTRACK IF SEEK    *
F400:            *     WAS REQUIRED.        *
F400:            *                          *
F400:            *  MONTIMEL AND MONTIMEH   *
F400:            *     ARE INCREMENTED BY   *
F400:            *     THE NUMBER OF        *
F400:            *     100 USEC QUANTUMS    *
F400:            *     REQUIRED BY SEEK     *
F400:            *     FOR MOTOR ON TIME    *
F400:            *     OVERLAP.             *
F400:            *                          *
F400:            * --- VARIABLES USED ---   *
F400:            *                          *
F400:            *  CURTRK, TRKN, COUNT,    *
F400:            *     PRIOR, SLOTTEMP      *
F400:            *     MONTIMEL, MONTIMEH   *
F400:            *                          *
F400:            ****************************
F400: 85 9E      SEEK     STA  TRKN        ;SAVE TARGET TRACK
F402: C5 8C               CMP  CURTRK      ;ON DESIRED TRACK?
F404: F0 42               BEQ  SETPHASE    ;YES,ENERGIZE PHASE AND RETURN
F406: A9 00               LDA  #$0
F408: 85 95               STA  TRKCNT      ;HALFTRACK COUNT.
F40A: A5 8C      SEEK2    LDA  CURTRK      ;SAVE CURTRK FOR
F40C: 85 9D               STA  PRIOR       ;DELAYED TURNOFF.
F40E: 38                  SEC
F40F: E5 9E               SBC  TRKN        ;DELTA-TRACKS.
F411: F0 31               BEQ  SEEKEND     ;BR IF CURTRK=DESTINATION
F413: B0 06               BCS  OUT         (MOVE OUT, NOT IN)
F415: 49 FF               EOR  #$FF        CALC TRKS TO GO
F417: E6 8C               INC  CURTRK      INCR CURRENT TRACK (IN)
F419: 90 04               BCC  MINTST      (ALWAYS TAKEN).
F41B: 69 FE      OUT      ADC  #$FE        ;CALC TRKS TO GO
F41D: C6 8C               DEC  CURTRK      ;DECR CURRENT TRACK (OUT)
F41F: C5 95      MINTST   CMP  TRKCNT
F421: 90 02               BCC  MAXTST       AND 'TRKS MOVED'
F423: A5 95               LDA  TRKCNT
F425: C9 09      MAXTST   CMP  #$9
F427: B0 02               BCS  STEP2       ;IF TRKCNT>$8 LEAVE Y ALONE (Y=$8)
F429: A8         STEP     TAY              ;ELSE SET ACCELERATION INDEX IN Y
F42A: 38                  SEC
F42B: 20 48 F4   STEP2    JSR  SETPHASE
F42E: B9 67 F4            LDA  ONTABLE,Y   ;FOR 'ONTIME'
F431: 20 56 F4            JSR  MSWAIT      ;(100 USEC INTERVALS)
F434: A5 9D               LDA  PRIOR
F436: 18                  CLC              ;FOR PHASEOFF
F437: 20 4A F4            JSR  CLRPHASE    ;TURN OFF PRIOR PHASE
F43A: B9 70 F4            LDA  OFFTABLE,Y   THEN WAIT 'OFFTIME'
F43D: 20 56 F4            JSR  MSWAIT      (100 USEC INTERVALS)
F440: E6 95               INC  TRKCNT      'TRACKS MOVED' COUNT.
F442: D0 C6               BNE  SEEK2       (ALWAYS TAKEN)
F444: 20 56 F4   SEEKEND  JSR  MSWAIT      ;SETTLE 25 MSEC
F447: 18                  CLC              ;SET FOR PHASE OFF
F448: A5 8C      SETPHASE LDA  CURTRK      ;GET CURRENT TRACK
F44A: 29 03      CLRPHASE AND  #3          ;MASK FOR 1 OF 4 PHASES
F44C: 2A                  ROL  A           ;DOUBLE FOR PHASEON/OFF INDEX
F44D: 05 81               ORA  IBSLOT
F44F: AA                  TAX
F450: BD 80 C0            LDA  PHASEOFF,X  ;TURN ON/OFF ONE PHASE
F453: A6 81               LDX  IBSLOT      ;RESTORE X-REG
F455: 60         SEEKRTS  RTS              ;AND RETURN
F456:            ***************************
F456:            *                         *
F456:            *   MSWAIT SUBROUTINE     *
F456:            *                         *
```

```
F456                431 ***************************
F456                432 *                         *
F456                433 *  DELAYS A SPECIFIED     *
F456                434 *   NUMBER OF 100 USEC    *
F456                435 *   INTERVALS FOR MOTOR   *
F456                436 *   ON TIMING.            *
F456                437 *                         *
F456                438 *   ---- ON ENTRY ----    *
F456                439 *                         *
F456                440 *  A-REG  HOLDS NUMBER    *
F456                441 *         OF 100 USEC     *
F456                442 *         INTERVALS TO    *
F456                443 *         DELAY.          *
F456                444 *                         *
F456                445 *   ---- ON EXIT ----     *
F456                446 *                         *
F456                447 *  A-REG  HOLDS $00       *
F456                448 *  X-REG  HOLDS $00       *
F456                449 *  Y-REG  UNCHANGED       *
F456                450 *  CARRY  SET             *
F456                451 *                         *
F456                452 *  MONTIMEL, MONTIMEH     *
F456                453 *   ARE INCREMENTED ONCE  *
F456                454 *   PER 100 USEC INTERVAL *
F456                455 *   FOR MOTOR ON TIMING.  *
F456                456 *                         *
F456                457 *     --- ASSUMES ---     *
F456                458 *                         *
F456                459 *   1 USEC CYCLE TIME     *
F456                460 *                         *
F456                461 ***************************
F456 A2 11          462 MSWAIT   LDX  #$11
F458 CA             463 MSW1     DEX              DELAY 86 USEC
F459 D0 FD          464          BNE  MSW1
F45B E6 [illegible] 465          INC  MONTIMEL
F45D D0 02          466          BNE  MSW2        DOUBLE-BYTE
F45F E6 [illegible] 467          INC  MONTIMEH     INCREMENT
F461 38             468 MSW2     SEC
F462 E9 01          469          SBC  #$1         DONE N INTERVALS?
F464 D0 F0          470          BNE  MSWAIT      (A-REG COUNTS)
F466 60             471          RTS
F467                472 *
F467:               474 ***************************
F467:               475 *                         *
F467:               476 *  PHASE ON-, OFF-TIME    *
F467:               477 *   TABLES IN 100-USEC    *
F467:               478 *   INTERVALS. (SEEK)     *
F467:               479 *                         *
F467:               480 ***************************
F467:01 30 28       481 ONTABLE DFB  1,$30,$28
F46A:24 20 1E       482         DFB  $24,$20,$1E
F46D:1D 1C 1C       483         DFB  $1D,$1C,$1C
F470:70 2C 26       484 OFFTABLE DFB $70,$2C,$26
F473:22 1F 1E       485         DFB  $22,$1F,$1E
F476:1D 1C 1C       486         DFB  $1D,$1C,$1C
```

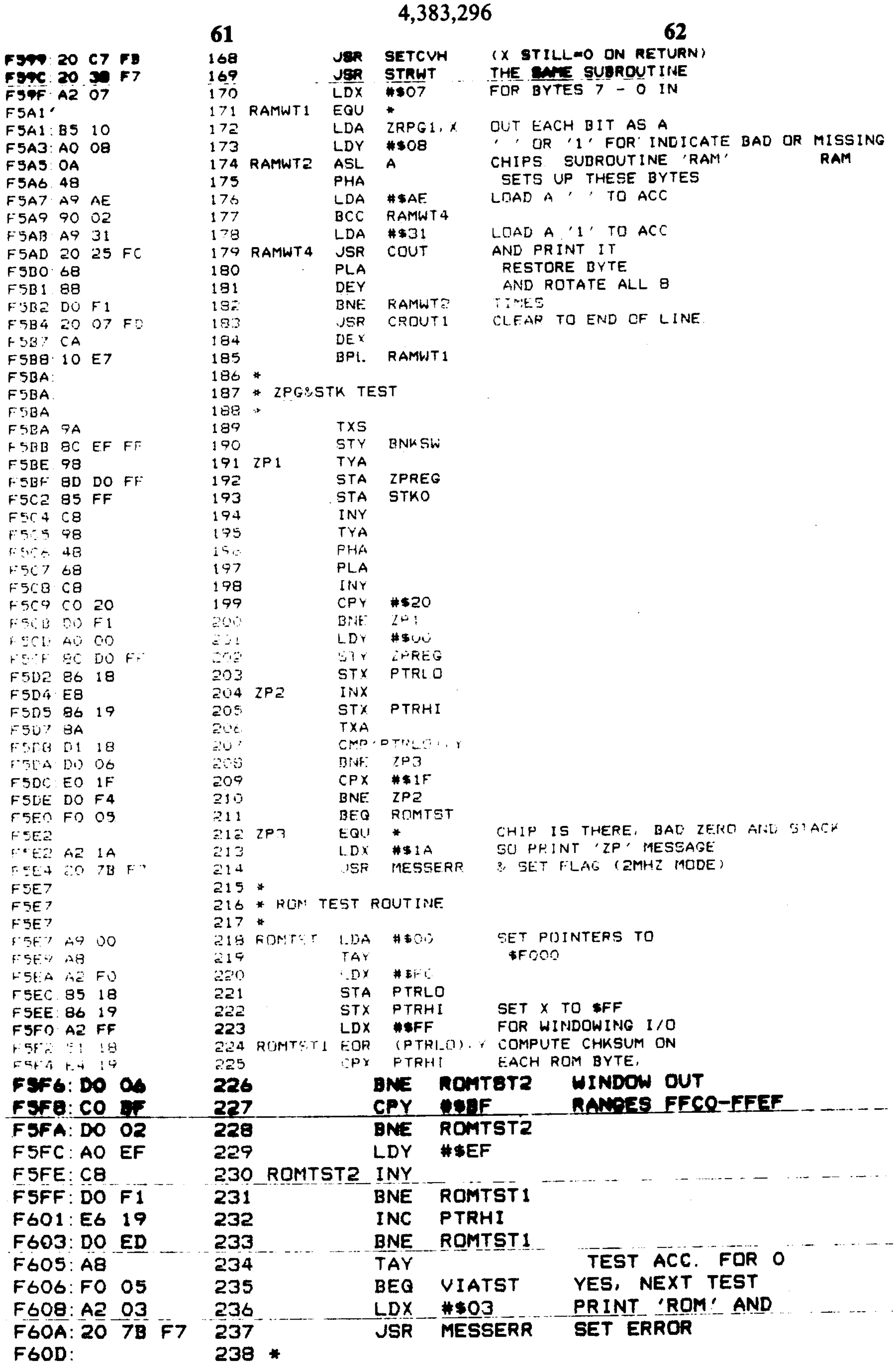
```
F479 86 83       488 BLOCKIO STX   IBTRK
F47B A0 05       489         LDY   #$5
F47D 48          490         PHA
F47E 0A          491 TRKSEC  ASL   A
F47F 26 83       492         ROL   IBTRK
F481 88          493         DEY
F482 D0 FA       494         BNE   TRKSEC
F484 68          495         PLA
F485 29 07       496         AND   #$7
F487 A8          497         TAY
F488 B9 A0 F4    498         LDA   SECTABL,Y
F48B 85 84       499         STA   IBSECT
F48D 20 00 F0    500         JSR   REGRWTS
F490 B0 0B       501         BCS   QUIT
F492 E6 86       502         INC   IBBUFP+1
F494 E6 84       503         INC   IBSECT
F496 E6 84       504         INC   IBSECT
F498 20 00 F0    505         JSR   REGRWTS
F49B C6 86       506         DEC   IBBUFP+1
F49D A5 88       507 QUIT    LDA   IBSTAT
F49F 60          508         RTS
F4A0             509 *
F4A0             510 SECTABI EQU   *
F4A0 00 04 08    511         DFB   $0,$4,$8
F4A3 0C 01 05    512         DFB   $C,$1,$5
F4A6 09 0D       513         DFB   $9,$D
F4A8             514 *
F4A8          516 * * * * * * * * * * * * * * * * * *
F4A8          517 *                                 *
F4A8          518 *   JOYSTICK READ ROUTINE         *
F4A8          519 *                                 *
F4A8          520 * * * * * * * * * * * * * * * * * *
F4A8          521 * ENTRY  ACC= COUNT DOWN HIGH     *
F4A8          522 *        X&Y= DON'T CARE          *
F4A8          523 *                                 *
F4A8          524 *  EXIT  ACC= TIMER HIGH BYTE     *
F4A8          525 *          Y= TIMER LOW BYTE      *
F4A8          526 *        CARRY CLEAR              *
F4A8          527 *                                 *
F4A8          528 *    IF CARRY SET, ROUTINE        *
F4A8          529 *       WAS INTERUPTED &          *
F4A8          530 *     ACC & Y ARE INVALID         *
F4A8          531 * * * * * * * * * * * * * * * * * *
F4A8          532 *
FFD9          533 TIMLATCH EQU $FFD9
FFD8          534 TIMER1L EQU  $FFD8
FFD9          535 TIMER1H EQU  $FFD9
C066          536 JOYRDY  EQU  $C066
F4A8          537 *
F4A8          538 ANALOG  EQU  *             ;CARRY SHOULD BE SET!
F4A8 8D D9 FF 539         STA  TIMLATCH      ;START THE TIMER!
F4AB AD EF FF 540 ANLOG1  LDA  INTERUPT
F4AE 2D 66 C0 541         AND  JOYRDY        ;WAIT FOR ONE OR THE OTHER TO GO LOW
F4B1 30 F8    542         BMI  ANLOG1
F4B3 AD 66 C0 543         LDA  JOYRDY        ;WAY IT REALLY THE JOYSTICK?
F4B6 30 0C    544         BMI  GOODTIME      ;NOPE, FORGET IT
F4B8 18       545         CLC                ;TIME'S A SLIP SLIDIN AWAY
F4B9 AD D9 FF 546         LDA  TIMER1H      ;NOW, WHAT TIME IS IT?
F4BC AC D8 FF 547         LDY  TIMER1L
F4BF 10 03    548         BPL  GOODTIME      ;TIME WAS VALID'
F4C1 AD D9 FF 549         LDA  TIMER1H       ;HI BYTE CHANGED
F4C4 60       550 GOODTIME RTS
*** SUCCESSFUL ASSEMBLY. NO ERRORS
```

```
 F0E9 ALDONE1
 F4AB ANLOG1
 F12B CHKDRV
 F050 CONWAIT
   97 CSSTV
 F300 DNIBL
?C08B DRV1EN
   E0 DVMOT
?F0A1 GOCAL
 F0E8 HNDLERR
   82 IBDERR
   84 IBSECT
   83 IBTRK
   8A IOBPDN
 F41F MINTST
 C088 MOTOROFF
 F456 MSWAIT
 F355 NIBL
 F2AC NOWRITE
   80 ONEMEG
?C081 PHASEON
 F322 PNIBL2
 F311 POSTNIB16
 F2E4 PRENIB3
 C08C Q6L
 F14D RD1
 F17E RD5A
 F1AF RD8
 F1EA RDA4
 F1BD RDADR16
 F19D RDCKSUM
 F148 READ16
 F14A RSYNC
?F106 SEEK1
 F444 SEEKEND
 F448 SETPHASE
?F429 STEP
 FFD9 TIMLATCH
   99 TRKN1
 F07B TRYTRK2
 F256 VRYFRST
 F24E WINTRPT
 F2BC WNIBL9
 F0F9 WRIT
   7F TWOMEG
   81 IBSLOT
   83 IBRERR
   85 IBBUFP
   89 IBSMOD
   93 RETRYCNT
   95 COUNT
   97 TEMP
   99 TRACK
   9D PRIOR
 0200 NBUF1
 C080 PHASEOFF
 C089 MOTORON
 C08D Q6H
 F031 DRIVSEL
 F050 CONWAIT
 F07F TRYADR
?F0A1 GOCAL
 F0C7 CORRECTSECT
 F0E9 ALDONE1
?F106 SEEK1
 F125 SETTRK

 F0E0 ALLDONE
?F479 BLOCKID
 F13D CKDRTS
 F0C7 CORRECTSECT
   97 CSUM1
 F031 DRIVSEL
 F0E5 DRVERR
 FFDF ENVIRON
 F4C4 GOODTIME
   80 HRDERRS
   82 IBDRVN
   81 IBSLOT
   81 IBWPER
 C066 JOYRDY
   9A MONTIMEH
 C089 MOTORON
 F105 MYSEEK
?F060 NODRIVERR
 F11B NXOFF
 F467 ONTABLE
?C081 PHASON
 F336 POST1
 F2CA PRENIB1
 F2F8 PRENIB4
 C08F Q7H
 F157 RD2
 F17D RD5
 F1C8 RDA1
 F1F2 RDA5
 F1E8 RDAFLD
 F1B8 RDERR
 F000 REGRWTS
 F0BB RTTRK
 F40A SEEK2
?F455 SEEKRTS
 F125 SETTRK
   97 TEMP
   99 TRACK
 F47E TRKSEC
 F065 TRYTRK
 F271 WDATA2
 F26A WMIDLE
 F267 WNTRPT1
?F223 WRT1
   80 IBNODRV
   81 IBWPER
   83 IBTRK
   87 IBCMD
   8A IOBPDN
   94 SEEKCNT
   96 CKSUM
   98 SECT
   9A MONTIMEH
   9E TRKN
 0302 NBUF2
?C081 PHASON
 C08A DRVOEN
 C08E Q7L
 F03D DRVWAIT
?F060 NODRIVERR
 F086 TRYADR2
 F0A7 RDRIGHT
 F0E0 ALLDONE
 F0F3 NOINTR2
?F116 GOSEEK
 F12B CHKDRV

 F119 ALLOFF
   9B BUF
   96 CKSUM
?F0DF CORRECTVOL
   89 CSUM
 C08A DRVOEN
 F13E DRVINDX
   9F ENVTEMP
?F116 GOSEEK
   85 IBBUFP
   80 IBNODRV
   89 IBSMOD
   8B IMASK
   95 LAST
   99 MONTIMEL
 F458 MSW1
 0200 NBUF1
 F08D NOINTR1
 F470 OFFTABLE
 F41D OUT
?C080 PHSOFF
 F338 POST2
 F2C6 PRENIB16
   9D PRIOR
 C08E Q7L
 F162 RD3
 F192 RD6
 F1D2 RDA2
 F204 RDA6
 F1CD RDASN1
 F217 RDEXIT
   93 RETRYCNT
 F4A0 SECTABL
   94 SEEKCNT
 F2B3 SERVICE
 F354 SEV
 FFD9 TIMER1H
   95 TRKCNT
 F086 TRYADR2
   7F TWOMEG
 F281 WDATA3
?F2BF WNIBL
 F295 WRCKSUM
 F258 WRTFRST
   80 HRDERRS
   82 IBDERR
   84 IBSECT
   88 IBSTAT
   8B IMASK
   95 LAST
   97 CSSTV
   99 MONTIMEL
   9A VOLUME
   9F ENVTEMP
 C066 JOYRDY
?C081 PHASEON
?C08B DRV1EN
 C08F Q7H
 F044 OK
 F065 TRYTRK
 F08B NOINTR1
 F0BB RTTRK
 F0E5 DRVERR
 F0F9 WRIT
 F119 ALLOFF
 F12D CHKDRV1

?F4A8 ANALOG
 F12D CHKDRV1
 F44A CLRPHASE
   95 COUNT
   8C CURTRK
   85 DRVOTRK
 F03D DRVWAIT
?F0A0 GOCAL1
 F1BA GOSERV
   87 IBCMD
?  83 IBRERR
   88 IBSTAT
 FFEF INTERUPT
 F425 MAXTST
 F04E MOTOF
 F461 MSW2
 0302 NBUF2
 F0F3 NOINTR2
 F044 OK
 C080 PHASEOFF
 F315 PNIBL1
 F34C POSTERR
?F2E7 PRENIB2
 C08D Q6H
 F49D QUIT
 F16B RD4
 F1A5 RD7
 F1DD RDA3
 F20E RDA7
 F1C1 RDASYN
 F0A7 RDRIGHT
 F152 RSYNC1
   98 SECT
 F400 SEEK
 F34C SET1MEG
 F42B STEP2
 FFD8 TIMER1L
   9E TRKN
 F07F TRYADR
   9A VOLUME
 F218 WEXIT
 F2BD WNIBL7
 F219 WRITE16
 F230 WSYNC
   80 ONEMEG
   82 IBDRVN
   85 DRVOTRK
   89 CSUM
   8C CURTRK
   95 TRKCNT
   97 CSUM1
   99 TRKN1
   9B BUF
   E0 DVMOT
?C080 PHSOFF
 C088 MOTOROFF
 C08C Q6L
 F000 REGRWTS
 F04E MOTOF
 F07B TRYTRK2
?F0A0 GOCAL1
?F0BF CORRECTVOL
 F0E8 HNDLERR
 F105 MYSEEK
 F11B NXOFF
 F13D CKDRTS
 F13E DRVINDX      F148 READ16        F14A RSYNC         F14D RD1
 F152 RSYNC1       F157 RD2           F162 RD3           F16B RD4
 F17D RD5          F17E RD5A          F192 RD6           F19D RDCKSUM
 F1A5 RD7          F1AF RD8           F1D8 RDERR         F1BA GOSERV
 F1BD RDADR16      F1C1 RDASYN        F1C8 RDA1          F1CD RDASN1
 F1D2 RDA2         F1DD RDA3          F1E8 RDAFLD        F1EA RDA4
 F1F2 RDA5         F204 RDA6          F20E RDA7          F217 RDEXIT
 F218 WEXIT        F219 WRITE16      ?F223 WRT1          F230 WSYNC
 F24E WINTRPT      F256 VRYFRST       F258 WRTFRST       F267 WNTRPT1
 F26A WMIDLE       F271 WDATA2        F281 WDATA3        F295 WRCKSUM
 F2AC NOWRITE      F2B3 SERVICE       F2BC WNIBL9        F2BD WNIBL7
?F2BF WNIBL        F2C6 PRENIB16      F2CA PRENIB1       F2E4 PRENIB3
?F2E7 PRENIB2      F2F8 PRENIB4       F300 DNIBL         F311 POSTNIB16
 F315 PNIBL1       F322 PNIBL2        F336 POST1         F338 POST2
 F34C POSTERR      F34C SET1MEG       F354 SEV           F355 NIBL
 F400 SEEK         F40A SEEK2         F41B OUT           F41F MINTST
 F425 MAXTST      ?F429 STEP          F42B STEP2         F444 SEEKEND
 F448 SETPHASE     F44A CLRPHASE     ?F455 SEEKRTS       F456 MSWAIT
 F458 MSW1         F461 MSW2          F467 ONTABLE       F470 OFFTABLE
?F479 BLOCKIO      F47E TRKSEC        F49D QUIT          F4A0 SECTABL
?F4A8 ANALOG       F4AB ANLOG1        F4C4 GOODTIME      FFD8 TIMER1L
 FFD9 TIMLATCH     FFD9 TIMER1H       FFDF ENVIRON       FFEF INTERUPT
 0000               2 ********************************
 0000               3 *
 0000:              4 *SARA DIAGNOSTIC TEST ROUTINES
 0000:              5 *
 0000:              6 *DECEMBER 19,1979
 0000               7 * BY
 0000:              8 *W.BROEDNER & R.LASHLEY
 0000:              9 *
 0000:             10 *COPYRIGHT 1979 BY APPLE COMPUTER, INC
 0000:             11 *
 0000:             12 ********************************
 0001:             13 ROM      EQU  $1          FOR RAM VERSION, 1 IF TRUELY ROM
 0000:             14 ZRPG     EQU  $0
 0010:             15 ZRPG1    EQU  $10
 0018:             16 PTRLO    EQU  ZRPG1+8
 0019:             17 PTRHI    EQU  ZRPG1+9
 001A:             18 BNK      EQU  ZRPG1+$A
 0087:             19 IBCMD    EQU  $87
 0085:             20 IBBUFP   EQU  $85
 0091:             21 PREVTRK  EQU  $91
 F479:             22 BLOCKIO  EQU  $F479
 005D:             23 CV       EQU  $5D
 00FF:             24 STKP     EQU  $FF
 1419:             25 IBNK     EQU  $1400+PTRHI
 1810:             26 PHP      EQU  $1800+ZRPG1
 C000:             27 KYPD     EQU  $C000
 C008:             28 KEYBD    EQU  $C008
 C010:             29 KBDSTRB  EQU  $C010
 C058:             30 PDLEN    EQU  $C058
 C047:             31 ADRS     EQU  $C047
 C050:             32 GRMD     EQU  $C050
 C051:             33 TXTMD    EQU  $C051
 C066:             34 ADTO     EQU  $C066
 C0D0:             35 DISKOFF  EQU  $C0D0
 C0F1:             36 ACIAST   EQU  $C0F1
 C0F2:             37 ACIACM   EQU  $C0F2
 C0F3:             38 ACIACN   EQU  $C0F3
 C100:             39 SLT1     EQU  $C100
 C200:             40 SLT2     EQU  $C200
 C300:             41 SLT3     EQU  $C300
 C400:             42 SLT4     EQU  $C400
 CFFF:             43 EXPROM   EQU  $CFFF
 FFD0:             44 ZPREG    EQU  $FFD0
 FFDF:             45 SYSD1    EQU  $FFDF
```

"APPLE_PAT_4_383_296_37" 144 KB 2000-02-27 dpi: 300h x 300v pix: 1837h x 2779v

```
FFD2:             46 SYSD2   EQU  $FFD2
FFD3:             47 SYSD3   EQU  $FFD3
FFE0:             48 SYSE0   EQU  $FFE0
FFEF:             49 BNKSW   EQU  $FFEF
FFE2:             50 SYSE2   EQU  $FFE2
FFE3:             51 SYSE3   EQU  $FFE3
FC25:             52 COUT    EQU  $FC25
FD07:             53 CROUT1  EQU  $FD07
FD0F:             54 KEYIN   EQU  $FD0F
FBC7:             55 SETCVH  EQU  $FBC7
FD98:             56 CLDSTRT EQU  $FD98
FD9D:             57 SETUP   EQU  $FD9D
F901:             58 MONITOR EQU  $F901
0000:             59 *
----- NEXT OBJECT FILE NAME IS DIAG.OBJ
F4C5:             60         ORG  $F4C5
F4C5:00 B1 B2     61 RAMTBL  DFB  $0,$B1,$B2,$BA,$B9,$10,$0,$13
F4C8:DA B9 10
F4CB:00 13
F4CD:             62 CHPG    EQU  *
F4CD:52 41 CD     63         DCI  'RAM'
F4D0:52 4F CD     64         DCI  'ROM'
F4D3:56 49 C1     65         DCI  'VIA'
F4D6:41 43 49     66         DCI  'ACIA'
F4D9:C1
F4DA:41 2F C4     67         DCI  'A/D'
F4DD:44 49 41     68         DCI  'DIAGNOSTIC'
F4E0:47 4E 4F
F4E3:53 54 49
F4E6:C3
F4E7:5A D0        69         DCI  'ZP'
F4E9:52 45 54     70         DCI  'RETRY'
F4EC:52 D9
F4EE:             71 *
F4EE:             72 * SETUP SYSTEM
F4EE:             73 *
F4EE:             74 *
F4EE:A9 53        75         LDA  #$52+ROM  TURN OFF SCREEN, SET 2MHZ SPEED
F4F0:8D DF FF     76         STA  SYSD1     AND RUN OFF ROM
F4F3:A2 00        77         LDX  #$00      SET BANK SWITCH TO ZERO
F4F5:8E E0 FF     78         STX  SYSE0
F4F8:8E EF FF     79         STX  BNKSW
F4FB:8E D0 FF     80         STX  ZPREG     AND SET ZERO PAGE SAME
F4FE:CA           81         DEX
F4FF:8E D2 FF     82         STX  SYSD2     PROGRAM DDR'S
F502:8E D3 FF     83         STX  SYSD3
F505:9A           84         TXS
F506:E8           85         INX
F507:A9 0F        86         LDA  #$0F
F509:8D E3 FF     87         STA  SYSE3
F50C:A9 3F        88         LDA  #$3F
F50E:8D E2 FF     89         STA  SYSE2
F511:A0 06        90         LDY  #$06
F513:B9 D0 C0     91 DISK1   LDA  DISKOFF,Y
F516:88           92         DEY
F517:88           93         DEY
F518:10 F9        94         BPL  DISK1
F51A:AD 08 C0     95         LDA  KEYBD
F51D:29 04        96         AND  #$04
F51F:D0 03        97         BNE  NXBYT
F521:4C 89 F6     98         JMP  RECON
F524:             99 *
F524:            100 * VERIFY ZERO PAGE
F524:            101 *
```

```
F524: A9 01      102 NXBYT   LDA  #$01       ROTATE A 1 THROUGH
F526: 95 00      103 NXBIT   STA  ZRPG,X     EACH BIT IN THE 0 PG
F528: D5 00      104         CMP  ZRPG,X     TO COMPLETELY TEST
F52A: D0 FE      105 NOGOOD  BNE  NOGOOD     THE PAGE. HANG IF NOGOOD
F52C: 0A         106         ASL  A          TRY NEXT BIT OF BYTE
F52D: D0 F7      107         BNE  NXBIT      UNTIL BYTE IS ZERO.
F52F: E8         108         INX              CONTINUE UNTIL PAGE
F530: D0 F2      109         BNE  NXBYT      IS DONE.
F532:            110 *
F532: 8A         111 CNTWR   TXA              PUSH A DIFFERENT
F533: 48         112         PHA              BYTE ONTO THE
F534: E8         113         INX              STACK UNTIL ALL
F535: D0 FB      114         BNE  CNTWR      STCK BYTES ARE FULL.
F537: CA         115         DEX              THEN PULL THEM
F538: 86 18      116         STX  PTRLO      OFF AND COMPARE TO
F53A: 68         117 PULBT   PLA              THE COUNTER GOING
F53B: C5 18      118         CMP  PTRLO      BACKWARDS. HANG IF
F53D: D0 EB      119         BNE  NOGOOD     THEY DON'T AGREE.
F53F: C6 18      120         DEC  PTRLO      GET NEXT COUNTER BYTE
F541: D0 F7      121         BNE  PULBT      CONTINUE UNTIL STACK
F543: 68         122         PLA              IS DONE. TEST LAST BYTE
F544: D0 E4      123         BNE  NOGOOD     AGAINST ZERO.
F546:            124 *
F546:            125 * SIZE THE MEMORY
F546:            126 *
F546: A2 08      127         LDX  #$08       ZERO THE BYTES USED TO DISPLAY
F548: 95 10      128 NOMEM   STA  ZRPG1,X    THE BAD RAM LOCATIONS
F54A: CA         129         DEX              EACH BYTE= A CAS LINE
F54B: 10 FB      130         BPL  NOMEM      ON THE SARA BOARD.
F54D:            131 *
F54D: A2 02      132         LDX  #$02       STARTING AT PAGE 2
F54F: 86 19      133 NMEM1   STX  PTRHI      TEST THE LAST BYTE
F551: A9 00      134         LDA  #$00       IN EACH MEM PAGE TO
F553: A0 FF      135         LDY  #$FF       SEE IF THE CHIPS ARE
F555: 91 18      136         STA  (PTRLO),Y  THERE. (AVOID 0 & STK PAGES)
F557: D1 18      137         CMP  (PTRLO),Y  CAN THE BYTE BE 0'D?
F559: F0 07      138         BEQ  NMEM2
F55B: 20 48 F7   139         JSR  RAM        NO, FIND WHICH CAS IT IS.
F55E: 94 10      140         STY  ZRPG1,X    SET CORRES. BYTE TO FF
F560: A6 19      141         LDX  PTRHI      RESTORE X REGISTER
F562: E8         142 NMEM2   INX              AND INCREMENT TO NEXT
F563: E0 C0      143         CPX  #$C0       PAGE UNTIL I/O IS REACHED.
F565: D0 E8      144         BNE  NMEM1
F567: A2 20      145         LDX  #$20       THEN RESET TO PAGE 20
F569: EE EF FF   146         INC  BNKSW      AND GOTO NEXT BANK TO
F56C: AD EF FF   147         LDA  BNKSW      CONTINUE. (MASK INPUTS
F56F: 29 0F      148         AND  #$0F       FROM BANKSWITCH TO SEE
F571: C9 03      149         CMP  #$03       WHAT SWITCH IS SET TO)
F573: D0 DA      150         BNE  NMEM1      CONTINUE UNTIL BANK '3'
F575:            151 *
F575:            152 * SETUP SCREEN
F575: 20 9D FD   153 ERRLP   JSR  SETUP      CALL SCRN SETUP ROUTINE
F578: A2 00      154         LDX  #$00       SETUP I/O AGAIN
F57A: 8E E0 FF   155         STX  SYSE0      FOR VIA TEST
F57D: CA         156         DEX              PROGRAM DATA DIR
F57E: 8E D2 FF   157         STX  SYSD2      REGISTERS
F581: 8E D3 FF   158         STX  SYSD3
F584: A9 3F      159         LDA  #$3F
F586: 8D E2 FF   160         STA  SYSE2
F589: A9 0F      161         LDA  #$0F
F58B: 8D E3 FF   162         STA  SYSE3
F58E: A2 10      163         LDX  #$10       HEADING OF 'DIAGNSTICS' WITH
F590: 20 38 F7   164         JSR  STRWT      THIS SUBROUTINE
F593: A2 00      165 ERRLP1  LDX  #$00       PRINT 'RAM'
F595: 86 5D      166         STX  CV         SET CURSOR TO 2ND LINE
F597: A9 04      167         LDA  #$04       SPACE CURSOR OUT 3
```



```
F599 20 C7 FB        168          JSR   SETCVH      (X STILL=0 ON RETURN)
F59C 20 38 F7        169          JSR   STRWT       THE SAME SUBROUTINE
F59F A2 07           170          LDX   #$07        FOR BYTES 7 - 0 IN
F5A1'                171 RAMWT1   EQU   *
F5A1: B5 10          172          LDA   ZRPG1,X     OUT EACH BIT AS A
F5A3: A0 08          173          LDY   #$08        ' ' OR '1' FOR INDICATE BAD OR MISSING
F5A5: 0A             174 RAMWT2   ASL   A           CHIPS  SUBROUTINE 'RAM'         RAM
F5A6 48              175          PHA                SETS UP THESE BYTES
F5A7 A9 AE           176          LDA   #$AE        LOAD A ' ' TO ACC
F5A9 90 02           177          BCC   RAMWT4
F5AB A9 31           178          LDA   #$31        LOAD A '1' TO ACC
F5AD 20 25 FC        179 RAMWT4   JSR   COUT        AND PRINT IT
F5B0 68              180          PLA                RESTORE BYTE
F5B1 88              181          DEY                AND ROTATE ALL 8
F5B2 D0 F1           182          BNE   RAMWT2      TIMES
F5B4 20 07 FD        183          JSR   CROUT1      CLEAR TO END OF LINE
F5B7 CA              184          DEX
F5B8 10 E7           185          BPL   RAMWT1
F5BA:                186 *
F5BA.                187 * ZPG&STK TEST
F5BA                 188 *
F5BA 9A              189          TXS
F5BB 8C EF FF        190          STY   BNKSW
F5BE 98              191 ZP1      TYA
F5BF 8D D0 FF        192          STA   ZPREG
F5C2 85 FF           193          STA   STK0
F5C4 C8              194          INY
F5C5 98              195          TYA
F5C6 48              196          PHA
F5C7 68              197          PLA
F5C8 C8              198          INY
F5C9 C0 20           199          CPY   #$20
F5CB D0 F1           200          BNE   ZP1
F5CD A0 00           201          LDY   #$00
F5CF 8C D0 FF        202          STY   ZPREG
F5D2 86 18           203          STX   PTRLO
F5D4 E8              204 ZP2      INX
F5D5 86 19           205          STX   PTRHI
F5D7 8A              206          TXA
F5D8 D1 18           207          CMP   (PTRLO),Y
F5DA D0 06           208          BNE   ZP3
F5DC E0 1F           209          CPX   #$1F
F5DE D0 F4           210          BNE   ZP2
F5E0 F0 05           211          BEQ   ROMTST
F5E2                 212 ZP3      EQU   *           CHIP IS THERE, BAD ZERO AND STACK
F5E2 A2 1A           213          LDX   #$1A        SO PRINT 'ZP' MESSAGE
F5E4 20 7B F7        214          JSR   MESSERR     & SET FLAG (2MHZ MODE)
F5E7                 215 *
F5E7                 216 * ROM TEST ROUTINE
F5E7                 217 *
F5E7 A9 00           218 ROMTST   LDA   #$00        SET POINTERS TO
F5E9 A8              219          TAY                $F000
F5EA A2 F0           220          LDX   #$F0
F5EC 85 18           221          STA   PTRLO
F5EE 86 19           222          STX   PTRHI       SET X TO $FF
F5F0 A2 FF           223          LDX   #$FF        FOR WINDOWING I/O
F5F2 51 18           224 ROMTST1  EOR   (PTRLO),Y   COMPUTE CHKSUM ON
F5F4 E4 19           225          CPX   PTRHI       EACH ROM BYTE,
F5F6: D0 06          226          BNE   ROMTST2     WINDOW OUT
F5F8: C0 BF          227          CPY   #$BF        RANGES FFC0-FFEF
F5FA: D0 02          228          BNE   ROMTST2
F5FC: A0 EF          229          LDY   #$EF
F5FE: C8             230 ROMTST2  INY
F5FF: D0 F1          231          BNE   ROMTST1
F601: E6 19          232          INC   PTRHI
F603: D0 ED          233          BNE   ROMTST1
F605: A8             234          TAY                TEST ACC. FOR 0
F606: F0 05          235          BEQ   VIATST      YES, NEXT TEST
F608: A2 03          236          LDX   #$03        PRINT 'ROM' AND
F60A: 20 7B F7       237          JSR   MESSERR     SET ERROR
F60D:                238 *
```

"APPLE_PAT_4_383_296_40" 138 KB 2000-02-27 dpi: 300h x 300v pix: 1855h x 2779v

```
F60D:          239 * VIA TEST ROUTINE
F60D:          240 *
F60D: 18       241 VIATST  CLC                 SET UP FOR ADDING BYTES
F60E: D8       242         CLD
F60F: AD E0 FF 243         LDA   SYSE0       MASK OFF INPUT BITS
F612: 29 3F    244         AND   #$3F        AND STORE BYTE IN
F614: 85 18    245         STA   PTRLO       TEMPOR. LOCATION
F616: AD EF FF 246         LDA   BNKSW       MASK OFF INPUT BITS
F619: 29 4F    247         AND   #$4F        AND ADD TO STORED
F61B: 65 18    248         ADC   PTRLO       BYTE IN TEMP. LOC.
F61D: 6D D0 FF 249         ADC   ZPREG       ADD REMAINING
F620: 85 18    250         STA   PTRLO       REGISTERS OF THE
F622: AD DF FF 251         LDA   SYSD1       VIA'S
F625  29 5F    252         AND   #$5F        (MASK THIS ONE)
F627: 65 18    253         ADC   PTRLO       AND TEST
F629  6D D2 FF 254         ADC   SYSD2       TO SEE
F62C: 6D D3 FF 255         ADC   SYSD3       IF THEY AGREE
F62F. 6D E2 FF 256         ADC   SYSE2       WITH THE RESET
F632  6D E3 FF 257         ADC   SYSE3       CONDITION.
F635  C9 E1    258         CMP   #$E0+ROM    =E1?
F637  F0 05    259         BEQ   ACIA        YES, NEXT TEST
F639  A2 06    260         LDX   #$06        NO, PRINT 'VIA' MESS.
F63B  20 7B F7 261         JSR   MESSERR     AND SET ERROR FLAG
F63E           262 *
F63E           263 * ACIA TEST ROUTINE
F63E           264 *
F63E  18       265 ACIA    CLC                 SETUP FOR ADDITION
F63F  A9 9F    266         LDA   #$9F        MASK INPUT BITS
F641: 2D F1 C0 267         AND   ACIAST      FROM STATUS REG
F644: 6D F2 C0 268         ADC   ACIACM      AND ADD DEFAULT STATES
F647: 6D F3 C0 269         ADC   ACIACN      OF CONTROL AND COMMND
F64A. C9 10    270         CMP   #$10        REGS. =10?
F64C: F0 05    271         BEQ   ATD         YES, NEXT TEST
F64E: A2 09    272         LDX   #$09        NO, 'ACIA' MESSAGE AND
F650: 20 7B F7 273         JSR   MESSERR     THEN SET ERROR FLAG
F653:          274 *
F653:          275 * A/D TEST ROUTINE
F653:          276 *
F653  A9 C0    277 ATD     LDA   #$C0
F655. 8D DC FF 278         STA   $FFDC
F658. AD 5A C0 279         LDA   PDLEN+2
F65B: AD 5E C0 280         LDA   PDLEN+6
F65E. AD 5C C0 281         LDA   PDLEN+4
F661. A0 20    282         LDY   #$20
F663: 88       283 ADCTST1 DEY                 WAIT FOR 40 USEC
F664. D0 FD    284         BNE   ADCTST1
F666: AD 5D C0 285         LDA   PDLEN+5     SET A/D RAMP
F669  C8       286 ADCTST3 INY                 COUNT FOR CONVERSION
F66A  F0 0A    287         BEQ   ADCERR      (>255=ERROR)
F66C  AD 66 C0 288         LDA   ADTO        IF BIT 7 =1?
F66F' 30 F8    289         BMI   ADCTST3     YES, CONTINUE
F671  98       290         TYA                 NO, MOVE COUNT TO ACC
F672: 29 E0    291         AND   #$E0          ACC<32?
F674: F0 05    292         BEQ   KEYPLUG
F676           293 ADCERR  EQU   *           NO,
F676  A2 0D    294         LDX   #$0D        PRINT 'A/D' MESS
F678  20 7B F7 295         JSR   MESSERR     AND SET ERROR FLAG
F67B:          296 *
F67B:          297 * KEYBOAD PLUGIN TEST
F67B.          298 *
F67B  AD 08 C0 299 KEYPLUG LDA   KEYBD       IS KYBD PLUGGED IN?
F67E. 0A       300         ASL   A           (IS LIGHT CURRENT
F67F 10 41      301             BPL  SEX         PRESENT?) NO, BRANCH
F681 AD DF FF   302             LDA  SYSD1       IS ERROR FLAG SET?
F684:10 03      303             BPL  RECON       (2MHZ MODE) NO, BRANCH
F686 4C 93 F3   304             JMP  ERRLP1      ERROR, HANG
F689:           305 *
F689:           306 * RECONFIGURE SYSTEM
F689:           307 *
F689:           308 RECON       EQU  *
F689:A9 77      309             LDA  #$77        TURN ON SCREEN
F68B 8D DF FF   310             STA  SYSD1
F68E 20 98 FD   311             JSR  CLDSTRT     INITIALIZE MONITOR AND DEFAULT CHARACTER
F691 A9 10      312             LDA  #$10        TEST FOR "APPLE 1"                 SET
F693:2D 08 C0   313             AND  KEYBD
F696:D0 09      314             BNE  BOOT        NO, DO REGULAR BOOT
F698:2C 10 C0   315             BIT  KBDSTRB     CLEAR KEYBOARD
F69B AD 50 C0   316             LDA  GRMD
F69E 20 01 F9   317             JSR  MONITOR     AND NEVER COME BACK.
F6A1 A2 01      318 BOOT        LDX  #1          READ BLOCK 0
F6A3 86 87      319             STX  IBCMD
F6A5:CA         320             DEX
F6A6 86 85      321             STX  IBBUFP      INTO RAM AT $A000
F6A8 A9 A0      322             LDA  #$A0
F6AA 85 86      323             STA  IBBUFP+1
F6AC 4A         324             LSR  A           ,FOR TRACK 80
F6AD 85 91      325             STA  PREVTRK     MAKE IT RECALIBRATE TOO!
F6AF 8A         326             TXA
F6B0 20 79 F4   327             JSR  BLOCKIO
F6B3 90 0A      328             BCC  GOBOOT      IF WE'VE SUCCEEDED, DO IT UP
F6B5 A2 1C      329             LDX  #$1C
F6B7 20 38 F7   330             JSR  STRWT       'RETRY?'
F6BA 20 0F FD   331             JSR  KEYIN
F6BD B0 E2      332             BCS  BOOT
F6BF 4C 00 A0   333 GOBOOT      JMP  $A000       ,GO TO IT FOOL...
F6C2            334 *
F6C2            335 * SYSTEM EXERCISER
F6C2            336 *
F6C2 A0 7F      337 SEX         LDY  #$7F        TRYFROM
F6C4 98         338 SEX1        TYA               7F TO 0
F6C5 29 FE      339             AND  #$FE        ADD. =
F6C7 49 4E      340             EOR  #$4E        4EOR4F?
F6C9 F0 03      341             BEQ  SEX2        YES, SKP
F6CB:B9 00 C0   342             LDA  KYBD,Y      NO, CONT
F6CE:88         343 SEX2        DEY               NXT ADD
F6CF:D0 F3      344             BNE  SEX1
F6D1:AD 51 C0   345             LDA  TXTMD       SET TXT
F6D4:B9 00 C1   346 SEX3        LDA  SLT1,Y      EXERCSE
F6D7:B9 00 C2   347             LDA  SLT2,Y      ALL
F6DA:B9 00 C3   348             LDA  SLT3,Y      SLOTS
F6DD:B9 00 C4   349             LDA  SLT4,Y
F6E0:AD FF CF   350             LDA  EXPROM      DISABLE EXPANSION ROM AREA
F6E3:C8         351             INY
F6E4:D0 EE      352             BNE  SEX3
F6E6:           353 *
F6E6:           354 * RAM TEST ROUTINE
F6E6:           355 *
F6E6:A9 73      356 USRENTRY LDA #$72+ROM
F6E8:8D DF FF   357             STA  SYSD1
F6EB:A9 18      358             LDA  #$18
F6ED:8D D0 FF   359             STA  ZPREG
F6F0:A9 00      360             LDA  #$00
F6F2:A2 07      361             LDX  #$07
F6F4:95 10      362 RAMTSTO     STA  ZRPG1,X
F6F6:CA         363             DEX
F6F7:10 FB      364             BPL  RAMTSTO
F6F9:20 84 F7   365             JSR  RAMSET
F6FC:08         366             PHP
F6FD:20 F7 F7   367 RAMTST1     JSR  RAMWT
F700:20 F7 F7   368             JSR  RAMWT
```

“APPLE_PAT_4_383_296_42” 150 KB 2000-02-27 dpi: 300h x 300v pix: 1836h x 2755v

```
F703: 28                       PLP
F704: 6A                       ROR   A
F705: 08                       PHP
F706: 20 A1 F7                 JSR   PTRINC
F709: D0 F2                    BNE   RAMTST1
F70B: 20 84 F7                 JSR   RAMSET
F70E: 08                       PHP
F70F: 20 FB F7      RAMTST4    JSR   RAMRD
F712: 48                       PHA
F713: A9 00                    LDA   #$00
F715: 91 18                    STA   (PTRLO),Y
F717: 68                       PLA
F718: 28                       PLP
F719: 6A                       ROR   A
F71A: 08                       PHP
F71B: 20 A1 F7                 JSR   PTRINC
F71E: D0 EF                    BNE   RAMTST4
F720                *
F720                * RETURN TO START
F720                *
F720  A9 00                    LDA   #$00
F722  8D EF FF                 STA   BNKSW
F725  8D D0 FF                 STA   ZPREG
F728  A2 07                    LDX   #$07
F72A  BD 10 18      RAMTST6    LDA   PHP,X
F72D  95 10                    STA   ZRPG1,X
F72F  CA                       DEX
F730  10 F8                    BPL   RAMTST6
F732: 20 7E F7                 JSR   ERROR
F735  4C 75 F5                 JMP   ERRLP
F738                ********************************
F738                * SARA TEST SUBROUTINES
F738                ********************************
F738                *
F738                * SUBROUTINE STRING WRITE
F738                *
F738  BD CD F4      STRWT      LDA   CHPG,X
F73B  48                       PHA
F73C  09 80                    ORA   #$80        NORMAL VIDEO
F73E  20 25 FC                 JSR   COUT        & PRNT
F741  E8                       INX                NXT
F742  68                       PLA                CHR
F743  10 F3                    BPL   STRWT
F745  4C 07 FD                 JMP   CROUT1      CLR TO END OF LINE
F748                *
F748                * SUBROUTINE RAM
F748                *
F748: 48            RAM        PHA                SV ACC
F749: 8A                       TXA                CONVRT
F74A  4A                       LSR   A           ADD TO
F74B: 4A                       LSR   A           USE FOR
F74C  4A                       LSR   A           8 ENTRY
F74D  4A                       LSR   A
F74E  08                       PHP
F74F  4A                       LSR   A
F750  28                       PLP
F751: AA                       TAX                LOOKUP
F752  BD C5 F4                 LDA   RAMTBL,X    IF VAL
```

```
F755 10 14    427          BPL   RAM0       NO,GET
F757 48       428          PHA              WHICH
F758 AD EF FF 429          LDA   BNKSW
F75B 29 0F    430          AND   #$0F
F75D AA       431          TAX
F75E 68       432          PLA
F75F E0 00    433          CPX   #$00
F761 F0 13    434          BEQ   RAM1       BANK0
F763 4A       435          LSR   A          SET
F764 4A       436          LSR   A          PROPER
F765 4A       437          LSR   A          RAM
F766 CA       438          DEX               VALUE
F767 D0 0B    439          BNE   RAM1
F769 29 05    440          AND   #$05       CONVERT
F76B D0 09    441 RAM0     BNE   RAM1       TO VAL
F76D 8A       442          TXA
F76E F0 02    443          BEQ   RAM00
F770 A9 03    444          LDA   #3
F772 90 02    445 RAM00    BCC   RAM1
F774 49 03    446          EOR   #3
F776 29 07    447 RAM1     AND   #$07       BANKSW
F778 AA       448          TAX
F779 68       449          PLA
F77A 60       450          RTS
F77B          451 *
F77B          452 * SUBROUTINE ERROR
F77B          453 *
F77B 20 38 F7 454 MESSERR  JSR   STRWT      PRINT MESSAGE FIRST
F77E A9 F3    455 ERROR    LDA   #$F2+ROM   SET 1
F780 8D DF FF 456          STA   SYSD1      MHZ MO
F783 60       457          RTS
F784          458 *
F784          459 * SUBROUTINE RAMSET
F784          460 *
F784 A2 01    461 RAMSET   LDX   #$01
F786 86 1A    462          STX   BNK
F788 A0 00    463          LDY   #$00
F78A A9 AA    464          LDA   #$AA
F78C 38       465          SEC
F78D 48       466 RAMSET1  PHA
F78E 08       467          PHP
F78F A5 1A    468          LDA   BNK
F791 09 80    469          ORA   #$80
F793 8D 19 14 470          STA   IBNK
F796 A9 02    471          LDA   #$02
F798 85 19    472          STA   PTRHI
F79A A2 00    473          LDX   #$00
F79C 86 18    474          STX   PTRLO
F79E 28       475          PLP
F79F 68       476          PLA
F7A0 60       477          RTS
F7A1          478 *
F7A1          479 * SUBROUTINE PTRINC
F7A1          480 *
F7A1 48       481 PTRINC   PHA
```

```
F7A2 E6 18      482          INC   PTRLO
F7A4 D0 1D      483          BNE   RETS
F7A6 A5 1A      484          LDA   BNK
F7A8 10 0E      485          BPL   PINC1
F7AA A5 19      486          LDA   PTRHI
F7AC C9 13      487          CMP   #$13
F7AE F0 06      488          BEQ   PINC2
F7B0 C9 17      489          CMP   #$17
F7B2 D0 04      490          BNE   PINC1
F7B4 E6 19      491          INC   PTRHI
F7B6 E6 19      492 PINC2    INC   PTRHI
F7B8 E6 19      493 PINC1    INC   PTRHI
F7BA D0 07      494          BNE   RETS
F7BC C6 1A      495          DEC   BNK
F7BE A4 1A      496          LDY   BNK
F7C0 20 9D F7   497          JSR   RAMSET1
F7C3 68         498 RETS     PLA
F7C4 A6 1A      499          LDX   BNK
F7C6 E0 FD      500          CPX   #$FD
F7C8 60         501          RTS
F7C9            502 *
F7C9            503 * SUBROUTINE RAMERR
F7C9            504 *
F7C9 48         505 RAMERR   PHA
F7CA A5 19      506          LDA   PTRHI
F7CC A4 1A      507          LDY   BNK
F7CE 0A         508          ASL
F7CF 30 19      509          BMI   RAMERR4
F7D1 0A         510          ASL
F7D2 90 1D      511          BCC   RAMERR5
F7D4 18         512          CLC
F7D5 69 20      513          ADC   #$20
F7D7 8C EF FF   514 RAMERR2  STY   BNKSW
F7DA AA         515          TAX
F7DB 20 48 F7   516 RAMERR3  JSR   RAM
F7DE 68         517          PLA
F7DF 48         518          PHA
F7E0 A0 00      519          LDY   #$00
F7E2 51 18      520          EOR   (PTRLO),Y
F7E4 15 10      521          ORA   ZRPG1,X
F7E6 95 10      522          STA   ZRPG1,X
F7E8 68         523          PLA
F7E9 60         524          RTS
F7EA A9 00      525 RAMERR4  LDA   #$00
F7EC 8D EF FF   526          STA   BNKSW
F7EF F0 EA      527          BEQ   RAMERR3
F7F1 38         528 RAMERR5  SEC
F7F2 E9 60      529          SBC   #$60
F7F4 C8         530          INY
F7F5 D0 E0      531          BNE   RAMERR2
F7F7            532 *
F7F7            533 * SUBROUTINE RAMWT
F7F7            534 *
F7F7 49 FF      535 RAMWT    EOR   #$FF
F7F9 91 18      536          STA   (PTRLO),Y
```

```
F7FB D1 18       537 RAMRD     CMP   (PTRLO),Y
F7FD D0 CA       538           BNE   RAMERR
F7FF 60          539           RTS

*** SUCCESSFUL ASSEMBLY  NO ERRORS
C0F3 ACIACN       F63E ACIA         C0F2 ACIACM       C0F1 ACIAST
F676 ADCERR       F663 ADCTST1      F669 ADCTST3     ?C047 ADRS
C066 ADTO         F653 ATD          F479 BLOCKIO        1A BNK
FFEF BNKSW        F6A1 BOOT         F4CD CHPG         FD98 CLDSTRT
F532 CNTWR        FC25 COUT         FD07 CROUT1         5D CV
F513 DISK1        C0D0 DISKOFF      F575 ERRLP        F593 ERRLP1
F77E ERROR        CFFF EXPROM       F6BF GOBOOT       C050 GRMD
  85 IBBUFP         87 IBCMD        1419 IBNK         C010 KBDSTRB
C008 KEYBD        FD0F KEYIN        F67B KEYPLUG      C000 KYBD
F77B MESSERR      F901 MONITOR      F54F NMEM1        F562 NMEM2
F52A NOGOOD       F548 NOMEM        F526 NXBIT        F524 NXBYT
C058 PDLEN        1810 PHP          F7B8 PINC1        F7B6 PINC2
  91 PREVTRK        19 PTRHI        F7A1 PTRINC         18 PTRLO
F53A PULBT        F772 RAM00        F776 RAM1         F748 RAM
F76B RAM0         F7DB RAMERR3      F7EA RAMERR4      F7C9 RAMERR
F7D7 RAMERR2      F7F1 RAMERR5      F7FB RAMRD        F784 RAMSET
F78D RAMSET1      F4C5 RAMTBL       F6F4 RAMTST0      F6FD RAMTST1
F70F RAMTST4      F72A RAMTST6      F5A1 RAMWT1       F5AD RAMWT4
F7F7 RAMWT        F5A5 RAMWT2       F689 RECON        F7C3 RETS
F5F2 ROMTST1      F5FE ROMTST2      F5E7 ROMTST         01 ROM
FBC7 SETCVH       FD9D SETUP        F6C4 SEX1         F6C2 SEX
F6CE SEX2         F6D4 SEX3         C100 SLT1         C200 SLT2
C300 SLT3         C400 SLT4           FF STK0         F738 STRWT
FFDF SYSD1        FFD2 SYSD2        FFD3 SYSD3        FFE0 SYSE0
FFE2 SYSE2        FFE3 SYSE3        C051 TXTMD       ?F6E6 USRENTRY
F60D VIATST       F5BE ZP1          F5D4 ZP2          F5E2 ZP3
FFD0 ZPREG          10 ZRPG1          00 ZRPG
  00 ZRPG           01 ROM            10 ZRPG1          18 PTRLO
  19 PTRHI          1A BNK            5D CV             85 IBBUFP
  87 IBCMD          91 PREVTRK        FF STK0         1419 IBNK
1810 PHP          C000 KYBD         C008 KEYBD        C010 KBDSTRB
?C047 ADRS        C050 GRMD         C051 TXTMD        C058 PDLEN
C066 ADTO         C0D0 DISKOFF      C0F1 ACIAST       C0F2 ACIACM
C0F3 ACIACN       C100 SLT1         C200 SLT2         C300 SLT3
C400 SLT4         CFFF EXPROM       F479 BLOCKIO      F4C5 RAMTBL
F4CD CHPG         F513 DISK1        F524 NXBYT        F526 NXBIT
F52A NOGOOD       F532 CNTWR        F53A PULBT        F548 NOMEM
F54F NMEM1        F562 NMEM2        F575 ERRLP        F593 ERRLP1
F5A1 RAMWT1       F5A5 RAMWT2       F5AD RAMWT4       F5BE ZP1
F5D4 ZP2          F5E2 ZP3          F5E7 ROMTST       F5F2 ROMTST1
F5FE ROMTST2      F60D VIATST       F63E ACIA         F653 ATD
F663 ADCTST1      F669 ADCTST3      F676 ADCERR       F67B KEYPLUG
F689 RECON        F6A1 BOOT         F6BF GOBOOT       F6C2 SEX
F6C4 SEX1         F6CE SEX2         F6D4 SEX3        ?F6E6 USRENTRY
F6F4 RAMTST0      F6FD RAMTST1      F70F RAMTST4      F72A RAMTST6
F738 STRWT        F748 RAM          F76B RAM0         F772 RAM00
F776 RAM1         F77B MESSERR      F77E ERROR        F784 RAMSET
F78D RAMSET1      F7A1 PTRINC       F7B6 PINC2        F7B8 PINC1
F7C3 RETS         F7C9 RAMERR       F7D7 RAMERR2      F7DB RAMERR3
F7EA RAMERR4      F7F1 RAMERR5      F7F7 RAMWT        F7FB RAMRD
F901 MONITOR      FBC7 SETCVH       FC25 COUT         FD07 CROUT1
FD0F KEYIN        FD98 CLDSTRT      FD9D SETUP        FFD0 ZPREG
FFD2 SYSD2        FFD3 SYSD3        FFDF SYSD1        FFE0 SYSE0
FFE2 SYSE2        FFE3 SYSE3        FFEF BNKSW
------ NEXT OBJECT FILE NAME IS MON.OBJ
F7FF              2            ORG   $F7FF
F7FF              3 *
```

```
F7FF             *
F7FF 60          RET1     RTS
F800 E9 01                SBC  #1
F802 F0 FB                BEQ  RET1
F804 E9 01                SBC  #1
F806 F0 F7                BEQ  RET1
F808 E9 01                SBC  #1
F80A F0 F3                BEQ  RET1
F80C E9 01                SBC  #1
F80E F0 EF                BEQ  RET1
F810 E9 01                SBC  #1
F812 F0 EB                BEQ  RET1
F814 E9 01                SBC  #1
F816 F0 E7                BEQ  RET1
F818 E9 01                SBC  #1
F81A F0 E3                BEQ  RET1
F81C E9 01                SBC  #1
F81E F0 DF                BEQ  RET1
F820 E9 01                SBC  #1
F822 F0 DB                BEQ  RET1
F824 E9 01                SBC  #1
F826 F0 D7                BEQ  RET1
F828 E9 01                SBC  #1
F82A F0 D3                BEQ  RET1
F82C E9 01                SBC  #1
F82E F0 CF                BEQ  RET1
F830 E9 01                SBC  #1
F832 F0 CB                BEQ  RET1
F834 E9 01                SBC  #1
F836 F0 C7                BEQ  RET1
F838 E9 01                SBC  #1
F83A F0 C3                BEQ  RET1
F83C E9 01                SBC  #1
F83E F0 BF                BEQ  RET1
F840 E9 01                SBC  #1
F842 F0 BB                BEQ  RET1
F844 E9 01                SBC  #1
F846 F0 B7                BEQ  RET1
F848 E9 01                SBC  #1
F84A F0 B3                BEQ  RET1
F84C E9 01                SBC  #1
F84E F0 AF                BEQ  RET1
F850 E9 01                SBC  #1
F852 F0 AB                BEQ  RET1
F854 E9 01                SBC  #1
F856 F0 A7                BEQ  RET1
F858 E9 01                SBC  #1
F85A F0 A3                BEQ  RET1
F85C E9 01                SBC  #1
F85E F0 9F                BEQ  RET1
F860 E9 01                SBC  #1
F862 F0 9B                BEQ  RET1
F864 E9 01                SBC  #1
F866 F0 97                BEQ  RET1
F868 E9 01                SBC  #1
```

“APPLE_PAT_4_383_296_47” 83 KB 2000-02-27 dpi: 300h x 300v pix: 1441h x 2701v

```
F86A F0 93        59          BEQ  RET1
F86C E9 01        60          SBC  #1
F86E F0 8F        61          BEQ  RET1
F870 E9 01        62          SBC  #1
F872 F0 8B        63          BEQ  RET1
F874 E9 01        64          SBC  #1
F876 F0 87        65          BEQ  RET1
F878 E9 01        66          SBC  #1
F87A F0 83        67          BEQ  RET1
F87C E9 01        68          SBC  #1
F87E F0 02        69          BEQ  RET3
F880 E9 01        70          SBC  #1
F882 F0 7C        71  RET3    BEQ  RET2
F884 E9 01        72          SBC  #1
F886 F0 78        73          BEQ  RET2
F888 E9 01        74          SBC  #1
F88A F0 74        75          BEQ  RET2
F88C E9 01        76          SBC  #1
F88E F0 70        77          BEQ  RET2
F890 E9 01        78          SBC  #1
F892 F0 6C        79          BEQ  RET2
F894 E9 01        80          SBC  #1
F896 F0 68        81          BEQ  RET2
F898 E9 01        82          SBC  #1
F89A F0 64        83          BEQ  RET2
F89C E9 01        84          SBC  #1
F89E F0 60        85          BEQ  RET2
F8A0 E9 01        86          SBC  #1
F8A2 F0 5C        87          BEQ  RET2
F8A4 E9 01        88          SBC  #1
F8A6 F0 58        89          BEQ  RET2
F8A8 E9 01        90          SBC  #1
F8AA F0 54        91          BEQ  RET2
F8AC E9 01        92          SBC  #1
F8AE F0 50        93          BEQ  RET2
F8B0 E9 01        94          SBC  #1
F8B2 F0 4C        95          BEQ  RET2
F8B4 E9 01        96          SBC  #1
F8B6 F0 48        97          BEQ  RET2
F8B8 E9 01        98          SBC  #1
F8BA F0 44        99          BEQ  RET2
F8BC E9 01       100          SBC  #1
F8BE F0 40       101          BEQ  RET2
F8C0 E9 01       102          SBC  #1
F8C2 F0 3C       103          BEQ  RET2
F8C4 E9 01       104          SBC  #1
F8C6 F0 38       105          BEQ  RET2
F8C8 E9 01       106          SBC  #1
F8CA F0 34       107          BEQ  RET2
F8CC E9 01       108          SBC  #1
F8CE F0 30       109          BEQ  RET2
F8D0 E9 01       110          SBC  #1
F8D2 F0 2C       111          BEQ  RET2
F8D4 E9 01       112          SBC  #1
```

"APPLE_PAT_4_383_296_48" 102 KB 2000-02-27 dpi: 300h x 300v pix: 1428h x 2714v

```
FBD6: F0 28             BEQ   RET2
FBD8: E9 01             SBC   #1
FBDA: F0 24             BEQ   RET2
FBDC: E9 01             SBC   #1
FBDE: F0 20             BEQ   RET2
FBE0: E9 01             SBC   #1
FBE2: F0 1C             BEQ   RET2
[illegible]             SBC   #1
[illegible] F0 18       BEQ   RET2
[illegible] E9 01       SBC   #1
FBEA: F0 14             BEQ   RET2
[illegible] E9 01       SBC   #1
[illegible]             BEQ   RET2
[illegible]             SBC   #
[illegible]             BEQ   RET2
FBF4: E9 01             SBC   #1
FBF6: F0 08             BEQ   RET2
FBF8: E9 01             SBC   #1
[illegible]             BEQ   RET2
[illegible] 01          SBC   #1
[illegible]             BEQ   RET2
[illegible] 60    RET2  RTS
F901:                   JMP   MON9A
F901:             *
F901:             *
0058              SCRNLOC EQU  $58
F901:             *
0058:             LMARGIN EQU  SCRNLOC
0059:             RMARGIN EQU  SCRNLOC+1
005A:             WINTOP  EQU  SCRNLOC+2
005B:             WINBTM  EQU  SCRNLOC+3
005C:             CH      EQU  SCRNLOC+4
005D:             CV      EQU  SCRNLOC+5
005E:             BAS4L   EQU  SCRNLOC+6
005F:             BAS4H   EQU  SCRNLOC+7
0060:             BAS8L   EQU  SCRNLOC+8
0061:             BAS8H   EQU  SCRNLOC+9
0062:             TBAS4L  EQU  SCRNLOC+$A
0063:             TBAS4H  EQU  SCRNLOC+$B
0064:             TBAS8L  EQU  SCRNLOC+$C
0065:             TBAS8H  EQU  SCRNLOC+$D
0066:             FORGND  EQU  SCRNLOC+$E
0067:             BKGND   EQU  SCRNLOC+$F
0068:             MODES   EQU  SCRNLOC+$10
0069:             CURSOR  EQU  SCRNLOC+$11
006A:             STACK   EQU  SCRNLOC+$12
006B:             PROMPT  EQU  SCRNLOC+$13
006C:             TEMPX   EQU  SCRNLOC+$14
006D:             TEMPY   EQU  SCRNLOC+$15
006E:             CSWL    EQU  SCRNLOC+$16
006F:             CSWH    EQU  SCRNLOC+$17
0070:             KSWL    EQU  SCRNLOC+$18
0071:             KSWH    EQU  SCPNLOC+$19
0072:             PCL     EQU  SCRNLOC+$1A
0073:             PCH     EQU  SCRNLOC+$1B
0074:             A1L     EQU  SCRNLOC+$1C
0075:             A1H     EQU  A1L+1
0076:             A2L     EQU  A1L+2
0077:             A2H     EQU  A1L+3
0078:             A3L     EQU  A1L+4
0079:             A3H     EQU  A1L+5
```

```
007A:             40 A4L      EQU  A1L+6
007B:             41 A4H      EQU  A1L+7
007C:             42 STATE    EQU  A1L+8
007D:             43 YSAV     EQU  A1L+9
007E:             44 INBUF    EQU  A1L+$A     ;AND $B
0080:             45 TEMP     EQU  A1L+$C
0069:             46 MASK     EQU  CURSOR
F901:             47 *
C000:             48 KBD      EQU  $C000
C010:             49 KBDSTRB EQU  $C010
F901:             50 *
03F8:             51 USERADR EQU  $3F8
F479:             52 BLOCKIO EQU  $F479
F689:             53 RECON    EQU  $F689      AS OF 12/20/79
F4EE:             54 DIAGN    EQU  $F4EE
0050:             55 INBUFLEN EQU $50          ;ONLY 80 BYTES ($3A0-3EF)
0081:             56 IBSLOT   EQU  $81
0082:             57 IBDRVN   EQU  IBSLOT+1
0085:             58 IBBUFP   EQU  IBSLOT+4
0087:             59 IBCMD    EQU  IBSLOT+6
F901             60 *
F901.            61 ENTRY   EQU  *
F901:BA          62         TSX
F902 86 6A       63         STX  STACK
F904             64 *
F904:D8          65 MON     CLD             ;MUST BE HEX MODE
F905:20 3A FC    66         JSR  BELL
F908 A6 6A       67 MONZ    LDX  STACK      ;RESTORE STACK TO ORIGINAL LOCATION
F90A 9A          68         TXS
F90B.A9 DF       69         LDA  #$DF        ;PROMPT (APPLE) FOR SARA MONITOR
F90D 85 6B       70         STA  PROMPT
F90F:20 D5 FC    71         JSR  GETLNZ     ,GET A LINE OF INPUT
F912:20 67 F9    72 SCAN    JSR  ZSTATE    ;SET REGULAR SCAN
F915:20 2C F9    73 NXTINP  JSR  GETNUM     ;ATTEMPT TO READ HEX BYTE
F918 84 7D       74         STY  YSAV       ;STORE CURRENT INPUT POINTER
F91A A0 11       75         LDY  #$11      ;17 COMMANDS
F91C 88          76 CMDSRCH DEY
F91D.30 E5       77         BMI  MON        ;GIVE UP IF UNRECOGNIZABLE
F91F:D9 6C F9    78         CMP  CMDTAB,Y   ;FOUND?
F922.D0 F8       79         BNE  CMDSRCH    ;NO KEEP LOOKING
F924:20 5E F9    80         JSR  TOSUB      ,PERFORM FUNCTION
F927 A4 7D       81         LDY  YSAV       ,GET NEXT POINTER
F929 4C 15 F9    82         JMP  NXTINP     ;DO NEXT COMMAND
F92C:            83 *
F92C:A2 00       84 GETNUM  LDX  #0         ;CLEAR A2
F92E.86 76       85         STX  A2L
F930 86 77       86         STX  A2H
F932.B1 7E       87 NXTCHR  LDA  (INBUF),Y
F934 C8          88         INY            ;BUMP INDEX FOR NEXT TIME
F935 49 B0       89         EOR  #$B0
F937.C9 0A       90         CMP  #$A        ;TEST FOR DIGIT
F939:90 06       91         BCC  DIGIT      ;SAVE IT IF 1-9
F93B.69 88       92         ADC  #$88       ;TEST FOR HEX A-F
F93D.C9 FA       93         CMP  #$FA
F93F 90 2A       94         BCC  DIGRET
F941:A2 03       95 DIGIT   LDX  #3
F943:0A          96         ASL  A
F944:0A          97         ASL  A
F945 0A          98         ASL  A
F946 0A          99         ASL  A
F947:0A         100 NXTBIT  ASL  A          ;SHIFT HEX DIGITS INTO A2
F948:26 76      101         ROL  A2L
F94A:26 77      102         ROL  A2H
F94C:CA         103         DEX             ;SHIFTED ALL YET?
F94D:10 F8      104         BPL  NXTBIT
F94F A5 7C      105 NXTBAS  LDA  STATE
F951:D0 06      106         BNE  NXTBS2     ;IF ZERO THEN COPY TO A1,3
```

```
F953:B5 77               LDA   A2H,X
F955:95 75               STA   A1H,X
F957:95 79               STA   A3H,X
F959 E8          NXTBS2  INX
F95A:F0 F3               BEQ   NXTBAS
F95C D0 D4               BNE   NXTCHR
F95E:            *
F95E:A9 FA       TOSUB   LDA   #<ASCII    ;PUSH ADDRESS OR FUNCTION
F960:48                  PHA               ;AND RETURN TO IT.
F961 B9 7C F9            LDA   CMDVEC,Y
F964 48                  PHA
F965:A5 7C               LDA   STATE      ;PASS MODE VIA ACC.
F967:A0 00       ZSTATE  LDY   #0
F969:84 7C               STY   STATE      ;RESET STATE OF SCAN
F96B:60          DIGRET  RTS
F96C:            *
F96C:            CMDTAB  EQU   *
F96C:00                  DFB   $0         ; G   =GO (CALL) SUBROUTINE
F96D:03                  DFB   $3         ; J   =JUMP (CONT) PROGRAM
F96E:06                  DFB   $6         ; M   =MOVE MEMORY
F96F:EB                  DFB   $EB        ; R   =READ DISK BLOCK
F970:EE                  DFB   $EE        ; U   =USER FUNCTION
F971:EF                  DFB   $EF        ; V   =VERIFY MEMORY BLOCKS
F972:F0                  DFB   $F0        ; W   =WRITE DISK BLOCK
F973:F1                  DFB   $F1        ; X   =REPEAT LINE OF COMMANDS
F974:99                  DFB   $99        ; SP  =SPACE (BYTE SEPARATOR)
F975:9B                  DFB   $9B        ; "   =ASCII (HI BIT ON)
F976:A0                  DFB   $A0        ; '   =ASCII (HI BIT OFF)
F977:93                  DFB   $93        ; :   =SET STORE MODE
F978:A7                  DFB   $A7        ; .   =RANGE SEPARATOR
F979:A8                  DFB   $A8        ; /   =COMMAND SEPARATOR
F97A:95                  DFB   $95        ; <   =DEST/SOURCE SEPARATOR
F97B:C6                  DFB   $C6        ; CR  =CARRAGE RETURN
F97C:            *
F97C:            CMDVEC  EQU   *
F97C:7C                  DFB   GO-1
F97D:7A                  DFB   JUMP-1
F97E:2B                  DFB   MOVE-1
F97F:DF                  DFB   READ-1
F980:77                  DFB   USER-1
F981:3A                  DFB   VRFY-1
F982:C2                  DFB   WRTE-1
F983:18                  DFB   REPEAT-1
F984:A3                  DFB   SPCE-1
F985:06                  DFB   ASCII-1
F986:06                  DFB   ASC1IO-1
F987 B7                  DFB   SETMODE-1
F988:B7                  DFB   SETMODE-1
F989:99                  DFB   SEP-1
F98A:90                  DFB   DEST-1
F98B:25                  DFB   CRMON-1
F98C:            *
F98C:            *
F98C:E6 7A       NXTA4   INC   A4L        ;BUMP 16 BIT POINTERS
F98E:D0 02               BNE   NXTA1
F990:E6 7B               INC   A4H
F992 E6 74       NXTA1   INC   A1L        ;BUMP A1
F994:D0 05               BNE   TSTA1
F996:E6 75               INC   A1H
F998:38                  SEC              ;IN CASE OF ROLL OVER.
F999:F0 10               BEQ   RETA1
F99B:A5 74       TSTA1   LDA   A1L        ;TEST A1>A2
F99D:38                  SEC
F99E:E5 76               SBC   A2L
F9A0:85 80               STA   TEMP
```

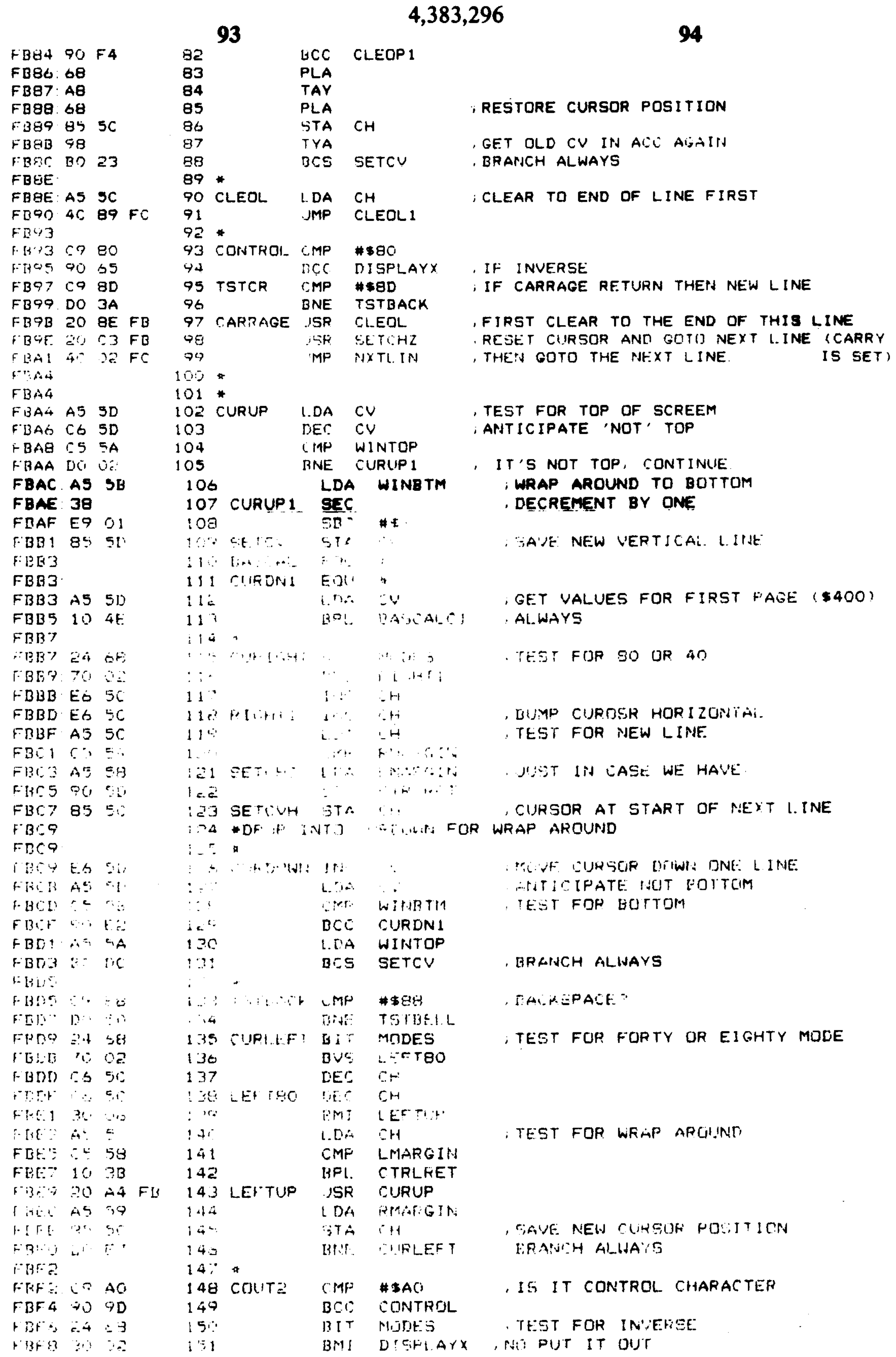
```
F9A2: A5 75      172           LDA   A1H
F9A4: E5 77      173           SBC   A2H
F9A6: 05 30      174           ORA   TEMP
F9A8: D0 01      175           BNE   RETA1       ; IF A1 LESS THAN OR EQUAL TO A2
F9AA: 18         176           CLC               ; THEN CARRY CLEAR ON RETURN
F9AB: 60         177 RETA1     RTS
F9AC:            178 *
F9AC:            179 *
F9AC: 48         180 PRBYTE    PHA               ; SAVE LOW NIBBLE
F9AD: 4A         181           LSR   A
F9AE: 4A         182           LSR   A           ; SHIFT HI NIBBLE TO PRINT.
F9AF: 4A         183           LSR   A
F9B0: 4A         184           LSR   A
F9B1: 20 B7 F9   185           JSR   PRHEXZ
F9B4: 68         186           PLA
F9B5: 29 0F      187 PRHEX     AND   #$0F        ; STRIP HI NIBBLE
F9B7: 09 B0      188 PRHEXZ    ORA   #$B0        ; MAKE IT NUMERIC
F9B9: C9 BA      189           CMP   #$BA        ; IS IT >'9'
F9BB: 90 02      190           BCC   PRHEX2
F9BD: 69 06      191           ADC   #$6         ; MAKE IT 'A'-'F'
F9BF: 4C 25 FC   192 PRHEX2    JMP   COUT
F9C2:            193 *
F9C2: 20 AC F9   194 PRBYCOL   JSR   PRBYTE
F9C5:            195 *
F9C5: A9 BA      196 PRCOLON   LDA   #$BA        ; PRINT A COLON
F9C7: D0 F6      197           BNE   PRHEX2      ; BRANCH ALWAYS
F9C9:            198 *
F9C9: A9 07      199 TST80WID  LDA   #7          ; ANTICIPATE
F9CB: 24 68      200           BIT   MODES       ; TEST FOR 80
F9CD: 50 02      201           BVC   SVMASK
F9CF: A9 0F      202           LDA   #$F
F9D1: 85 69      203 SVMASK    STA   MASK
F9D3: 60         204           RTS
F9D4:            205 *
F9D4: 8A         206 A1PC      TXA               ; TEST FOR NEW PC
F9D5: F0 07      207           BEQ   OLDPC
F9D7: B5 74      208 A1PC1     LDA   A1L,X
F9D9: 95 72      209           STA   PCL,X
F9DB: CA         210           DEX
F9DC: 10 F9      211           BPL   A1PC1
F9DE: 60         212 OLDPC     RTS
F9DF:            213 *
F9DF: 85 69      214 ASCII1    STA   MASK        ; SAVE HI BIT STATUS
F9E1: A4 7D      215 ASCII2    LDY   YSAV        ; MOVE ASCII TO MEMORY
F9E3: B1 7E      216           LDA   (INBUF),Y
F9E5: E6 7D      217           INC   YSAV        ; BUMP FOR NEXT THING.
F9E7: A0 00      218           LDY   #0
F9E9: C9 A2      219           CMP   #$A2        ; ASCII " ?
F9EB: D0 05      220           BNE   ASCII3      ; NOPE, CONTINUE.
F9ED: A5 69      221           LDA   MASK
F9EF: 10 20      222           BPL   BITON       ; HE'S CHANGED MODES.
F9F1: 60         223           RTS               ; NO, HE'S DONE.
F9F2: C9 A7      224 ASCII3    CMP   #$A7        ; ASCII ' ?
F9F4: D0 05      225           BNE   CRCHK       ; NO, TEST FOR EOL.
F9F6: A5 69      226           LDA   MASK
F9F8: 30 1B      227           BMI   BITOFF      ; CHANGE MODES.
F9FA: 60         228           RTS
F9FB: C9 8D      229 CRCHK     CMP   #$8D        ; END OF LINE?
F9FD: F0 07      230           BEQ   ASCDONE     ; YES, FINISHED
F9FF: 25 69      231           AND   MASK
FA01: 20 AF FA   232           JSR   STOR1       ; GO STORE IT!
FA04: D0 DB      233           BNE   ASCII2      ; DO NEXT.
```

```
FA06 60         234 ASCDONE RTS
FA07            235 *
FA07 38         236 ASCII    SEC                    ;INDICATE HI ON.
FA08 90         237          DFB   $90              ;(BCC - NEVER TAKEN)
FA09 18         238 ASCII0   CLC                    ;INDICATE HI OFF
FA0A AA         239 CKMDE    TAX                    ;SAVE STATE
FA0B 86 7C      240          STX   STATE           ;RETAIN STATE
FA0D 49 BA      241          EOR   #$BA             ;ARE WE IN STORE MODE?
FA0F D0 7D      242          BNE   ERROR
FA11 A9 FF      243 BITON    LDA   #$FF             ;SET HI BIT UNMASKED
FA13 B0 CA      244          BCS   ASCII1
FA15 A9 7F      245 BITOFF   LDA   #$7F             ;MASK HI BIT
FA17 10 C6      246          BPL   ASCII1          ;ALWAYS
FA19 2C 00 C0   247 REPEAT   BIT   KBD             ;REPEAT UNTIL KEYPRESS
FA1C 10 03      248          BPL   REPEAT1
FA1E 4C 0F FD   249          JMP   KEYIN
FA21 68         250 REPEAT1  PLA                    ;CLEAN UP STACK
FA22 68         251          PLA
FA23 4C 12 F9   252          JMP   SCAN
FA26            253 *
FA26            254 *
FA26 20 A0 FA   255 CRMON    JSR   BL1
FA29 4C 08 F9   256          JMP   MONZ
FA2C            257 *
FA2C 20 9B F9   258 MOVE     JSR   TSTA1            ;DON'T MOVE ANYTHING IF ILLEGAL INPUT
FA2F B0 5D      259          BCS   ERROR
FA31 B1 74      260 MOVNXT   LDA   (A1L),Y          ;MOVE A BYTE
FA33 91 7A      261          STA   (A4L),Y
FA35 20 8C F9   262          JSR   NXTA4            ;BUMP BOTH A1 AND A4
FA38 90 F7      263          BCC   MOVNXT
FA3A 60         264          RTS                    ;ALL DONE WITH MOVE
FA3B            265 *
FA3B            266 *
FA3B 20 9B F9   267 VRFY     JSR   TSTA1           ,TEST VALID RANGE
FA3E B0 4E      268          BCS   ERROR
FA40 B1 74      269 VRFY1    LDA   (A1L),Y          ;COMPARE BYTE FOR BYTE
FA42 D1 7A      270          CMP   (A4L),Y          ;MATCH?
FA44 F0 06      271          BEQ   VRFY2            ;YES, DO NEXT.
FA46 20 52 FA   272          JSR   MISMATCH         ;PRINT BOTH BYTES
FA49 20 EF FC   273          JSR   CROUT            ;GOTO NEWLINE
FA4C 20 8C F9   274 VRFY2    JSR   NXTA4           ,BUMP BOTH A1 AND A4
FA4F 90 EF      275          BCC   VRFY1
FA51 60         276          RTS                    ;VERIFY DONE.
FA52            277 *
FA52 A5 7B      278 MISMATCH LDA   A4H              ;PRINT ADDRESS OF A4
FA54 20 AC F9   279          JSR   PRBYTE
FA57 A5 7A      280          LDA   A4L
FA59 20 C2 F9   281          JSR   PRBYCOL          ;OUTPUT A COLON FOR SEPARATOR
FA5C B1 7A      282          LDA   (A4L),Y          ;AND THE DATA...
FA5E 20 70 FA   283          JSR   PRBYTSP          ;PRINT THE BYTE AND A SPACE
FA61 20 73 FA   284 PRINTA1  JSR   PRSPC            ;LEAD WITH A SPACE
FA64 A5 75      285          LDA   A1H              ;OUTPUT ADDRESS A1
FA66 20 AC F9   286          JSR   PRBYTE
FA69 A5 74      287          LDA   A1L
FA6B 20 C2 F9   288          JSR   PRBYCOL          ,SEPARATE WITH A COLEN
FA6E B1 74      289 PRA1BYTE LDA   (A1L),Y          ;PRINT BYTE POINTED TO BY A1
FA70 20 AC F9   290 PRBYTSP  JSR   PRBYTE
FA73 A9 A0      291 PRSPC    LDA   #$A0             ;PRINT A SPACE
FA75 4C 25 FC   292          JMP   COUT             ;END VIA OUTPUT ROUTINE.
FA78            293 *
FA78 4C F8 03   294 USER     JMP   USERADR
FA7B            295 *
FA7B 68         296 JUMP     PLA
FA7C 68         297          PLA                    ;LEAVE STACK WITH NOTHIN' ON IT
FA7D 20 D4 F9   298 GO       JSR   A1PC             ;STUFF PROGRAM COUNTER
FA80 6C 72 00   299          JMP   (PCL)            ;JUMP TO USER PROG
FA83            300 *
FA83            301 RWERROR EQU    *               ,PRINT ERROR NUMBER
```

```
FA83 20 AC F9   302          JSR  PRBYTE       ;PRINT THE OFFENDER
FA86 A9 A1      303          LDA  #$A1         ,FOLLOWED BY A "!"
FA88 20 25 FC   304          JSR  COUT
FA8B 20 07 FD   305 ERROR2   JSR  NOSTOP       ,OUTPUT A CARRAGE RETURN (NO STOPLST)
FA8E 4C 04 F9   306 ERROR    JMP  MON
FA91            307 *
FA91 A5 76      308 DEST     LDA  A2L          ;COPY A2 TO A4 FOR DESTINATION OF
FA93 85 7A      309          STA  A4L
FA95 A5 77      310          LDA  A2H
FA97 85 7B      311          STA  A4H
FA99 60         312          RTS
FA9A            313 *
FA9A 20 A4 FA   314 SEP      JSR  SPCE         ;SEPARATOR  TEST STORE MODE OR DUMP
FA9D 98         315          TYA               ,  ZERO MODE
FA9E F0 1D      316          BEQ  SETMDZ       ;BRANCH ALWAYS
FAA0            317 *
FAA0 C6 7D      318 BL1      DEC  YSAV         ;TEST FOR NO LINE
FAA2 F0 45      319          BEQ  DUMP8        ;IF NO LINE, GIVEN A ROW OF BYTES
FAA4 C8         320 SPCE     DEY               ,TEST IF AFTER ANOTHER SPACE
FAA5 D0 16      321          BNE  SETMDZ
FAA7 C9 DA      322          CMP  #$DA         ;STORE MODE?
FAA9 D0 4B      323          BNE  TSTDUMP
FAAB 85 7C      324 STOR     STA  STATE        ,KEEP IT IN STORE STATE
FAAD A5 76      325          LDA  A2L          ;GET BYTE TO BE STORED
FAAF 91 78      326 STOR1    STA  (A3L),Y      ;PUT IT IN MEMORY
FAB1 E6 78      327          INC  A3L          ,BUMP POINTER
FAB3 D0 02      328          BNE  DUMMY
FAB5 E6 79      329          INC  A3H
FAB7 60         330 DUMMY    RTS               ;ALSO USED FOR '/' TO CLEAR MODE
FAB8            331 *
FAB8 A4 7D      332 SETMODE  LDY  YSAV         ;USE INPUT CHARACTER
FABA 88         333          DEY
FABB B1 7E      334          LDA  (INBUF),Y    ,TO SET MODE
FABD 85 7C      335 SETMDZ   STA  STATE
FABF 60         336          RTS
FAC0            337 *
FAC0 A9 01      338 READ     LDA  #1           ;SET DISK COMMAND TO READ
FAC2 2C         339          DFB  $2C          ,DUMMY BIT TO SKIF 2 BYTES
FAC3 A9 02      340 WRTE     LDA  #2           ;SET DISK COMMAND TO WRITE
FAC5 85 87      341 SAVCMD   STA  IBCMD
FAC7 A5 74      342 RWLOOP   LDA  A1L
FAC9 85 85      343          STA  IBBUFP       ;COMMAND FORMAT IS
FACB A5 75      344          LDA  A1H          ;  BLOCKNUMBER<ADDRESS>CMDADDRESS
FACD 85 86      345          STA  IBBUFP+1
FACF A6 7B      346          LDX  A4H          ,SEND BLOCK NUMBER VIA X & A
FAD1 A5 7A      347          LDA  A4L
FAD3 78         348          SE1               ;NO INTERUPTS WHILE IN MONITOR
FAD4 20 ?? F?   349          JSR  BLOCKIO      ;DO DISKO FEVER
FAD7 B0 AA      350          BCS  RWERROR      ;GIVE UP IF ERROR ENCOUNTERED
FAD9 E6 7A      351          INC  A4L          ,BUMP BLOCK NUMBER
FADB D0 02      352          BNE  NOVER
FADD E6 7B      353          INC  A4H
FADF E6 75      354 NOVER    INC  A1H          ;BUMP RAM ADDRESS BY 512 BYTES
FAE1 E6 75      355          INC  A1H
FAE3 20 9B F9   356          JSR  TSTA1        ,TEST FOR FINISHED
FAE6 90 DF      357          BCC  RWLOOP       ,NOT DONE. DO NEXT BLOCK
FAE8 60         358          RTS
FAE9            359 *
FAE9            360          CHN  MON9B
FAE9              1 DUMP8    EQU  *            ,OUTPUT 1 ROW OF BYTES
FAE9 A5 75        2          LDA  A1H
FAEB 85 77        3          STA  A2H
FAED 20 C9 F9     4          JSR  TSTROWID     ;GET WIDTH MASK INTO ACC
FAF0 05 74        5          ORA  A1L
FAF2 85 76        6          STA  A2L
FAF4 D0 06        7          BNE  DUMP0        ;BRANCH ALWAYS
FAF6              8 *
FAF6 4A           9 TSTDUMP  LSR  A            ;DUMP?
FAF7 B0 95       10 ERROR1   BCS  ERROR
```

“APPLE_PAT_4_383_296_54” 136 KB 2000-02-27 dpi: 300h x 300v pix: 1842h x 2732v

```
FAF9 20 C9 F9    11 DUMP     JSR  TST80WID    ;SET FOR EITHER 80 OR 40 COLUMNS
FAFC A5 74       12 DUMP0    LDA  A1L         ;USE A4 FOR ASCII DUMP
FAFE 85 7A       13          STA  A4L
FB00 A5 75       14          LDA  A1H
FB02 85 7B       15          STA  A4H
FB04 20 9B [illegible] 16          JSR  TSTA1       ;TEST FOR VALID RANGE
FB07 B0 EE       17          BCS  ERROR1
FB09 20 61 FA    18 DUMP1    JSR  PRINTA1     ;PRINT ADDRESS AND FIRST BYTE
FB0C 20 92 [illegible] 19 DUMP2    JSR  NXTA1
FB0F B0 10       20          BCS  DUMPASC     ;END WITH ASCII
FB11 A5 74       21          LDA  A1L         ;TEST END OF LINE
FB13 25 59       22          AND  MASK        ;FOR 40/80 COLUMN
FB15 D0 05       23          BNE  DUMP3
FB17 20 21 FB    24          JSR  DUMPASC
FB1A D0 ED       25          BNE  DUMP1       ;BRANCH ALWAYS
FB1C 20 6E FA    26 DUMP3    JSR  PRA1BYTE    ;GO PRINT NEXT BYTE AND A SPACE
FB1F D0 EB       27          BNE  DUMP2       ;ALWAYS (ACC JUST PULLED AS $A0)
FB21             28 *
FB21 A5 7A       29 DUMPASC LDA  A4L          ;RESET TO BEGINING OF LINE
FB23 85 74       30          STA  A1L
FB25 A5 7B       31          LDA  A4H
FB27 85 75       32          STA  A1H
FB29 20 73 [illegible] 33          JSR  PRSPC      ;PRINT AN EXTRA SPACE
FB2C A0 00       34 ASC1     LDY  #0          ;TO INDEX MEMORY INDIRECT
FB2E B1 74       35          LDA  (A1L),Y
FB30 09 80       36          ORA  #$80        ;SET NORMAL VIDEO
FB32 C9 A0       37          CMP  #$A0        ;TEST FOR CONTROL CHARACTERS
FB34 B0 02       38          BCS  ASC2        ;OK TO PRINT NON CONTROLS
FB36 A9 AE       39          LDA  #$AE        ;OTHERWISE PRINT A SPACE
FB38 20 25 [illegible] 40 ASC2     JSR  COUT        ;PUT IT OUT
FB3B 20 3C [illegible] 41          JSR  NXTA4       ;BUMP BOTH A1 AND A4
FB3E B0 06       42          BCS  ASC3        ;FINISHED
FB40 A5 74       43          LDA  A1L         ;TEST END OF LINE
FB42 25 59       44          AND  MASK
FB44 D0 E6       45          BNE  ASC1        ;NOT DONE, PRINT NEXT
FB46 4C [illegible] 46 ASC3     JMP  CROUT
FB49             47 *
FB49             48 *
FB49             49 *
FB49 38          50 COL80    SEC              ;INDICATE 80 COLUMNS DESIRED
FB4A AD 53 C0    51          LDA  $C053       ;GOTO 80 COLUMN MODE
FB4D B0 04       52          BCS  SET80       ;BRANCH ALWAYS
FB4F             53 *
FB4F 18          54 COL40    CLC              ;INDICATE 40 COLUMNS DESIRED
FB50 AD 52 C0    55          LDA  $C052       ;GOTO 40 COLUMN MODE
FB53 A5 68       56 SET80    LDA  MODES
FB55 09 40       57          ORA  #$40        ;ASSUME 80
FB57 B0 02       58          BCS  SET80A      ;AND BRANCH IF IT IS
FB59 29 BF       59          AND  #$BF        ;BUT FIX FOR 40 IF NOT
FB5B 85 68       60 SET80A   STA  MODES
FB5D 09 7F       61          ORA  #$7F        ;ISOLATE BIT 7
FB5F 29 A0       62          AND  #$A0        ;(BIT 7 SETS NORMAL/INVERSE)
FB61 85 66       63          STA  FORGND
FB63 B0 02       64          BCS  SET80B      ;AGAIN ASSUMES 80 COLUMNS
FB65 A9 F0       65          LDA  #$F0        ;IF NOT, SET FOR/BACKGROUND COLOR
FB67 85 67       66 SET80B   STA  BKGND
FB69             67 *
FB69 A5 58       68 CLSCRN   LDA  LMARGIN     ;SET CURSOR TO TOP LEFT OF WINDOW
FB6B 85 5C       69          STA  CH
FB6D A5 5A       70          LDA  WINTOP
FB6F 85 5D       71          STA  CV          ;NOW DROP INTO CLEAR END  OF PAGE
FB71             72 *
FB71 A5 5C       73 CLEOP    LDA  CH          ;SAVE CURRENT CURSOR POSITION
FB73 48          74          PHA
FB74 A5 5D       75          LDA  CV
FB76 48          76          PHA
FB77 20 B1 FB    77          JSR  SETCV
FB7A 20 8E FB    78 CLEOP1   JSR  CLEOL       ;CLEAR TO END OF FIRST LINE
FB7D A5 58       79          LDA  LMARGIN
FB7F 85 5C       80          STA  CH
FB81 20 C9 FB    81          JSR  CURDOWN     ;GOTO NEXT LINE
```



```
FB84 90 F4                      BCC   CLEOP1
FB86:68                         PLA
FB87:A8                         TAY
FB88:68                         PLA                ;RESTORE CURSOR POSITION
FB89 85 5C                      STA   CH
FB8B 98                         TYA                ,GET OLD CV IN ACC AGAIN
FB8C B0 23                      BCS   SETCV        ,BRANCH ALWAYS
FB8E                        *
FB8E:A5 5C                  CLEOL   LDA   CH           ;CLEAR TO END OF LINE FIRST
FB90 4C 89 FC                   JMP   CLEOL1
FB93                        *
FB93 C9 80                  CONTROL CMP   #$80
FB95 90 65                      BCC   DISPLAYX     ,IF INVERSE
FB97 C9 8D                  TSTCR   CMP   #$8D         ;IF CARRAGE RETURN THEN NEW LINE
FB99 D0 3A                      BNE   TSTBACK
FB9B 20 8E FB               CARRAGE JSR   CLEOL        ,FIRST CLEAR TO THE END OF THIS LINE
FB9E 20 C3 FB                   JSR   SETCHZ       ,RESET CURSOR AND GOTO NEXT LINE (CARRY
FBA1 4C 02 FC                   JMP   NXTLIN       ,THEN GOTO THE NEXT LINE.          IS SET)
FBA4                        *
FBA4                        *
FBA4 A5 5D                  CURUP   LDA   CV           ,TEST FOR TOP OF SCREEM
FBA6 C6 5D                      DEC   CV           ;ANTICIPATE 'NOT' TOP
FBA8 C5 5A                      CMP   WINTOP
FBAA D0 02                      BNE   CURUP1       , IT'S NOT TOP, CONTINUE
FBAC A5 5B                      LDA   WINBTM       ;WRAP AROUND TO BOTTOM
FBAE 38                     CURUP1  SEC                ,DECREMENT BY ONE
FBAF E9 01                      SBC   #$1
FBB1 85 5D                  SETCV   STA   CV           ;SAVE NEW VERTICAL LINE
FBB3                        BASCAL  EQU   *
FBB3                        CURDN1  EQU   *
FBB3 A5 5D                      LDA   CV           ;GET VALUES FOR FIRST PAGE ($400)
FBB5 10 4E                      BPL   BASCALC1     ,ALWAYS
FBB7                        *
FBB7 24 6B                  CURIGHT BIT   MODES        ,TEST FOR 80 OR 40
FBB9:70 02                      BVS   RIGHT1
FBBB E6 5C                      INC   CH
FBBD E6 5C                  RIGHT1  INC   CH           ,BUMP CUROSR HORIZONTAL
FBBF A5 5C                      LDA   CH           ,TEST FOR NEW LINE
FBC1 C5 59                      CMP   RMARGIN
FBC3 A5 58                  SETCHZ  LDA   LMARGIN      ,JUST IN CASE WE HAVE
FBC5 90 5D                      BCC   CTRLRET
FBC7 85 5C                  SETCVH  STA   CH           ,CURSOR AT START OF NEXT LINE
FBC9                        *DROP INTO CURDOWN FOR WRAP AROUND
FBC9                        *
FBC9 E6 5D                  CURDOWN INC   CV           ;MOVE CURSOR DOWN ONE LINE
FBCB A5 5D                      LDA   CV           ;ANTICIPATE NOT BOTTOM
FBCD C5 5B                      CMP   WINBTM       ,TEST FOR BOTTOM
FBCF 90 E2                      BCC   CURDN1
FBD1 A5 5A                      LDA   WINTOP
FBD3 B0 DC                      BCS   SETCV        ,BRANCH ALWAYS
FBD5                        *
FBD5 C9 88                  TSTBACK CMP   #$88         ,BACKSPACE?
FBD7 D0 30                      BNE   TSTBELL
FBD9 24 6B                  CURLEFT BIT   MODES        ;TEST FOR FORTY OR EIGHTY MODE
FBDB 70 02                      BVS   LEFT80
FBDD C6 5C                      DEC   CH
FBDF C6 5C                  LEFT80  DEC   CH
FBE1 30 06                      BMI   LEFTUP
FBE3 A5 5C                      LDA   CH           ;TEST FOR WRAP AROUND
FBE5 C5 58                      CMP   LMARGIN
FBE7 10 3B                      BPL   CTRLRET
FBE9 20 A4 FB               LEFTUP  JSR   CURUP
FBEC A5 59                      LDA   RMARGIN
FBEE 85 5C                      STA   CH           ,SAVE NEW CURSOR POSITION
FBF0 D0 E7                      BNE   CURLEFT       BRANCH ALWAYS
FBF2                        *
FBF2 C9 A0                  COUT2   CMP   #$A0         ,IS IT CONTROL CHARACTER
FBF4 90 9D                      BCC   CONTROL
FBF6 24 6B                      BIT   MODES        ,TEST FOR INVERSE
FBF8 30 02                      BMI   DISPLAYX   ,NO PUT IT OUT
```

"APPLE_PAT_4_383_296_56" 137 KB 2000-02-27 dpi: 300h x 300v pix: 1792h x 2739v

```
FBFA 29 7F     152           AND  #$7F         ;STRIP HI BIT
FBFC 20 9D FC  153 DISPLAYX  JSR  DISPLAY
FBFF           154 *
FBFF 20 B7 FD  155 INCHORZ   JSR  CURIGHT      ;MOVE CURSOR RIGHT
FC02 B0 57     156 DXFLIN    BCS  SCROLL       ; IF IS BOTTOM, RESET CH=0 AND SCROLL
FC04 60        157           RTS               ;RESET CH ONLY.
FC05           158 *
FC05 08        159 BASCALC1  PHP               ;CALC BASE ADR IN BAS4L,H
FC06 48        160           PHA
FC07 4A        161           LSR  A            ;FOR GIVEN LINE NO.
FC08 29 03     162           AND  #$03         ;O<=LINE NO.<=$17
FC0A 09 04     163           ORA  #$04         ;ARG=000ABCDE, GENERATE
FC0C 85 5F     164           STA  BAS4H        ;BAS4H=000001CD
FC0E 49 0C     165           EOR  #$C
FC10 85 61     166           STA  BAS8H
FC12 68        167           PLA               ;AND
FC13 29 18     168           AND  #$18         ;BAS4L=EABAB000
FC15 90 02     169           BCC  BSCLC2
FC17 69 7F     170           ADC  #$7F
FC19 85 5E     171 BSCLC2    STA  BAS4L
FC1B 0A        172           ASL  A
FC1C 0A        173           ASL  A
FC1D 05 5E     174           ORA  BAS4L
FC1F 85 5E     175           STA  BAS4L
FC21 85 60     176           STA  BAS8L         ;SAME FOR PAGE 2
FC23 28        177           PLP
FC24 60        178 CTRLRET   RTS
FC25           179 *
FC25 48        180 COUT      PHA                ;SAVE CHARACTER
FC26 94 6D     181           STY  TEMPY
FC28 66 6C     182           STX  TEMPX
FC2A 20 33 FC  183           JSR  COUT1
FC2D A4 6D     184           LDY  TEMPY
FC2F A6 6C     185           LDX  TEMPX
FC31 68        186           PLA
FC32 60        187           RTS
FC33           188 *
FC33 6C 6E 00  189 COUT1     JMP  (CSWL)        ;NORMALLY COUT1
FC36           190 *
FC36 C9 87     191 TSTBELL   CMP  #$87          ;BELL?
FC38 D0 18     192           BNE  LNFD          ;NO TEST FOR FORM FEED
FC3A           193 *
FC3A A2 10     194 BELL      LDX  #$10
FC3C 8A        195           TXA
FC3D A8        196 BELL1     TAY
FC3E 2C D8 FF  197 BELL2     BIT  $FFD8
FC41 F0 FB     198           BEQ  BELL2
FC43 2C D8 FF  199 BELL3     BIT  $FFD8
FC46 D0 FB     200           BNE  BELL3
FC48 88        201           DEY
FC49 D0 F3     202           BNE  BELL2
FC4B 2C 30 C0  203           BIT  $C030
FC4E CA        204           DEX
FC4F D0 EC     205           BNE  BELL1
FC51 60        206           RTS
FC52           207 *
FC52 C9 8A     208 LNFD      CMP  #$8A          ;LINE FEED?
FC54 D0 CE     209           BNE  CTRLRET
FC56 20 [illegible] 210      JSR  CURDOWN       ;MOVE CURSOR DOWN A LINE
FC59 90 C9     211           BCC  CTRLRET       ;BRANCH IF NO SCROLL NECESSARY.
FC5B           212 *
FC5B A5 5A     213 SCROLL    LDA  WINTOP        ;START WITH TOP LINE
FC5D 48        214           PHA                ;SAVE IT FOR NOW
FC5E 20 [illegible] 215      JSR  SETCV         ;GET BASCALC FOR THIS LINE
FC61 A2 03     216 SCRL1     LDX  #3            ;MOVE CURRENT BASCALC AS DESTINATION
FC63 B5 5E     217 SCRL2     LDA  BAS4L,X
FC65 95 [illegible] 218      STA  TBAS4L,X   ;(TEMPORARY BASE ADDR.)
FC67 CA        219           DEX
FC68 10 F9     220           BPL  SCRL2
FC6A 68        221           PLA                ;GET DESTINATION LINE
```

```
FC6B: 18                  CLC
FC6C: 69 01               ADC   #1          ;CALCULATE SOURCE LINE
FC6E: C5 58               CMP   WINBTM      ;IS IT THE LAST LINE?
FC70: B0 15               BCS   LASTLN      ;YES, CLEAR IT
FC72: 48                  PHA               ;SAVE AS NEXT DESTINATION LINE
FC73: 20 B1 FB            JSR   SETCV       ;GET BASE ADDR FOR SOURCE LINE
FC76: A5 59               LDA   RMARGIN     ; MOVE SOURCE TO DESTINATION
FC78: 4A                  LSR   A           ;DIVIDE BY 2
FC79: A8                  TAY
FC7A: 88          SCRL3   DEY               ;DONE YET?
FC7B: 30 E4               BMI   SCRL1       ;YES, DO NEXT LINE
FC7D: B1 5E               LDA   (BAS4L),Y
FC7F: 91 62               STA   (TBAS4L),Y
FC81: B1 60               LDA   (BAS8L),Y   ;MOVE BOTH PAGES
FC83: 91 64               STA   (TBAS8L),Y
FC85: 90 F3               BCC   SCRL3       ;BRANCH ALWAYS
FC87: A5 58       LASTLN  LDA   LMARGIN     ;BLANK FILL THE LAST LINE
FC89: 4A          CLEOL1  LSR   A           ;DIVIDE BY 2
FC8A: A8                  TAY
FC8B: B0 04               BCS   CLEOL2
FC8D: A5 66               LDA   FORGND      ;(NORMALLY A SPACE)
FC8F: 91 5E               STA   (BAS4L),Y
FC91: A5 67       CLEOL2  LDA   BKGND       ;(IF 80 COLUMNS, ALSO A SPACE)
FC93: 91 60               STA   (BAS8L),Y
FC95: C8                  INY
FC96: 98                  TYA               ;TEST FOR END OF LINE
FC97: 0A                  ASL   A           ;MULT BY 2 AGAIN
FC98: C5 59               CMP   RMARGIN
FC9A: 90 ED               BCC   CLEOL1      ;CONTINUE IF MORE TO DO
FC9C: 60                  RTS               ;ALL DONE.
FC9D                *
FC9D: 24 68       DISPLAY BIT   MODES       ;TEST FOR 40 OR 80
FC9F: 70 0C               BVS   DSPL80      ;STORE THE SINGLE CHARACTER AND RETURN
FCA1: 46 5C               LSR   CH          ;INSURE PROPER 40 COLUMN DISPLAY
FCA3: 06 5C               ASL   CH          ;BY DROPPING BIT 0
FCA5: 20 AD FC            JSR   DSPL80      ;DISPLAY IN $400 PAGE.
FCA8: A5 67               LDA   BKGND       ;ALSO SET BACKGROUND COLOR
FCAA: 91 60       DSPBKGND STA  (BAS8L),Y
FCAC: 60                  RTS
FCAD                *
FCAD: 48          DSPL80  PHA               ;PRESERVE CHARACTER
FCAE: A5 5C               LDA   CH          ;DETERMINE WICH PAGE
FCB0: 4A                  LSR   A
FCB1: A8                  TAY
FCB2: 68                  PLA
FCB3: B0 F5               BCS   DSPBKGND    ;BRANCH IF $800 PAGE
FCB5: 91 5E               STA   (BAS4L),Y
FCB7: 60                  RTS
FCB8                *
FCB8: B1 7E       NOTCR   LDA   (INBUF),Y   ;ECHO CHARACTER
FCBA: 20 25 FC            JSR   COUT
FCBD: C9 88               CMP   #$88        ;BACKSPACE?
FCBF: F0 1D               BEQ   BKSPCE
FCC1: C9 98               CMP   #$98        ;CANCEL?
FCC3: F0 08               BEQ   CANCEL
FCC5: E6 80               INC   TEMP
FCC7: A5 80               LDA   TEMP
FCC9: C9 50               CMP   #INBUFLEN
FCCB: D0 17               BNE   NXTCHAR     ;NO WRAP AROUND ALLOWED.
FCCD: A9 DC       CANCEL  LDA   #$DC        ;OUTPUT BACKSLASH
FCCF: 20 25 FC            JSR   COUT
FCD2: 20 EF FC            JSR   CROUT
FCD5              GETLNZ  EQU   *
FCD5: A5 6B       GETLN   LDA   PROMPT
FCD7: 20 25 FC            JSR   COUT
FCDA: A0 01               LDY   #1
FCDC: 84 80               STY   TEMP        ;START AT BEGINNING OF INBUF
FCDE: A4 80       BKSPCE  LDY   TEMP
FCE0: F0 F3               BEQ   GETLN
FCE2: C6 80               DEC   TEMP        ;BACK UP INPUT BUFFER
FCE4: 20 60 FD    NXTCHAR JSR   RDCHAR      ;GET INPUT
FCE7: A4 80               LDY   TEMP
```

```
FCE9: 91 7E                 STA   (INBUF),Y
FCEB: C9 8D                 CMP   #$8D
FCED: D0 C9                 BNE   NOTCR
FCEF:           CROUT       EQU   *
FCEF: 2C 00 C0              BIT   KBD        ;TEST FOR START/STOP
FCF2: 10 13                 BPL   NOSTOP
FCF4: 20 2E FD              JSR   KEYIN3     ;READ KBD
FCF7: C9 A0                 CMP   #$A0       ;IS IT A SPACE?
FCF9: F0 07                 BEQ   STOPLST   ;YES, PAUSE TIL NEXT KEYPRESS.
FCFB: C9 8D                 CMP   #$8D       ;QUIT THIS OPERATION?
FCFD: D0 08                 BNE   NOSTOP    ;NO, IGNORE THIS KEY.
FCFF: 4C 8B FA              JMP   ERROR2     ;YES, RESTART
FD02: AD 00 C0  STOPLST     LDA   KBD
FD05: 10 FB                 BPL   STOPLST
FD07: A9 8D     NOSTOP      LDA   #$8D
FD09: 4C 25 FC              JMP   COUT
FD0C:           *
FD0C: 6C 70 00  RDKEY       JMP   (KSWL)
FD0F:           *
FD0F: A9 7F     KEYIN       LDA   #$7F       ;MAKE SURE FIRST IS CURSOR
FD11: 85 63                 STA   TBAS4H
FD13: 20 88 FD              JSR   PICK       ;GO READ SCREEN
FD16: 48        KEYIN1      PHA              ;SAVE CHR AT CURSOR POSITION
FD17: 20 35 FD              JSR   KEYWAIT    ;TEST FOR KEYPRESS
FD1A: B0 08                 BCS   KEYIN2     ;GO GET IT
FD1C: A5 69                 LDA   CURSOR    ;GIVE THEM AN UNDERSCORE FOR A TIME
FD1E: 20 9D FC              JSR   DISPLAY
FD21: 20 35 FD              JSR   KEYWAIT    ;GO SEE IF KEYPRESSED
FD24: 68        KEYIN2      PLA
FD25: 08                    PHP             ;SAVE KEYPRESS STATUS
FD26: 48                    PHA
FD27: 20 9D FC              JSR   DISPLAY
FD2A: 68                    PLA
FD2B: 28                    PLP
FD2C: 90 E8                 BCC   KEYIN1
FD2E: AD 00 C0  KEYIN3      LDA   KBD        ;READ KEYBOARD
FD31: 2C 10 C0  KEYIN4      BIT   KBDSTRB    ;CLEAR KEYBOARD STROBE
FD34: 60                    RTS
FD35: E6 62     KEYWAIT     INC   TBAS4L     ;JUST KEEP COUNTING
FD37: D0 09                 BNE   KWAIT2
FD39: E6 63                 INC   TBAS4H
FD3B: A9 7F                 LDA   #$7F       ;TEST FOR DONE
FD3D: 18                    CLC
FD3E: 25 63                 AND   TBAS4H
FD40: F0 05               BEQ   KEYRET     ;RETURN IF TIMED OUT
FD42: 0E 00 C0  KWAIT2    ASL   KBD
FD45: 90 EE               BCC   KEYWAIT
FD47: 60        KEYRET    RTS
FD48:           *
FD48            *
FD48            ESC3      EQU   *
FD48 20 77 FD             JSR   GOESC
FD4B A5 68      ESCAPE    LDA   MODES      ;SET TO + SIGN FOR CURSOR MOVES
FD4D 29 80                AND   #$80
FD4F 49 AB                EOR   #$AB
FD51: 85 69               STA   CURSOR
FD53: 20 0C FD  ESC1      JSR   RDKEY      ;READ NEXT CHARACTER
FD56. A0 08               LDY   #8         ;TEST FOR ESCAPE COMMAND
FD58 D9 F0 FF   ESC2      CMP   ESCTABL,Y
FD5B F0 EB                BEQ   ESC3
FD5D 88                   DEY
FD5E. 10 F8               BPL   ESC2       ;LOOP TIL FOUND OR DONE
FD60.           *
FD60. A9 80     RDCHAR    LDA   #$80       ;GO READ A CHARACTER
FD62. 25 68               AND   MODES
FD64 85 69                STA   CURSOR     ;SAVE STANDARD CURSOR.
FD66 20 0C FD             JSR   RDKEY
FD69: C9 9B               CMP   #$9B       ;ESCAPE CHARACTER?
FD6B: F0 DE               BEQ   ESCAPE
FD6D C9 95                CMP   #$95       ;FORWARD COPY?
FD6F D0 D6                BNE   KEYRET
FD71: 20 88 FD            JSR   PICK       ;GET CHARACTER FROM SCREEN
```

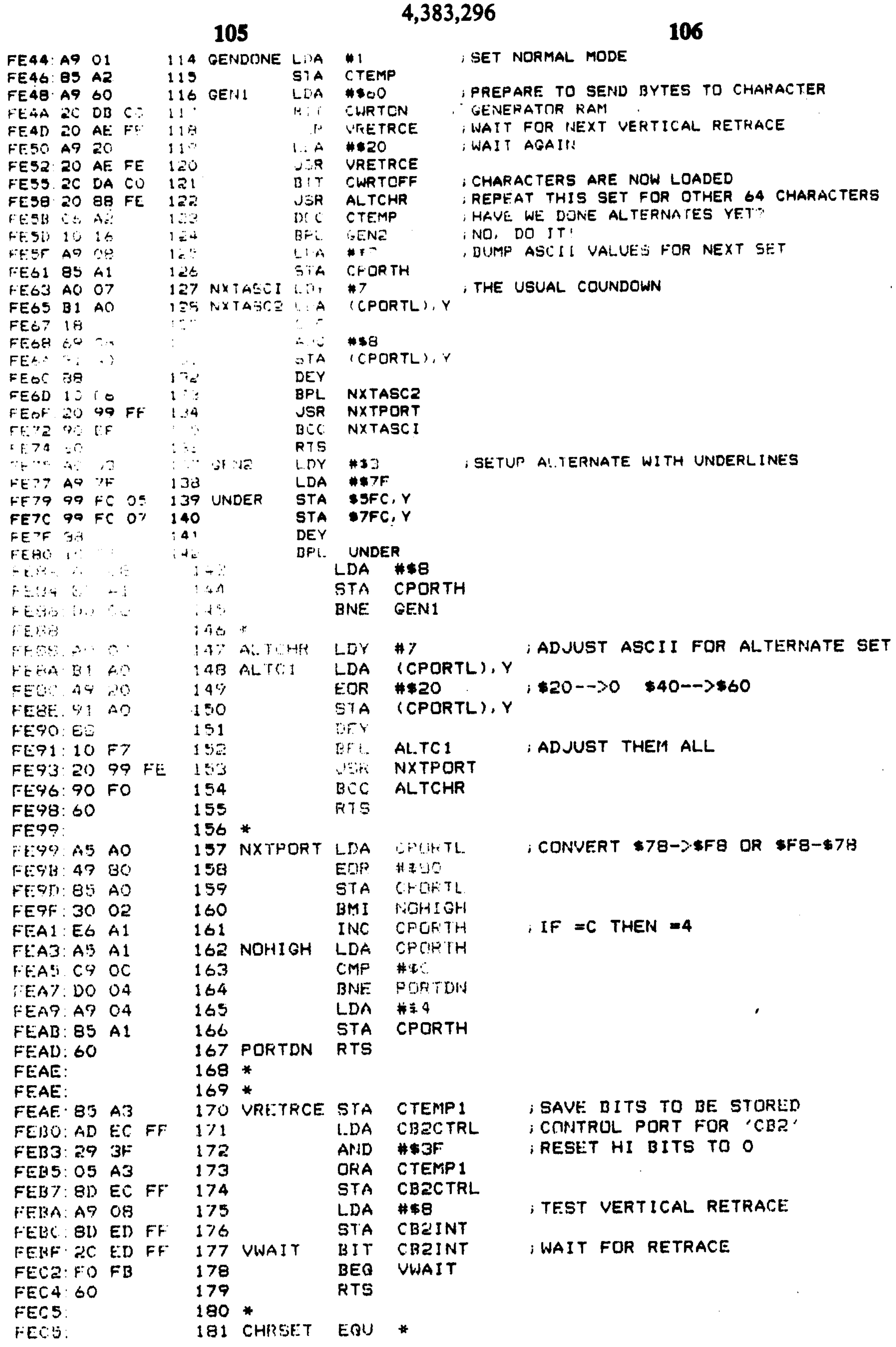
```
FC6B: 18                  CLC
FC6C: 69 01               ADC  #1          ;CALCULATE SOURCE LINE
FC6E: C5 5B               CMP  WINDTM      ;IS IT THE LAST LINE?
FC70  B0 15               BCS  LASTLN      ,YES, CLEAR IT
FC72: 48                  PHA              ,SAVE AS NEXT DESTINATION LINE
FC73: 20 B1 FB            JSR  SETCV       ,GET BASE ADDR FOR SOURCE LINE
FC76  A5 59               LDA  RMARGIN     , MOVE SOURCE TO DESTINATION
FC78. 4A                  LSR  A           ;DIVIDE BY 2
FC79  A8                  TAY
FC7A  88         SCRL3    DEY              ,DONE YET?
FC7B  30 E4               BMI  SCRL1       ,YES, DO NEXT LINE
FC7D  B1 5E               LDA  (BAS4L),Y
FC7F: 91 62               STA  (TBAS4L),Y
FC81: B1 60               LDA  (BAS8L),Y   ;MOVE BOTH PAGES
FC83  91 64               STA  (TBAS8L),Y
FC85  90 F3               BCC  SCRL3       ,BRANCH ALWAYS
FC87  A5 58      LASTLN   LDA  LMARGIN     ,BLANK FILL THE LAST LINE.
FC89  4A         CLEOL1   LSR  A           ;DIVIDE BY 2
FC8A  A8                  TAY
FC8B: B0 04               BCS  CLEOL2
FC8D  A5 66               LDA  FORGND      ,(NORMALLY A SPACE)
FC8F  91 5E               STA  (BAS4L),Y
FC91  A5 67      CLEOL2   LDA  BKGND       ,(IF 80 COLUMNS, ALSO A SPACE)
FC93. 91 60               STA  (BAS8L),Y
FC95  C8                  INY
FC96  98                  TYA              ;TEST FOR END OF LINE
FC97  0A                  ASL  A           ,MULT BY 2 AGAIN
FC98  C5 59               CMP  RMARGIN
FC9A. 90 ED               BCC  CLEOL1      ,CONTINUE IF MORE TO DO
FC9C  60                  RTS              ;ALL DONE.
FC9D                 *
FC9D  24 68      DISPLAY  BIT  MODES       ;TEST FOR 40 OR 80
FC9F  70 0C               BVS  DSPL80      ;STORE THE SINGLE CHARACTER AND RETURN
FCA1  46 5C               LSR  CH          ,INSURE PROPER 40 COLUMN DISPLAY
FCA3  06 5C               ASL  CH          ;BY DROPPING BIT 0
FCA5. 20 AD FC            JSR  DSPL80      ,DISPLAY IN $400 PAGE.
FCA8: A5 67               LDA  BKGND       ;ALSO SET BACKGROUND COLOR
FCAA  91 60      DSPBKGND STA  (BAS8L),Y
FCAC  60                  RTS
FCAD                 *
FCAD  48         DSPL80   PHA              ,PRESERVE CHARACTER
FCAE  A5 5C               LDA  CH          ;DETERMINE WICH PAGE
FCB0: 4A                  LSR  A
FCB1  A8                  TAY
FCB2  68                  PLA
FCB3  B0 F5               BCS  DSPBKGND    ,BRANCH IF $800 PAGE
FCB5  91 5E               STA  (BAS4L),Y
FCB7. 60                  RTS
FCB8                 *
FCB8  B1 7E      NOTCR    LDA  (INBUF),Y   ;ECHO CHARACTER
FCBA  20 25 FC            JSR  COUT
FCBD  C9 88               CMP  #$88        ,BACKSPACE?
FCBF  F0 1D               BEQ  BKSPCE
FCC1  C9 98               CMP  #$98        ,CANCEL?
FCC3  F0 08               BEQ  CANCEL
FCC5: E6 80               INC  TEMP
FCC7  A5 80               LDA  TEMP
FCC9  C9 50               CMP  #INBUFLEN
FCCB: D0 17               BNE  NXTCHAR     ,NO WRAP AROUND ALLOWED.
FCCD: A9 DC      CANCEL   LDA  #$DC        ;OUTPUT BACKSLASH
FCCF  20 25 FC            JSR  COUT
FCD2: 20 EF FC            JSR  CROUT
FCD5.            GETLNZ   EQU  *
FCD5: A5 6B      GETLN    LDA  PROMPT
FCD7: 20 25 FC            JSR  COUT
FCDA: A0 01               LDY  #1
FCDC: 84 80               STY  TEMP        ;START AT BEGINNING OF INBUF
FCDE: A4 80      BKSPCE   LDY  TEMP
FCE0: F0 F3               BEQ  GETLN
FCE2: C6 80               DEC  TEMP        ;BACK UP INPUT BUFFER
FCE4: 20 60 FD   NXTCHAR  JSR  RDCHAR      ;GET INPUT
FCE7: A4 80               LDY  TEMP
```

```
FCE9: 91 7E      294          STA  (INBUF),Y
FCEB: C9 8D      295          CMP  #$8D
FCED: D0 C9      296          BNE  NOTCR
FCEF:            297 CROUT    EQU  *
FCEF: 2C 00 C0   298          BIT  KBD         ;TEST FOR START/STOP
FCF2: 10 13      299          BPL  NOSTOP
FCF4: 20 2E FD   300          JSR  KEYIN3      ;READ KBD
FCF7: C9 A0      301          CMP  #$A0        ;IS IT A SPACE?
FCF9: F0 07      302          BEQ  STOPLST    ;YES, PAUSE TIL NEXT KEYPRESS.
FCFB: C9 8D      303          CMP  #$8D        ;QUIT THIS OPERATION?
FCFD: D0 08      304          BNE  NOSTOP     ;NO, IGNORE THIS KEY.
FCFF: 4C 8B FA   305          JMP  ERROR2      ;YES, RESTART
FD02: AD 00 C0   306 STOPLST  LDA  KBD
FD05: 10 FB      307          BPL  STOPLST
FD07: A9 8D      308 NOSTOP   LDA  #$8D
FD09: 4C 25 FC   309          JMP  COUT
FD0C:            310 *
FD0C: 6C 70 00   311 RDKEY    JMP  (KSWL)
FD0F:            312 *
FD0F: A9 7F      313 KEYIN    LDA  #$7F        ;MAKE SURE FIRST IS CURSOR
FD11: 85 63      314          STA  TBAS4H
FD13: 20 88 FD   315          JSR  PICK        ;GO READ SCREEN
FD16: 48         316 KEYIN1   PHA              ;SAVE CHR AT CURSOR POSITION
FD17: 20 35 FD   317          JSR  KEYWAIT     ;TEST FOR KEYPRESS
FD1A: B0 08      318          BCS  KEYIN2      ;GO GET IT
FD1C: A5 69      319          LDA  CURSOR     ;GIVE THEM AN UNDERSCORE FOR A TIME
FD1E: 20 9D FC   320          JSR  DISPLAY
FD21: 20 35 FD   321          JSR  KEYWAIT     ;GO SEE IF KEYPRESSED
FD24: 68         322 KEYIN2   PLA
FD25: 08         323          PHP             ;SAVE KEYPRESS STATUS
FD26: 48         324          PHA
FD27: 20 9D FC   325          JSR  DISPLAY
FD2A: 68         326          PLA
FD2B: 28         327          PLP
FD2C: 90 E8      328          BCC  KEYIN1
FD2E: AD 00 C0   329 KEYIN3   LDA  KBD         ;READ KEYBOARD
FD31: 2C 10 C0   330 KEYIN4   BIT  KBDSTRB     ;CLEAR KEYBOARD STROBE
FD34: 60         331          RTS
FD35: E6 62      332 KEYWAIT  INC  TBAS4L      ;JUST KEEP COUNTING
FD37: D0 09      333          BNE  KWAIT2
FD39: E6 63      334          INC  TBAS4H
FD3B: A9 7F      335          LDA  #$7F        ;TEST FOR DONE
FD3D: 18         336          CLC
FD3E: 25 63      337          AND  TBAS4H
FD40: F0 05      338          BEQ  KEYRET      ;RETURN IF TIMED OUT
FD42: 0E 00 C0   339 KWAIT2   ASL  KBD
FD45: 90 EE      340          BCC  KEYWAIT
FD47: 60         341 KEYRET   RTS
FD48:            342 *
FD48             343 *
FD48             344 ESC3     EQU  *
FD48 20 77 FD    345          JSR  GOESC
FD4B A5 68       346 ESCAPE   LDA  MODES       ;SET TO + SIGN FOR CURSOR MOVES
FD4D 29 80       347          AND  #$80
FD4F 49 AB       348          EOR  #$AB
FD51: 85 69      349          STA  CURSOR
FD53: 20 0C FD   350 ESC1     JSR  RDKEY       ;READ NEXT CHARACTER
FD56. A0 08      351          LDY  #8          ;TEST FOR ESCAPE COMMAND
FD58 D9 F0 FF    352 ESC2     CMP  ESCTABL,Y
FD5B F0 EB       353          BEQ  ESC3
FD5D 88          354          DEY
FD5E. 10 F8      355          BPL  ESC2        ;LOOP TIL FOUND OR DONE
FD60             356 *
FD60. A9 80      357 RDCHAR   LDA  #$80        ;GO READ A CHARACTER
FD62. 25 68      358          AND  MODES
FD64 85 69       359          STA  CURSOR      ,SAVE STANDARD CURSOR.
FD66 20 0C FD    360          JSR  RDKEY
FD69: C9 9B      361          CMP  #$9B        ;ESCAPE CHARACTER?
FD6B: F0 DE      362          BEQ  ESCAPE
FD6D C9 95       363          CMP  #$95        ;FORWARD COPY?
FD6F D0 D6       364          BNE  KEYRET
FD71: 20 88 FD   365          JSR  PICK        ;GET CHARACTER FROM SCREEN
```

```
FD74 09 80              ORA   #$80        ,SET TO NORMAL ASCII
FD76:60                 RTS
FD77:              *
FD77:A9 FB         GOESC    LDA   #<CLSCRN
FD79 48                 PHA
FD7A B9 7F FD           LDA   ESCVECT,Y
FD7D 48                 PHA
FD7E 60                 RTS
FD7F               *
FD7F 8D            ESCVECT  DFB   CLEOL-1
FD80 70                 DFB   CLEOP-1
FD81 65                 DFB   CLSCRN-1
FD82 4E                 DFB   COL40-1
FD83 48                 DFB   COL80-1
FD84 D8                 DFB   CURLEFT-1
FD85 B6                 DFB   CURIGHT-1
FD86 C8                 DFB   CURDOWN-1
FD87 A3                 DFB   CURUP-1
FD88               *
FD88:A5 5C         PICK     LDA   CH          ,GET A CHARACTER AT CURRENT CURSOR POSITION
FD8A 4A                 LSR   A           ,DETERMINE WHICH PAGE
FD8B:A8                 TAY
FD9C 24 68              BIT   MODES       ,AND IF 80 COLUMN MODE
FD8E 50 05              BVC   PICK40      ,FORGET CARRY IF 40 COLUMNS
FD90 90 02              BCC   PICK40      ,GET CHARACTER FROM $400 PAGE
FD92 B1 60              LDA   (BAS8L),Y
FD94 60                 RTS
FD95 B1 5E         PICK40   LDA   (BAS4L),Y
FD97 60                 RTS
FD98               *
FD98:              CLDSTRT EQU   *
FD98:A9 03                  LDA   #$3
FD9A:8D D0 FF               STA   $FFD0       ;ZERO PAGE IS ON 3!
FD9D:              SETUP   EQU   *
FD9D:D8                     CLD               ;OF COURSE!
FD9E:A2 03                  LDX   #3
FDA0:86 7F                  STX   INBUF+1
FDA2:BD BC FF      SETUP1  LDA   NMIRQ,X
FDA5:9D CA FF               STA   $FFCA,X
FDA8:BD B4 FF               LDA   HOOKS,X
FDAB:95 6E                  STA   CSWL,X
FDAD:BD B8 FF               LDA   VBOUNDS,X
FDB0:95 58                  STA   LMARGIN,X
FDB2:CA                     DEX
FDB3:10 ED                  BPL   SETUP1
FDB5:85 82                  STA   IBDRVN
FDB7:A9 A0                  LDA   #$A0        ;INPUT BUFFER AT $3A0
FDB9:85 7E                  STA   INBUF
FDBB:A9 60                  LDA   #$60
FDBD:85 81                  STA   IBSLOT
FDBF:A9 FF                  LDA   #$FF
FDC1:85 68                  STA   MODES
FDC3:20 4F FB               JSR   COL40       ;SET 40 COLUMNS, CLEAR SCREEN
FDC6:              *
00A0:              ADR      EQU   $A0
00A0:              CPORTL   EQU   ADR
00A1               CPORTH   EQU   ADR+1
00A2               CTEMP    EQU   ADR+2
00A3               CTEMP1   EQU   ADR+3
00A4               YTEMP    EQU   ADR+4
00B4               ROWTEMP  EQU   ADR+20
C0DB               CWRTON   EQU   $C0DB
C0DA               CWR[illegible] EQU $C0DA
FFEC               [illegible] EQU $FFEC
FFED               [illegible] EQU $FFED
FDC6               *
FDC6.              *
FDC6 A9 78         GENENTR  LDA   #$78        ;INIT SCREEN INDX LOCATIONS
FDC8 85 A0                  STA   CPORTL
FDCA A9 08                  LDA   #$8
FDCC 85 A1                  STA   CPORTH
```

```
FDCE A9 F0              LDA  #240        ;SET UP INDEX TO CHRSET
FDD0 85 A4              STA  YTEMP
FDD2 A9 00              LDA  #0
FDD4 A8                 TAY
FDD5 95 B4     ZIPTEMPS STA  ROWTEMP,X
FDD7 E8                 INX
FDD8 E0 20              CPX  #$20
FDDA D0 F9              BNE  ZIPTEMPS
FDDC A9 05              LDA  #5          ;FAKE THE FIRST BIT PATTERN
FDDE 18                 CLC              ;(PHANTOM 9TH BIT SHIFTED AS BIT 0)
FDDF 08                 PHP
FDE0 48                 PHA
FDE1 86 A2     GENASC   STX  CTEMP       ;GENERATE THE ASCII
FDE3 A0 07     GASCI1   LDY  #7          ; CODES FOR THE FIRST PASS
FDE5 A6 A2     GASCI2   LDX  CTEMP
FDE7 8A        GASCI3   TXA
FDE8 91 A0              STA  (CPORTL),Y  ; $XXF=CHR 0 / 4
FDEA E8                 INX              ; $XXE=CHR 1   5
FDEB 88                 DEY              ; $XXD=CHR 2   6
FDEC 30 06              BMI  GASCI4      ; $XXC=CHR 3   7
FDEE C0 03              CPY  #$3         ; $XXB=CHR 0   4
FDF0 D0 F5              BNE  GASCI3      ; $XXA=CHR 1   5
FDF2 F0 F1              BEQ  GASCI2      ; $XX9=CHR 2   6
FDF4 20 99 FE  GASCI4   JSR  NXTPORT     ; $XX8=CHR 3   7
FDF7 B0 [illegible]     BCS  CBYTES      ;DO DECODE CHARACTER TABLE
FDF9 C9 0A              CMP  #$A         ;SECOND SET OF 4?
FDFB D0 E6              BNE  GASCI1
FDFD A2 [illegible]     LDX  #$[illegible]
FDFF D0 [illegible]     BNE  GENASC      ;BRANCH ALWAYS
FE01 68        CBYTES   PLA              ;RESTORE BIT PATTERN
FE02 28                 PLP
FE03 A2 17              LDX  #23         ;(4 CHARACTERS OF 6 ROWS)
FE05 A0 05     CCOLMS   LDY  #5          ;(FIVE COLUMNS)
FE07 36 B8     CSHFT    ROL  ROWTEMP+4,X ; BREAK BYTE INTO
FE09 2A                 ROL  A           ; 5 BIT GROUPS
FE0A D0 0E              BNE  SHFTCNT     ;BRANCH IF MORE BITS IN THIS BYTE
FE0C 84 A2              STY  CTEMP
FE0E C6 A4              DEC  YTEMP       ;(NOTE: CARRY IS SET)
FE10 F0 16              BEQ  DONE        ;BRANCH IF ALL DONE
FE12 A4 A4              LDY  YTEMP       ;GET CHARACTER TABLE INDEX
FE14 B9 [illegible]     LDA  CHRSET-1,Y
FE17 2A                 ROL  A           ;(CARRY KEEPS BYTE NON-ZERO UNTIL ALL E
FE18 A4 A2              LDY  CTEMP       ;RESTORE COLUMN COUNT        ARE SHIFTED)
FE1A 88        SHFTCNT  DEY              ;GOT ALL FIVE BITS?
FE1B D0 EA              BNE  CSHFT       ;NO, DO NEXT
FE1D CA                 DEX              ;ALL ROWS DONE?
FE1E 10 E5              BPL  CCOLMS      ;NO, DO NEXT
FE20 08                 PHP              ;SAVE REMAINING BIT PATTERN AND CARRY
FE21 48                 PHA
FE22 20 28 FE           JSR  STORCHRS    ;MOVE EM TO NON DISPLAYED VIDEO AREA
FE25 4C 01 FE           JMP  CBYTES
                   *
FE28           DONE     EQU  *
FE28 A2 1F     STORCHRS LDX  #$1F        ;MOVE CHARACTER PATTERNS TO VIDEO AREA
FE2A A0 00     STORSET  LDY  #0
FE2C B5 B4     STOROW   LDA  ROWTEMP,X
FE2E 0A                 ASL  A           ;SHIFT TO CENTER
FE2F 29 3E              AND  #$3E        ;STRIP EXTRA GARBAGE
FE31 91 A0              STA  (CPORTL),Y
FE33 CA                 DEX
FE34 C8                 INY
FE35 C0 08              CPY  #$8         ;THIS GROUP DONF
FE37 D0 F3              BNE  STOROW      ;NO, NEXT ROW
FE39 20 99 FE           JSR  NXTPORT
FE3C C9 08              CMP  #$8
FE3E F0 04              BEQ  GENDONE     ;ALL ROWS STORED?
FE40 8A                 TXA
FE41 10 E7              BPL  STORSET
FE43 60                 RTS              ;PARTIAL SET ($478-$5FF)
FE44               *
```

“APPLE_PAT_4_383_296_63” 118 KB 2000-02-27 dpi: 300h x 300v pix: 1836h x 2696v



```
FE44: A9 01      114 GENDONE LDA   #1          ;SET NORMAL MODE
FE46: 85 A2      115         STA   CTEMP
FE48: A9 60      116 GEN1    LDA   #$60        ;PREPARE TO SEND BYTES TO CHARACTER
FE4A  2C DB C0   117         BIT   CWRTON      ; GENERATOR RAM
FE4D  20 AE FE   118         JSR   VRETRCE     ;WAIT FOR NEXT VERTICAL RETRACE
FE50  A9 20      119         LDA   #$20        ;WAIT AGAIN
FE52: 20 AE FE   120         JSR   VRETRCE
FE55. 2C DA C0   121         BIT   CWRTOFF     ;CHARACTERS ARE NOW LOADED
FE58: 20 88 FE   122         JSR   ALTCHR      ;REPEAT THIS SET FOR OTHER 64 CHARACTERS
FE5B  C6 A2      123         DEC   CTEMP       ;HAVE WE DONE ALTERNATES YET?
FE5D  10 16      124         BPL   GEN2        ;NO, DO IT!
FE5F  A9 08      125         LDA   #$8         ,DUMP ASCII VALUES FOR NEXT SET
FE61  85 A1      126         STA   CPORTH
FE63  A0 07      127 NXTASCI LDY   #7          ;THE USUAL COUNDOWN
FE65  B1 A0      128 NXTASC2 LDA   (CPORTL),Y
FE67  18         129         CLC
FE68  69 08      130         ADC   #$8
FE6A  91 A0      131         STA   (CPORTL),Y
FE6C  88         132         DEY
FE6D  10 F6      133         BPL   NXTASC2
FE6F  20 99 FF   134         JSR   NXTPORT
FE72  90 EF      135         BCC   NXTASCI
FE74  60         136         RTS
FE75  A0 03      137 GEN2    LDY   #$3         ;SETUP ALTERNATE WITH UNDERLINES
FE77  A9 7F      138         LDA   #$7F
FE79  99 FC 05   139 UNDER   STA   $5FC,Y
FE7C  99 FC 07   140         STA   $7FC,Y
FE7F  88         141         DEY
FE80  10 F7      142         BPL   UNDER
FE82  A9 08      143         LDA   #$8
FE84  85 A1      144         STA   CPORTH
FE86: D0 C0      145         BNE   GEN1
FE88             146 *
FE88. A0 07      147 ALTCHR  LDY   #7          ;ADJUST ASCII FOR ALTERNATE SET
FE8A: B1 A0      148 ALTC1   LDA   (CPORTL),Y
FE8C. 49 20      149         EOR   #$20        ;$20-->0  $40-->$60
FE8E. 91 A0      150         STA   (CPORTL),Y
FE90: 88         151         DEY
FE91: 10 F7      152         BPL   ALTC1       ;ADJUST THEM ALL
FE93: 20 99 FE   153         JSR   NXTPORT
FE96: 90 F0      154         BCC   ALTCHR
FE98: 60         155         RTS
FE99:            156 *
FE99: A5 A0      157 NXTPORT LDA   CPORTL      ;CONVERT $78->$F8 OR $F8-$78
FE9B: 49 80      158         EOR   #$80
FE9D: 85 A0      159         STA   CPORTL
FE9F: 30 02      160         BMI   NOHIGH
FEA1: E6 A1      161         INC   CPORTH      ;IF =C THEN =4
FEA3: A5 A1      162 NOHIGH  LDA   CPORTH
FEA5: C9 0C      163         CMP   #$C
FEA7: D0 04      164         BNE   PORTDN
FEA9: A9 04      165         LDA   #$4
FEAB: 85 A1      166         STA   CPORTH
FEAD: 60         167 PORTDN  RTS
FEAE:            168 *
FEAE:            169 *
FEAE: 85 A3      170 VRETRCE STA   CTEMP1      ;SAVE BITS TO BE STORED
FEB0: AD EC FF   171         LDA   CB2CTRL     ;CONTROL PORT FOR 'CB2'
FEB3: 29 3F      172         AND   #$3F        ;RESET HI BITS TO 0
FEB5: 05 A3      173         ORA   CTEMP1
FEB7: 8D EC FF   174         STA   CB2CTRL
FEBA: A9 08      175         LDA   #$8         ;TEST VERTICAL RETRACE
FEBC: 8D ED FF   176         STA   CB2INT
FEBF: 2C ED FF   177 VWAIT   BIT   CB2INT      ;WAIT FOR RETRACE
FEC2: F0 FB      178         BEQ   VWAIT
FEC4: 60         179         RTS
FEC5:            180 *
FEC5:            181 CHRSET  EQU   *
```

"APPLE_PAT_4_383_296_64" 144 KB 2000-02-27 dpi: 300h x 300v pix: 1848h x 2695v

```
FEC5 F0 01 82   182          DFB  $F0,$01,$82,$18
FEC8:18
FEC9:40 84 81   183          DFB  $40,$84,$81,$2F
FECC:2F
FECD:58 44 81   184          DFB  $58,$44,$81,$29
FED0:29
FED1:02 1E 01   185          DFB  $02,$1E,$01,$91
FED4:91
FED5:7C 1F 49   186          DFB  $7C,$1F,$49,$30
FED8:30
FED9:8A 08 43   187          DFB  $8A,$08,$43,$14
FEDC:14
FEDD:31 2A 22   188          DFB  $31,$2A,$22,$13
FEE0:13
FEE1:E3 F7 C4   189          DFB  $E3,$F7,$C4,$91
FEE4:91
FEE5:48 A2 DA   190          DFB  $48,$A2,$DA,$24
FEE8:24
FEE9:C6 4A 62   191          DFB  $C6,$4A,$62,$8C
FEEC:8C
FEED:24 C6 F8   192          DFB  $24,$C6,$F8,$63
FEF0:63
FEF1 8C C1 46   193          DFB  $8C,$C1,$46,$17
FEF4 17
FEF5 52 8A AF   194          DFB  $52,$8A,$AF,$16
FEF8 16
FEF9 14 E3 33   195          DFB  $14,$E3,$33,$31
FEFC 31
FEFD C6 F8 DC   196          DFB  $C6,$F8,$DC,$73
FF00 73
FF01 3F 46 17   197          DFB  $3F,$46,$17,$62
FF04 62
FF05 8C 21 E6   198          DFB  $8C,$21,$E6,$18
FF08 18
FF09 6A 8D 61   199          DFB  $6A,$8D,$61,$CF
FF0C CF
FF0D 18 62 74   200          DFB  $18,$62,$74,$D1
FF10 D1
FF11 B9 18 49   201          DFB  $B9,$18,$49,$4C
FF14 4C
FF15 91 C0 F3   202          DFB  $91,$C0,$F3,$09
FF18 09
FF19 2C 91 C0   203          DFB  $2C,$91,$C0,$14
FF1C 14
FF1D 1D 8C EF   204          DFB  $1D,$8C,$EF,$07
FF20 07
FF21 17 43 88   205          DFB  $17,$43,$88,$31
FF24 31
FF25 84 1E DF   206          DFB  $84,$1E,$DF,$0B
FF28 0B
FF29 31 84 F8   207          DFB  $31,$84,$F8,$FE
FF2C FE
FF2D 77 3E 3E   208          DFB  $77,$3E,$3E,$17
FF30 17
FF31 62 8C FD   209          DFB  $62,$8C,$FD,$C7
FF34 C7
FF35 50 E3 0B   210          DFB  $50,$E3,$0B,$51
```

"APPLE_PAT_4_383_296_65" 111 KB 2000-02-27 dpi: 300h x 300v pix: 1459h x 2664v

```
FF38 51
FF39 C5 E8 C8   211            DFB   $C5, $E8, $C8, $73
FF3C 73
FF3D 18 0C 42   212            DFB   $18, $0C, $42, $3E
FF40 3E
FF41 01 02 20   213            DFB   $01, $02, $20, $42
FF44 42
FF45 3E 41 18   214            DFB   $3E, $41, $18, $8C
FF48 8C
FF49 08 00 70   215            DFB   $08, $00, $70, $EE
FF4C EE
FF4D 00 11 11   216            DFB   $00, $11, $11, $21
FF50 21
FF51 11 02 E0   217            DFB   $11, $02, $E0, $30
FF54 30
FF55 21 31 02   218            DFB   $21, $31, $02, $E0
FF58 E0
FF59 1C 00 C8   219            DFB   $1C, $00, $C8, $B9
FF5C B9
FF5D 80 62 14   220            DFB   $80, $62, $14, $1F
FF60 1F
FF61 46 A2 DE   221            DFB   $46, $A2, $DE, $43
FF64 43
FF65 2C 04 88   222            DFB   $2C, $04, $88, $BE
FF68 BE
FF69 FF CE 7D   223            DFB   $FF, $CE, $7D, $37
FF6C 37
FF6D 49 88 95   224            DFB   $49, $88, $95, $18
FF70 18
FF71 98 09 62   225            DFB   $98, $09, $62, $D1
FF74 D1
FF75 44 E8 88   226            DFB   $44, $E8, $88, $FB
FF78 FB
FF79 02 90 40   227            DFB   $02, $90, $40, $00
FF7C 00
FF7D 10 E0 03   228            DFB   $10, $E0, $03, $02
FF80 02
FF81 00 40 00   229            DFB   $00, $40, $00, $00
FF84 00
FF85 08 00 00   230            DFB   $08, $00, $00, $28
FF88 28
FF89 10 42 44   231            DFB   $10, $42, $44, $25
FF8C 25
FF8D 82 88 2F   232            DFB   $82, $88, $2F, $48
FF90 48
FF91 25 44 10   233            DFB   $25, $44, $10, $82
FF94 82
FF95 02 00 2F   234            DFB   $02, $00, $2F, $5A
FF98 5A
FF99 40 45 02   235            DFB   $40, $45, $02, $8E
FF9C 8E
FF9D 64 50 90   236            DFB   $64, $50, $90, $01
FFA0 01
FFA1 3E 26 42   237            DFB   $3E, $26, $42, $80
FFA4 80
FFA5 21 80 00   238            DFB   $21, $80, $00, $05
FFA8 05
FFA9 00 F8 80   239            DFB   $00, $F8, $80, $00
FFAC 00
FFAD 05 08 F8   240            DFB   $05, $08, $F8, $80
FFB0 80
FFB1 28 05 88   241            DFB   $28, $05, $88
FFB4            242 *
FFB4            243 HOOKS      EQU   *
FFB4 F2 FB      244            DW    COUT2
FFB6 0F FD      245            DW    KEYIN
FFB8            246 *
```

"APPLE_PAT_4_383_296_66" 104 KB 2000-02-27 dpi: 300h x 300v pix: 1477h x 2702v

```
FFB8               247 VBOUNDS EQU  *
FFB8 00 50 00      248         DFB  $0,$50,0,$18
FFBB 18
FFBC               249 *
FFBC 4C 89 F6      250 NMIRQ   JMP  RECON        ;IN DIAGNOSTICS
FFBF 40            251         RTI
FFC0 C3 CF D0      252         ASC  'COPYRIGHT JANUARY, 1980  APPLE COMPUTER INC..JRH'
FFC3 D9 D2 C9
FFC6 C7 C8 D4
FFC9 A0 CA C1
FFCC CE D5 C1
FFCF D2 D9 AC
FFD2 A0 B1 B9
FFD5 B8 B0 A0
FFD8 A0 C1 D0
FFDB D0 CC C5
FFDE A0 C3 CF
FFE1 CD D0 D5
FFE4 D4 C5 D2
FFE7 A0 C9 CE
FFEA C3 AE AE
FFED CA D2 C8
FFF0               253         CHN  MONVECT
FFF0                 1 *
FFF0 CC              2 ESCTABL DFB  $CC
FFF1 D0              3         DFB  $D0
FFF2 D3              4         DFB  $D3
FFF3 B4              5         DFB  $B4
FFF4 B8              6         DFB  $B8
FFF5 88              7         DFB  $88
FFF6 95              8         DFB  $95
FFF7 8A              9         DFB  $8A
FFF8 8B             10         DFB  $8B
FFF9 00             11         DFB  $00          ;NOTHING
FFFA                12 *
FFFA CA FF          13 NMI     DW   $FFCA
FFFC EE F4          14 RESET   DW   DIAGN        ;FIRST DIAGNOSTICS
FFFE CD FF          15 IRQ     DW   $FFCD
0000                16 *

*** SUCCESSFUL ASSEMBLY: NO ERRORS
   75 A1H           74 A1L         F9D4 A1PC          F9D7 A1PC1
   77 A2H           76 A2L           79 A3H             78 A3L
   7B A4H           7A A4L           A0 ADR           FE8A ALTC1
 FE88 ALTCHR      FB2C ASC1        FB38 ASC2          FB46 ASC3
 FA06 ASCDONE     FA09 ASCII0      F9DF ASCII1        F9E1 ASCII2
 FA07 ASCII       F9F2 ASCII3        5F BAS4H           5E BAS4L
   61 BAS8H         60 BAS8L       FC05 BASCALC1     ?FBB3 BASCALC
 FC3D BELL1       FC3E BELL2       FC43 BELL3         FC3A BELL
 FA15 BITOFF      FA11 BITON         67 BKGND         FCDE BKSPCE
 FAA0 BL1         F479 BLOCKIO     FC19 BSCLC2        FCCD CANCEL
?FB9B CARRAGE     FFEC CB2CTRL     FFED CB2INT        FE01 CBYTES
 FE05 CCOLMS        5C CH          FEC5 CHRSET       ?FA0A CKMDE
?FD98 CLDSTRT     FC89 CLEOL1      FB8E CLEOL         FC91 CLEOL2
 FB71 CLEOP       FB7A CLEOP1      FB69 CLSCRN        F91C CMDSRCH
 F96C CMDTAB      F97C CMDVEC      FB4F COL40         FB49 COL80
 FB93 CONTROL     FC33 COUT1       FBF2 COUT2         FC25 COUT
   A1 CPORTH        A0 CPORTL      F9FB CRCHK         FA26 CRMON
 FCEF CROUT       FE07 CSHFT     ?   6F CSWH            6E CSWL
   A3 CTEMP1        A2 CTEMP       FC24 CTRLRET       FBB3 CURDN1
 FBC9 CURDOWN     FBB7 CURIGHT     FBD9 CURLEFT         69 CURSOR
```

J. Richard (Dick)
Huston

(also worked on
Apple /// SOS)

```
 FBAE CURUP1        FBA4 CURUP          5D CV            CODA CWRTOFF
 CODB CWRTON        FA91 DEST         F4EE DIAGN         F941 DIGIT
 F96B DIGRET        FBFC DISPLAYX     FC9D DISPLAY       FE28 DONE
 FCAA DSPBKGND      FCAD DSPL80       FAB7 DUMMY         FAFC DUMPO
 FB09 DUMP1         FB0C DUMP2        FB1C DUMP3         FAE9 DUMP8
 FB21 DUMPASC      ?FAF9 DUMP        ?F901 ENTRY         FA8B ERROR2
 FABE ERROR         FAF7 ERROR1      ?FD53 ESC1          FD58 ESC2
 FD48 ESC3          FD4B ESCAPE       FFF0 ESCTABL       FD7F ESCVECT
   66 FORGND        FDE3 GASCI1       FDE5 GASCI2        FDE7 GASCI3
 FDF4 GASCI4        FE48 GEN1         FE75 GEN2          FDE1 GENASC
 FE44 GENDONE      ?FDC6 GENENTR      FCD5 GETLN         FCD5 GETLNZ
 F92C GETNUM        FD77 GOESC        FA7D GO            FFB4 HOOKS
   85 IDBUFP          87 IBCMD          82 IBDRVN          81 IBSLOT
   50 INBUFLEN        7E INBUF       ?FBFF INCHORZ      ?FFFE IRQ
 FA7D JUMP          C010 KBDSTRB      C000 KBD           FD16 KEYIN1
 FD24 KEYIN2        FD2E KEYIN3      ?FD31 KEYIN4        FD0F KEYIN
 FD47 KEYRET        FD35 KEYWAIT     ?  71 KSWH            70 KSWL
 FD42 KWAIT2        FC87 LASTLN       FBDF LEFT80        FBE9 LEFTUP
   58 LMARGIN       FC52 LNFD           69 MASK          FA52 MISMATCH
   68 MODES         F904 MON          F908 MONZ          FA2C MOVE
 FA31 MOVNXT        FFBC NMIRG       ?FFFA NMI           FEA3 NOHIGH
 FD07 NOSTOP        FCB8 NOTCR        FADF NOVER         F992 NXTA1
 F98C NXTA4         FE65 NXTASC2      FE63 NXTASCI       F94F NXTBAS
 F947 NXTBIT        F959 NXTBS2       FCE4 NXTCHAR       F932 NXTCHR
 F915 NXTINP        FC02 NXTLIN       FE99 NXTPORT       F9DE OLDPC
?  73 PCH             72 PCL          FD95 PICK40        FD88 PICK
 FEAD PORTDN        FA6E PRA1BYTE     F9C2 PRBYCOL       F9AC PRBYTE
 FA70 PRBYTSP      ?F9C5 PRCOLON      F9BF PRHEX2        F9B7 PRHEXZ
?F9B5 PRHEX         FA61 PRINTA1        6B PROMPT        FA73 PRSPC
 FD60 RDCHAR        FD0C RDKEY        FAC0 READ          F689 RECON
 FA19 REPEAT        FA21 REPEAT1     ?FFFC RESET         F7FF RET1
 F900 RET2          F882 RET3         F9AB RETA1         FBBD RIGHT1
   59 RMARGIN         B4 ROWTEMP      FA83 RWERROR       FAC7 RWLOOP
?FAC5 SAVCMD        F912 SCAN         FC61 SCRL1         FC63 SCRL2
 FC7A SCRL3           58 SCRNLOC      FC5B SCROLL        FA9A SEP
 FB5B SET80A        FB53 SET80        FB67 SET80B        FBC3 SETCHZ
 FBB1 SETCV        ?FBC7 SETCVH       FADD SETMDZ        FAB8 SETMODE
?FD9D SETUP         FDA2 SETUP1       FE1A SHFTCNT       FAA4 SPCE
   6A STACK           7C STATE        FD02 STOPLST       FAAF STOR1
 FE28 STORCHRS      FE2C STOROW       FE2A STORSET      ?FAAB STOR
 F9D1 SVMASK          63 TBAS4H         62 TBAS4L       ?  65 TBAS8H
   64 TBAS8L          6C TEMPX          80 TEMP            6D TEMPY
 F95E TOSUB         F9C9 TST80WID     F99B TSTA1         FBD5 TSTBACK
 FC36 TSTBELL      ?FB97 TSTCR        FAF6 TSTDUMP       FE79 UNDER
 03F8 USERADR       FA78 USER         FFB8 VBOUNDS       FEAE VRETRCE
 FA4C VRFY2         FA3D VRFY         FA40 VRFY1         FEBF VWAIT
   5B WINBTM          5A WINTOP       FAC3 WRTE            7D YSAV
   A4 YTEMP         FDD5 ZIPTEMPS     F967 ZSTATE
   50 INBUFLEN        58 SCRNLOC        58 LMARGIN         59 RMARGIN
   5A WINTOP          5B WINBTM         5C CH              5D CV
   5E BAS4L           5F BAS4H          60 BAS8L           61 BAS8H
   62 TBAS4L          63 TBAS4H         64 TBAS8L       ?  65 TBAS8H
   66 FORGND          67 BKGND          68 MODES           69 MASK
   69 CURSOR          6A STACK          6B PROMPT          6C TEMPX
   6D TEMPY           6E CSWL        ?  6F CSWH            70 KSWL
?  71 KSWH            72 PCL         ?  73 PCH             74 A1L
   75 A1H             76 A2L            77 A2H             78 A3L
   79 A3H             7A A4L            7B A4H             7C STATE
   7D YSAV            7E INBUF          80 TEMP            81 IBSLOT
   82 IBDRVN          85 IBBUFP         87 IBCMD           A0 CPORTL
   A0 ADR             A1 CPORTH         A2 CTEMP           A3 CTEMP1
   A4 YTEMP           B4 ROWTEMP      03F8 USERADR       C000 KBD
 C010 KBDSTRB       CODA CWRTOFF      CODB CWRTON        F479 BLOCKIO
 F4EE DIAGN         F689 RECON        F7FF RET1          F882 RET3
 F900 RET2         ?F901 ENTRY        F904 MON           F908 MONZ
 F912 SCAN          F915 NXTINP        F91C CMDSRCH       F92C GETNUM
 F932 NXTCHR        F941 DIGIT         F947 NXTBIT        F94F NXTBAS
 F959 NXTBS2        F95E TOSUB         F967 ZSTATE        F96B DIGRET
 F96C CMDTAB        F97C CMDVEC        F98C NXTA4         F992 NXTA1
 F99B TSTA1         F9AB RETA1         F9AC PRBYTE       ?F9B5 PRHEX
 F9B7 PRHEXZ        F9BF PRHEX2        F9C2 PRBYCOL      ?F9C5 PRCOLON
 F9C9 TST80WID      F9D1 SVMASK        F9D4 A1PC          F9D7 A1PC1
 F9DE OLDPC         F9DF ASCII1        F9E1 ASCII2        F9F2 ASCII3
 F9FB CRCHK         FA06 ASCDONE       FA07 ASCII         FA09 ASCII0
?FA0A CKMDE         FA11 BITON         FA15 BITOFF        FA19 REPEAT
 FA21 REPEAT1       FA26 CRMON         FA2C MOVE          FA31 MOVNXT
 FA3B VRFY          FA40 VRFY1         FA4C VRFY2         FA52 MISMATCH
 FA61 PRINTA1       FA6E PRA1BYTE      FA70 PRBYTSP       FA73 PRSPC
 FA78 USER          FA7B JUMP          FA7D GO            FA83 RWERROR
 FA8B ERROR2        FA8E ERROR         FA91 DEST          FA9A SEP
 FAA0 BL1           FAA4 SPCE         ?FAAB STOR          FAAF STOR1
 FAB7 DUMMY         FAB8 SETMODE       FABD SETMDZ        FAC0 READ
 FAC3 WRTE         ?FAC5 SAVCMD        FAC7 RWLOOP        FADF NOVER
 FAE9 DUMP8         FAF6 TSTDUMP       FAF7 ERROR1       ?FAF9 DUMP
 FAFC DUMP0         FB09 DUMP1         FB0C DUMP2         FB1C DUMP3
 FB21 DUMPASC       FB2C ASC1          FB38 ASC2          FB46 ASC3
 FB49 COL80         FB4F COL40         FB53 SET80         FB5B SET80A
 FB67 SET80B        FB69 CLSCRN        FB71 CLEOP         FB7A CLEOP1
 FB8E CLEOL         FB93 CONTROL      ?FB97 TSTCR        ?FB9B CARRAGE
 FBA4 CURUP         FBAE CURUP1        FBB1 SETCV         FBB3 CURDN1
?FBB3 BASCALC       FBB7 CURIGHT       FBBD RIGHT1        FBC3 SETCH2
?FBC7 SETCVH        FBC9 CURDOWN       FBD5 TSTBACK       FBD9 CURLEFT
 FBDF LEFT80        FBE9 LEFTUP        FBF2 COUT2         FBFC DISPLAYX
?FBFF INCHORZ       FC02 NXTLIN        FC05 BASCALC1      FC19 BSCLC2
 FC24 CTRLRET       FC25 COUT          FC33 COUT1         FC36 TSTBELL
 FC3A BELL          FC3D BELL1         FC3E BELL2         FC43 BELL3
 FC52 LNFD          FC5B SCROLL        FC61 SCRL1         FC63 SCRL2
 FC7A SCRL3         FC87 LASTLN        FC89 CLEOL1        FC91 CLEOL2
 FC9D DISPLAY       FCAA DSPBKGND      FCAD DSPL80        FCB8 NOTCR
 FCCD CANCEL        FCD5 GETLN         FCD5 GETLNZ        FCDE BKSPCE
 FCE4 NXTCHAR       FCEF CROUT         FD02 STOPLST       FD07 NOSTOP
 FD0C RDKEY         FD0F KEYIN         FD16 KEYIN1        FD24 KEYIN2
 FD2E KEYIN3       ?FD31 KEYIN4        FD35 KEYWAIT       FD42 KWAIT2
 FD47 KEYRET        FD48 ESC3          FD4B ESCAPE       ?FD53 ESC1
 FD58 ESC2          FD60 RDCHAR        FD77 GOESC         FD7F ESCNEXT
 FD89 PICK          FD95 PICK40       ?FD98 CLDSTRT      ?FD90 SETUP
 FDA2 SETUP1       ?FDC6 GENENTR       FDD5 ZIPTEMPS      FDE1 GENASC
 FDE3 GASCI1        FDE5 GASCI2        FDE7 GASCI3        FDF4 GASCI4
 FE01 CBYTES        FE05 CCOLMS        FE07 CSHFT         FE1A SHFTCNT
 FE28 DONE          FE28 STORCHRS      FE2A STORSET       FE2C STOROW
 FE44 GENDONE       FE48 GEN1          FE63 NXTASCI       FE65 NXTASC2
 FE75 GEN2          FE79 UNDER         FE88 ALTCHR        FE8A ALTC1
 FE99 NXTPORT       FEA3 NOHIGH        FEAD PORTDN        FEAE VRETRCE
 FEBF VWAIT         FEC5 CHRSET        FFB4 HOOKS         FFB8 VBOUNDS
 FFBC NMIRG         FFEC CB2CTRL       FFED CB2INT        FFF0 ESCTABL
?FFFA NMI          ?FFFC RESET        ?FFFE IRQ
```

I claim:

1. In a digital computer which includes a central processing unit (CPU), a random-access memory (RAM), an address bus interconnecting said CPU and RAM such that said CPU addresses locations in said RAM and a data bus interconnecting said CPU and RAM, said CPU for certain functions addressing predetermined locations in said RAM with a predetermined range of address signals, an improvement comprising:

detection means for detecting said predetermined range of address signals, coupled to said address bus;

register means for storing digital signals, coupled to said data bus, and;

switching means for coupling said digital signals stored in said register means to said address bus when said detection means detects said predetermined range of said address signals;

whereby data for said certain functions normally stored by said CPU in said predetermined locations may be stored elsewhere in said RAM, thereby enhancing the performance of said computer.

2. The improvement defined by claim 2 wherein said detection means detects all binary zeros.

3. The improvement defined by claim 1 wherein said switching means comprises a multiplexer controlled by said detection means for selecting said register means.

4. The improvement defined by claim 1 including a read-only memory coupled to said address bus and said data bus.

5. The improvement defined by claim 4 wherein said stored signals in said register means provide a pointer for locations in said RAM during a direct memory access transfer.

6. The improvement defined by claim 5 wherein said read-only memory in response to signals on said address bus provides instructions to said CPU causing it to increment address signals during said direct memory access transfer.

7. In a digital computer which includes a central processing unit (CPU), a random-access memory (RAM), an address bus having a first plurality and a second plurality of lines for coupling said CPU with said RAM, and a data bus interconnecting said CPU and RAM, said CPU for certain operations addressing predetermined locations in said RAM with address signals on said first plurality of lines by coupling a predetermined address on said second plurality of lines, an improvement comprising:

register means for storing signals, coupled to said data bus;

multiplexing means coupled to said second plurality of lines and said register means for selecting signals from one of said second plurality of lines and said register means;

logic means coupled to said second plurality of lines and said multiplexing means for causing said multiplexing means to select signals from said register means when said CPU couples said predetermined address on said second plurality of lines;

whereby said signals from said register means provide alternate locations in RAM for storage associated with said certain operations.

8. The improvement defined by claim 7 wherein said predetermined address is all binary zeros.

9. The improvement defined by claim 7 including a read-only memory coupled to said address bus and said data bus.

10. The improvement defined by claim 8 wherein said stored signal in said register means provides a pointer for locations in said RAM during a direct memory access transfer.

11. The improvement defined by claim 9 wherein said read-only memory in response to signals on said address bus provides instructions to said CPU causing it to increment address signals during said direct memory access transfer.

12. In a digital processor used in conjunction with a display, said processor including a data bus and an address bus, a memory comprising:

a first plurality of memory devices for storing data, coupled to receive data from said data bus;

a first memory output bus coupled to receive data from said first plurality of memory device;

a second plurality of memory devices for storing data coupled to receive data from said data bus;

a second memory output bus coupled to receive data from said second plurality of memory devices;

addressing means coupled to said address bus for providing address signal for addressing said first and second plurality of memory devices;

first switching means for selecting data from one of said first and second memory buses for coupling to said data bus, said first switching means coupled to said first and second memory bus and said data bus;

second switching means for selecting data from said first and second memory buses for coupling to said display, said second switching means coupled to said first and second memory buses and said display; and,

circuit means for coupling one of a selected said first and second memory buses to said addressing means such that data from said selected one of said buses provides addressing information for selecting subsequent locations in said memory devices when said data bus is receiving data from the other of said memory buses,

whereby said memory provides data for a high resolution display and whereby some data stored in said memory is used for remapping locations in said memory.

13. The memory defined by claim 12 wherein said circuit means comprises a multiplexer, said multiplexer selecting between said data from said selected one of said buses and bank switching signals coupled to said multiplexer.

14. The memory defined by claim 13 wherein said multiplexer is controlled by a logic circuit which is coupled to said address bus and said selected one of said buses.

15. The memory defined by claim 14 wherein said logic circuit causes said multiplexer to select said bank switching signals each time said processor switches an OP code.

16. In a digital computer with a memory, which is used in conjunction with a raster scanned display, said display including a digital counter which provides a vertical count representative of the horizontal line scanned by the beam for said display, said memory providing data for displaying rows of characters, an

addressing means coupled to said memory for scrolling displayed characters, comprising:

an adder having a first and a second input terminal, the output of said adder providing a portion of an address signal for said memory, said first terminal of said adder being coupled to receive the lesser significant bits of said vertical count;

said computer providing a periodically repeated sequence of digital numbers coupled to said second terminal of said adder, said sequence of digital numbers provided by said computer having a maximum value equal to the number of scanned lines in each of said rows,

whereby the characters on said display are scrolled with a minimum of movement of data within said memory.

17. The addressing means defined by claim **16** wherein said sequence of digital numbers is incremented for each displayed frame.

18. In a ditital computer which includes a single chip central processing unit (CPU), a random-access memory (RAM), an address bus interconnecting said CPU and RAM such that said CPU addresses locations in said RAM, and a data bus coupled to said CPU and RAM, said CPU for certain functions addressing the zero page in said RAM by providing binary zeroes on certain lines of said address bus; an improvement comprising:

a detection circuit for detecting said binary zeroes on said certain lines of said address bus;

a register for storing digital signals, said register coupled to said data bus for receiving digital signals from said data bus; and,

a multiplexer for selecting between said digital signals stored in said register and said certain lines of said address bus, said multiplexer being controlled by said detection circuit so as to select said register when said binary zeroes are detected on said certain lines of said address bus;

whereby data for said certain functions normally stored on page one of said RAM, may be stored elsewhere in said RAM, and still easily addressed by said CPU.

19. The improvement defined by claim **18** wherein one of said stored signals from said register is coupled to said multiplexer through an exclusive OR gate, said gate being coupled to one of said certain lines of said address bus.

20. The improvement defined by claim **18** or **19** wherein said computer provides an alternate stack signal and wherein said detection circuit also detects addresses for page one on said address bus, and said multiplexer selects said register if said page one addresses are detected and said alternate stack signal is in a predetermined state.

21. In a digital computer which includes a central processing unit (CPU), a random-access memory (RAM), an address bus interconnecting said CPU and RAM such that said CPU addresses locations in said RAM and a data bus interconnecting said CPU and RAM, said CPU for certain functions addressing predetermined locations in said RAM with a predetermined range of address signals, an improvement comprising:

detection means for detecting said predetermined range of address signals, coupled to said address bus;

register means for storing digital signals, coupled to said data bus, and;

switching means for coupling said digital signals stored in said register means to said address bus when said detection means detects said predetermined range of said address signals, said switching means also for coupling said digital signals stored in said register means to said address bus when a certain direct memory access (DMA) signal is in a predetermined state;

a read-only memory (ROM) coupled between said address bus and said data bus, said ROM in response to signals on said address bus providing instructions to said CPU on said data bus to cause said CPU to increment address signals when said DMA signal is in said predetermined state;

said register providing a pointer for locations in said RAM when said DMA signal is in said predetermined state, and said register providing RAM address signals when said certain functions are selected by said CPU,

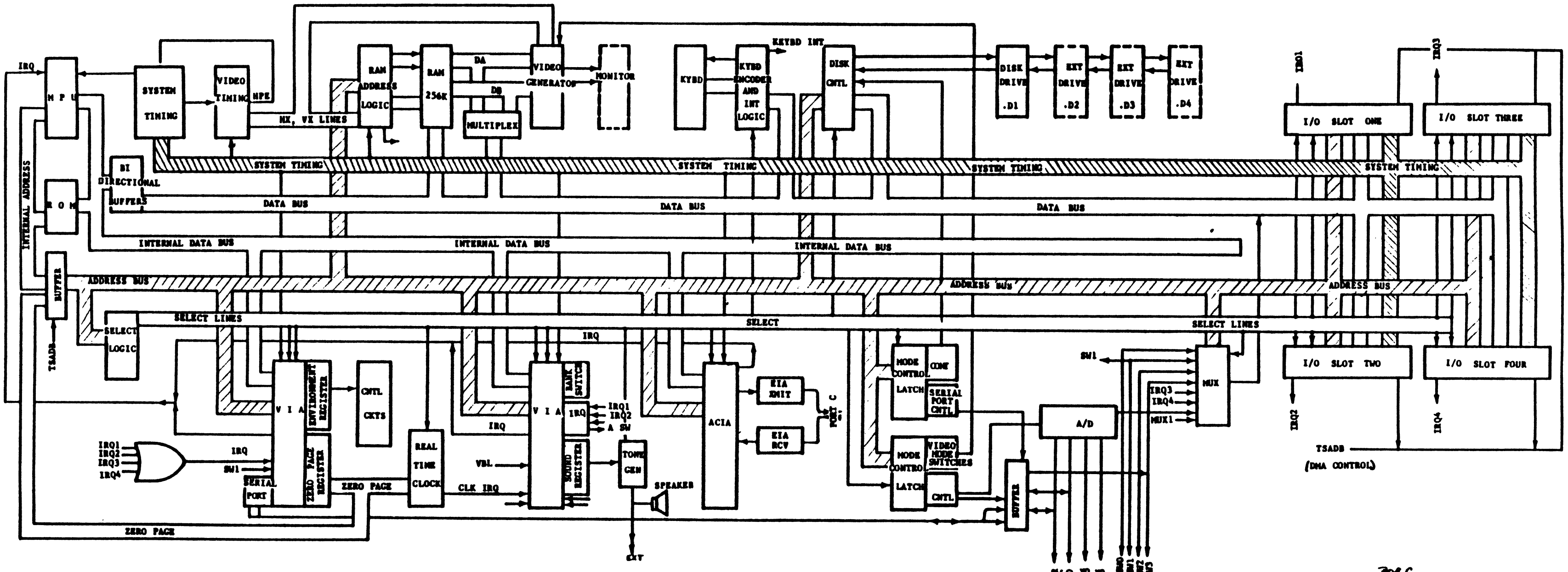
whereby data for said certain functions normally stored by said CPU in said predetermined locations may be stored elsewhere in said RAM, thereby enhancing the performance of said computer.

22. The improvement defined by claim **21** wherein said switching means comprise a multiplexer which selects said register when said detection means detects all binary zeroes or when said DMA signal is in said predetermined state.

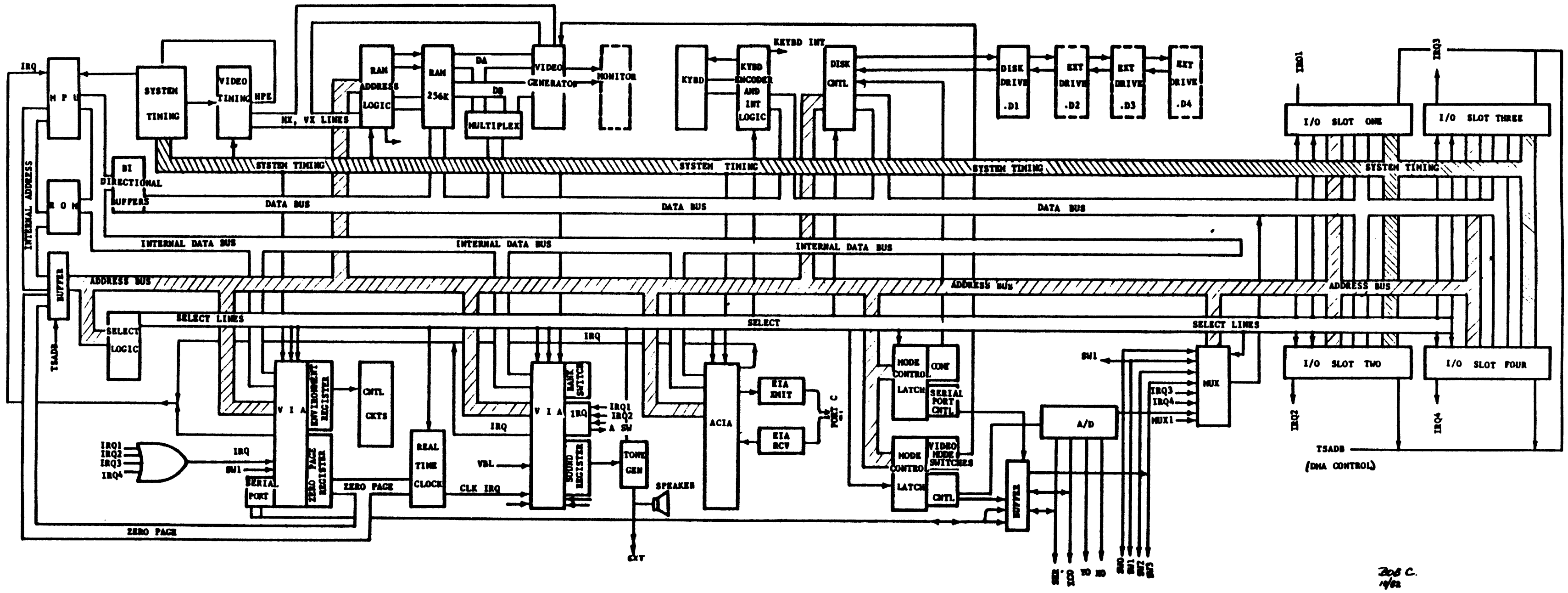
* * * * *

FINIS

Apple /// computer system diagram



Apple /// computer system diagram



“APPLE_PAT_4_383_296_A_02” 885 KB 2000-02-27 dpi: 800h x 800v pix: 4144h x 7776v

APPLE /// ROM INFORMATION

ROM REVISION #Ø

by
David Craig
736 Edgewater, Wichita, Kansas 67230
1986

This document describes the Apple /// microcomputer ROM organization. The ROM listing used was from Apple Computer's patent (# 4,383,296) of May 10, 1983 as assigned to Wendell B. Sander. The ROM listing appears to be from December 20, 1979.

The ROM occupies 4K bytes of memory in the address range $F000–$FFFF. This ROM is used by the Apple /// at system power-up to test various hardware components, initialize the character generator bitmap, and boot SOS (Sophisticated Operating System) from the Apple ///'s internal floppy diskette drive.

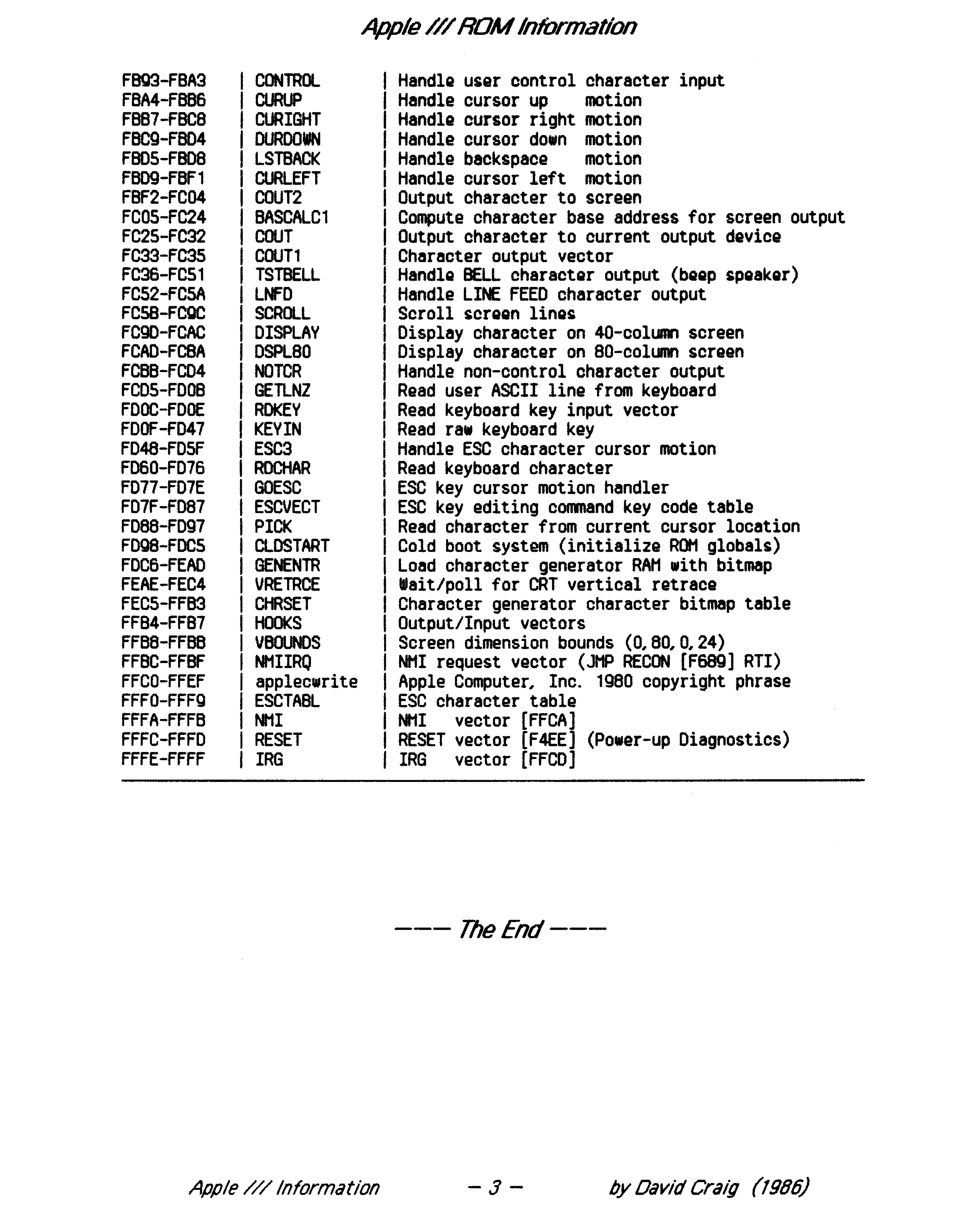
The ROM is organized as follows (routine names in lowercase were created by me since the source code did not contain a name at the particular location):

Addresses	Name	Description
F000-F124	REGRWTS	Read/Write a disk track and sector
F125-F12A	SETTRK	Set slot dependent track location
F12B-F13D	CHKDRV	Check if disk motor is stopped
F13E-F147	DRVINDX	Get index to drive number
F148-F1B9	READ16	Read disk sector
F1BA-F1BC	GOSERV	Interrupt service vector
F1BD-F218	RDADR16	Read disk sector address field
F219-F2B2	WRITE16	Write disk sector
F2B3-F2BB	SERVICE	Interrupt servicer
F2BC-F2C5	WNIBL9	Write 7-bit nibbles to disk
F2C6-F310	PRENIB16	Pre-nibblize disk sector data
F311-F354	POSTNIB16	Post-nibblize disk sector data
F355-F395	NIBL	6-bit to 7-bit nibble conversion table
F396-F3FF	DNIBL	7-bit to 6-bit denibbleize conversion table
F400-F455	SEEK	Disk track seeker
F456-F466	MSWAIT	100 microsecond delayer
F467-F46F	ONTABLE	Disk phase ON time table (in 100 microsecs)
F470-F478	OFFTABLE	Disk phase OFF time table (in 100 microsecs)
F479-F49F	BLOCKIO	Read/write a disk block (2 sectors)

Apple /// ROM Information

F4A0-F4A7	SECTABL	Block to sector conversion table
F4A8-F4C4	ANALOG	Joystick read routine
F4C5-F4CC	RAMTBL	RAM test bytes
F4CD-F4ED	CHPG	Hardware component phrases (eg "RAM", "ROM",...)
F4EE-F523	DIAGN	ROM system power-up entry (calls RECON [F689])
F524-F531	NXBYT	Test RAM page 0 (Zero Page)
F532-F545	CNTWR	Test RAM page 1 (Stack Page)
F546-F574	memsize	Size the RAM
F575-F5B9	ERRLP	Display screen error line ("DIAGNOSTICS")
F5BA-F5E6	zpgstktst	Test RAM zero page & stack page
F5E7-F60C	ROMTST	Test ROM hardware
F60D-F63D	VIATST	Test VIA hardware
F63E-F652	ACIA	Test ACIA hardware
F653-F67A	ATD	Test A/D hardware
F67B-F688	KEYPLUG	Test keyboard plugin
F689-F6C1	RECON	Reconfigure system (tests for Apple-1 key)
F6C2-F6E5	SEX	System exerciser
F5E6-F737	USRENTRY	Main RAM tester
F738-F747	STRWT	Error message string writer
F748-F77A	RAM	Determine size of RAM
F77B-F783	MESSERR	Display error message
F784-F7A0	RAMSET	Setup RAM
F7A1-F7C8	PTRINC	Increment extended addressing pointer
F7C9-F7F6	RAMERR	RAM error handler
F7F7-F7FF	RAMWT	RAM write
F800-F900	RET1	Nested RTS 'table' routine
F901-F92B	ENTRY	SARA Monitor entry point
F92C-F95D	GETNUM	Get number from user
F92E-F96B	TOSUB	Execute Monitor command
F96C-F97B	CMDTAB	Monitor command code table
F97C-F98B	CMDVEC	Monitor command vector table (byte-long entries)
F98C-F9AB	NXTA4	Increment 2 byte pointer
F9AC-F9C1	PRBYTE	Output a byte to screen
F9C2-F9C8	PRBYCOL	Output a byte followed by a colon
F9C9-F9D3	TST80WID	Test for 80-column screen width
F9D4-F9DE	A1PC	Test for new P.C.
F9DF-FA06	ASCII1	Store user ASCII string into memory
FA07-FA25	ASCII	Fetch ASCII character from keyboard
FA26-FA2B	CRMON	Dump line of hexadecimal bytes due to user CR
FA2C-FA3A	MOVE	Move bytes around in memory
FA3B-FA51	VRFY	Verify memory byte range
FA52-FA77	MISMATCH	Output verify mismatch data line
FA78-FA7A	USER	User control vector
FA7B-FA82	JUMP	Transfer control to user routine
FA83-FA90	RWERROR	Output error number
FA91-FA99	DEST	Copy source pointer to destination pointer
FA9A-FAB7	SEP	Test for seperator character in input line
FAB8-FABF	SETMODE	Setup user mode
FAC0-FAE8	READ	Handle Monitor READ disk block command
FAE9-FB20	DUMP8	Output line of memory bytes
FB21-FB48	DUMPASC	Output line of memory bytes as ASCII
FB49-FB4E	COL80	Setup 80-column display mode
FB4F-FB92	COL40	Setup 40-column display mode

ENTRY POINT

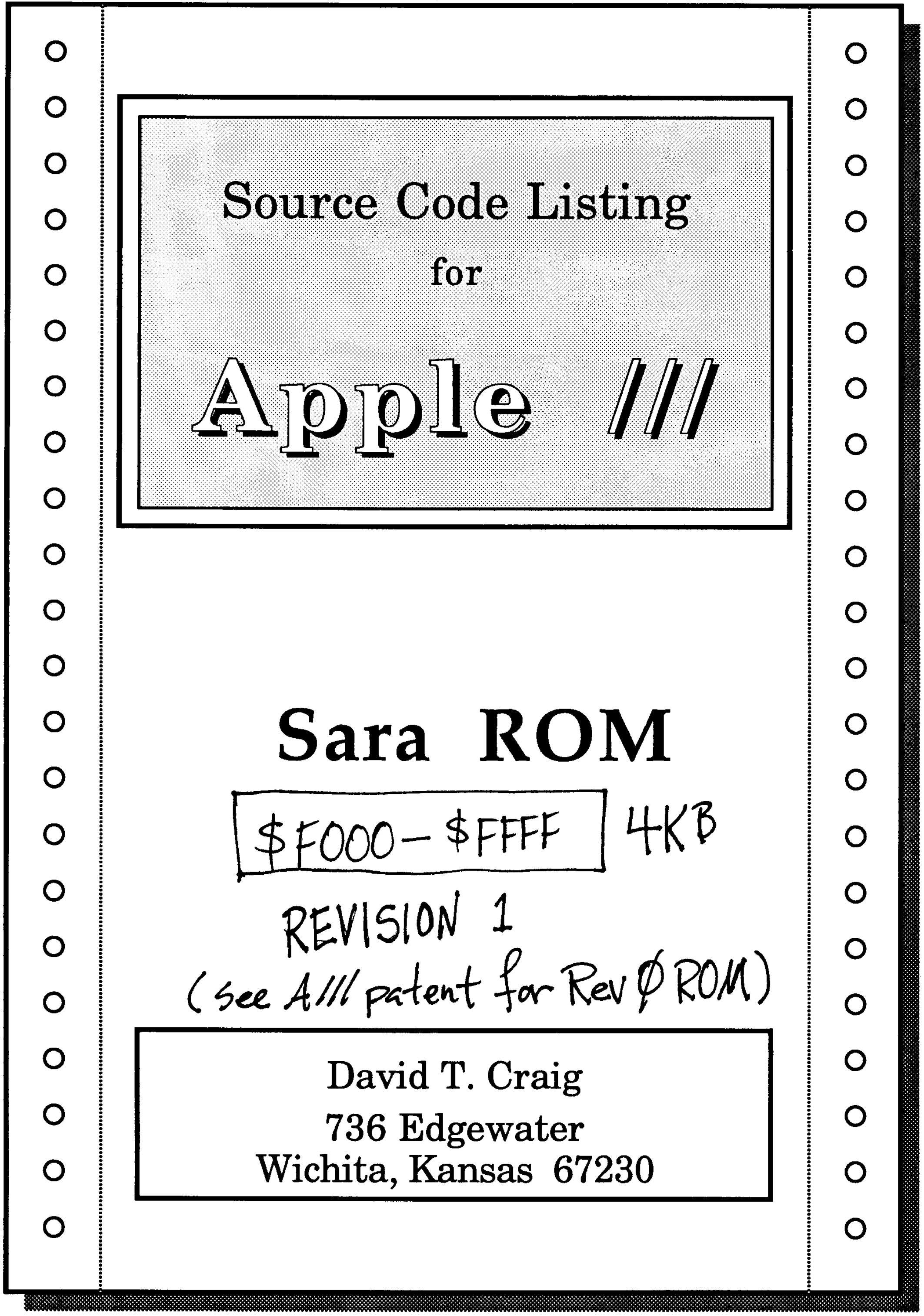


Apple /// ROM Information

FB93-FBA3	CONTROL	Handle user control character input
FBA4-FBB6	CURUP	Handle cursor up motion
FBB7-FBC8	CURIGHT	Handle cursor right motion
FBC9-FBD4	DURDOWN	Handle cursor down motion
FBD5-FBD8	LSTBACK	Handle backspace motion
FBD9-FBF1	CURLEFT	Handle cursor left motion
FBF2-FC04	COUT2	Output character to screen
FC05-FC24	BASCALC1	Compute character base address for screen output
FC25-FC32	COUT	Output character to current output device
FC33-FC35	COUT1	Character output vector
FC36-FC51	TSTBELL	Handle BELL character output (beep speaker)
FC52-FC5A	LNFD	Handle LINE FEED character output
FC5B-FC9C	SCROLL	Scroll screen lines
FC9D-FCAC	DISPLAY	Display character on 40-column screen
FCAD-FCBA	DSPL80	Display character on 80-column screen
FCBB-FCD4	NOTCR	Handle non-control character output
FCD5-FD0B	GETLNZ	Read user ASCII line from keyboard
FD0C-FD0E	RDKEY	Read keyboard key input vector
FD0F-FD47	KEYIN	Read raw keyboard key
FD48-FD5F	ESC3	Handle ESC character cursor motion
FD60-FD76	RDCHAR	Read keyboard character
FD77-FD7E	GOESC	ESC key cursor motion handler
FD7F-FD87	ESCVECT	ESC key editing command key code table
FD88-FD97	PICK	Read character from current cursor location
FD98-FDC5	CLDSTART	Cold boot system (initialize ROM globals)
FDC6-FEAD	GENENTR	Load character generator RAM with bitmap
FEAE-FEC4	VRETRCE	Wait/poll for CRT vertical retrace
FEC5-FFB3	CHRSET	Character generator character bitmap table
FFB4-FFB7	HOOKS	Output/Input vectors
FFB8-FFBB	VBOUNDS	Screen dimension bounds (0,80,0,24)
FFBC-FFBF	NMIIRQ	NMI request vector (JMP RECON [F689] RTI)
FFC0-FFEF	applecwrite	Apple Computer, Inc. 1980 copyright phrase
FFF0-FFF9	ESCTABL	ESC character table
FFFA-FFFB	NMI	NMI vector [FFCA]
FFFC-FFFD	RESET	RESET vector [F4EE] (Power-up Diagnostics)
FFFE-FFFF	IRG	IRG vector [FFCD]

---- The End ----

Assembler: Apple /// Pascal TLA 6502 Assembler 41 pages
(converted to TLA format most likely by Scott Stinson)



Source Code Listing for Apple ///

Sara ROM

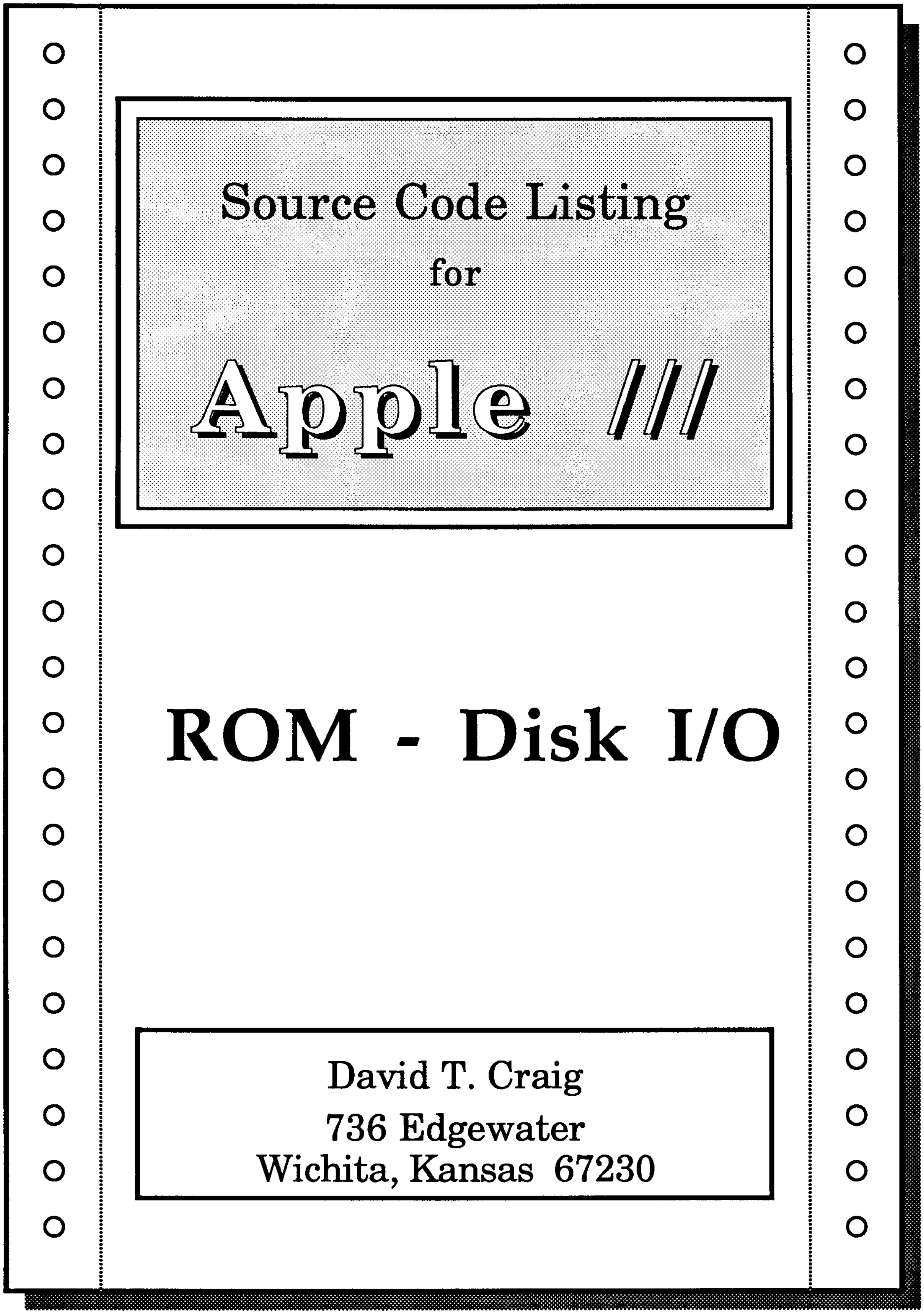
$F000-$FFFF 4KB

REVISION 1
(see A/// patent for Rev Ø ROM)

David T. Craig
736 Edgewater
Wichita, Kansas 67230

Copyrighted Jan. 1980

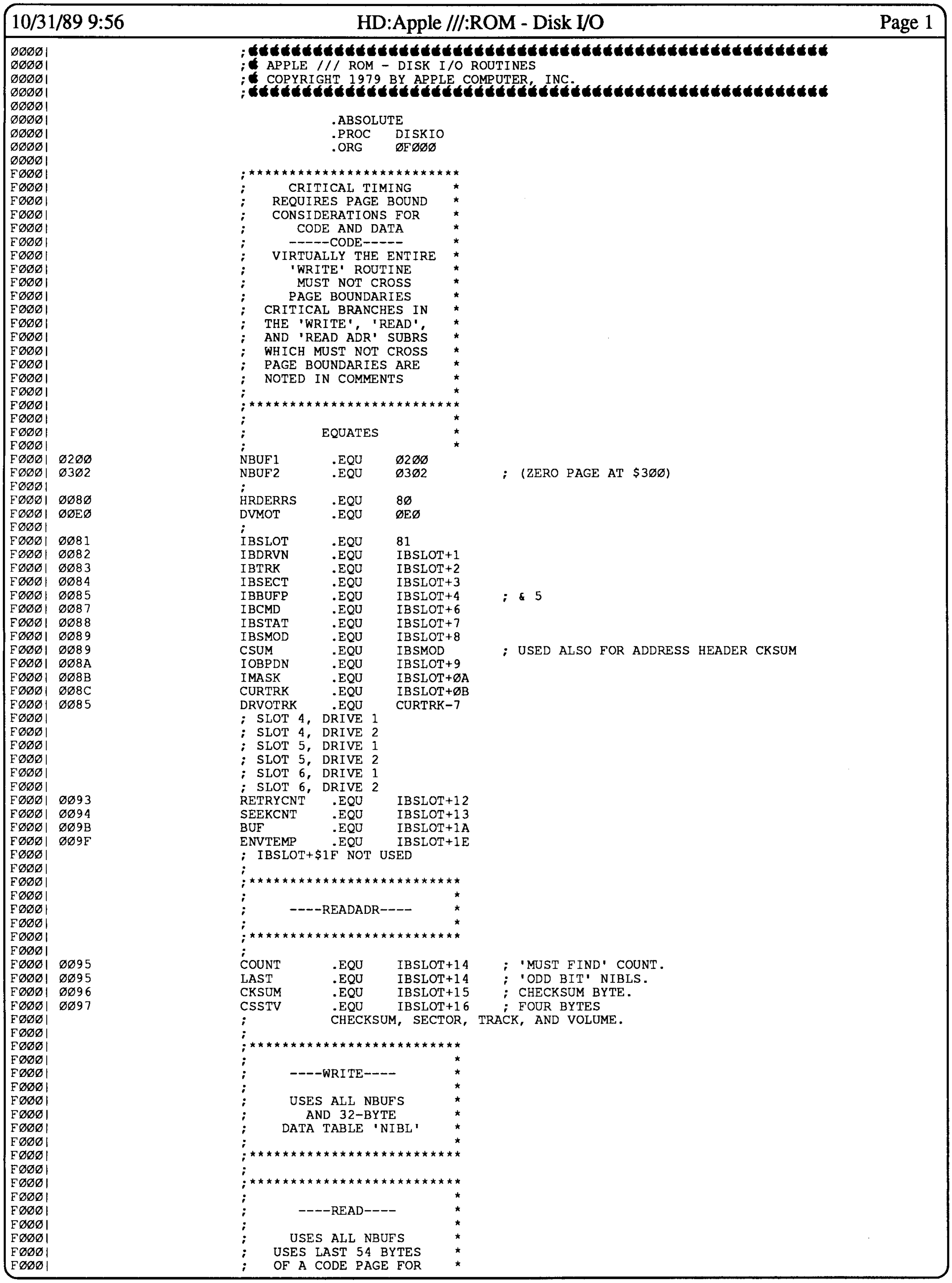
"APPLE_PAT_4_383_296_C_01" 214 KB 2000-02-27 dpi: 300h x 300v pix: 1934h x 3081v



# Source Code Listing
for
Apple ///

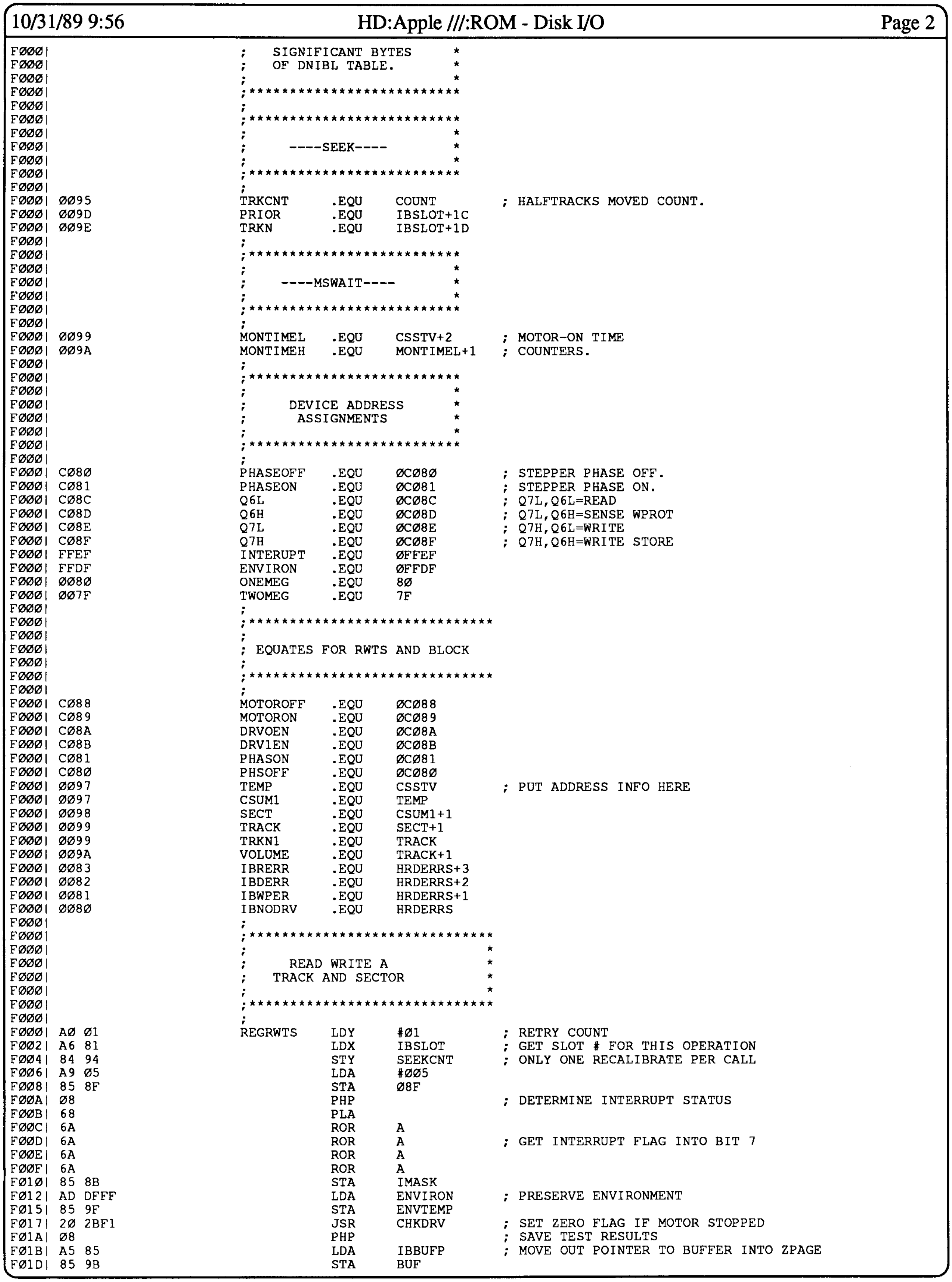
ROM - Disk I/O

David T. Craig
736 Edgewater
Wichita, Kansas 67230



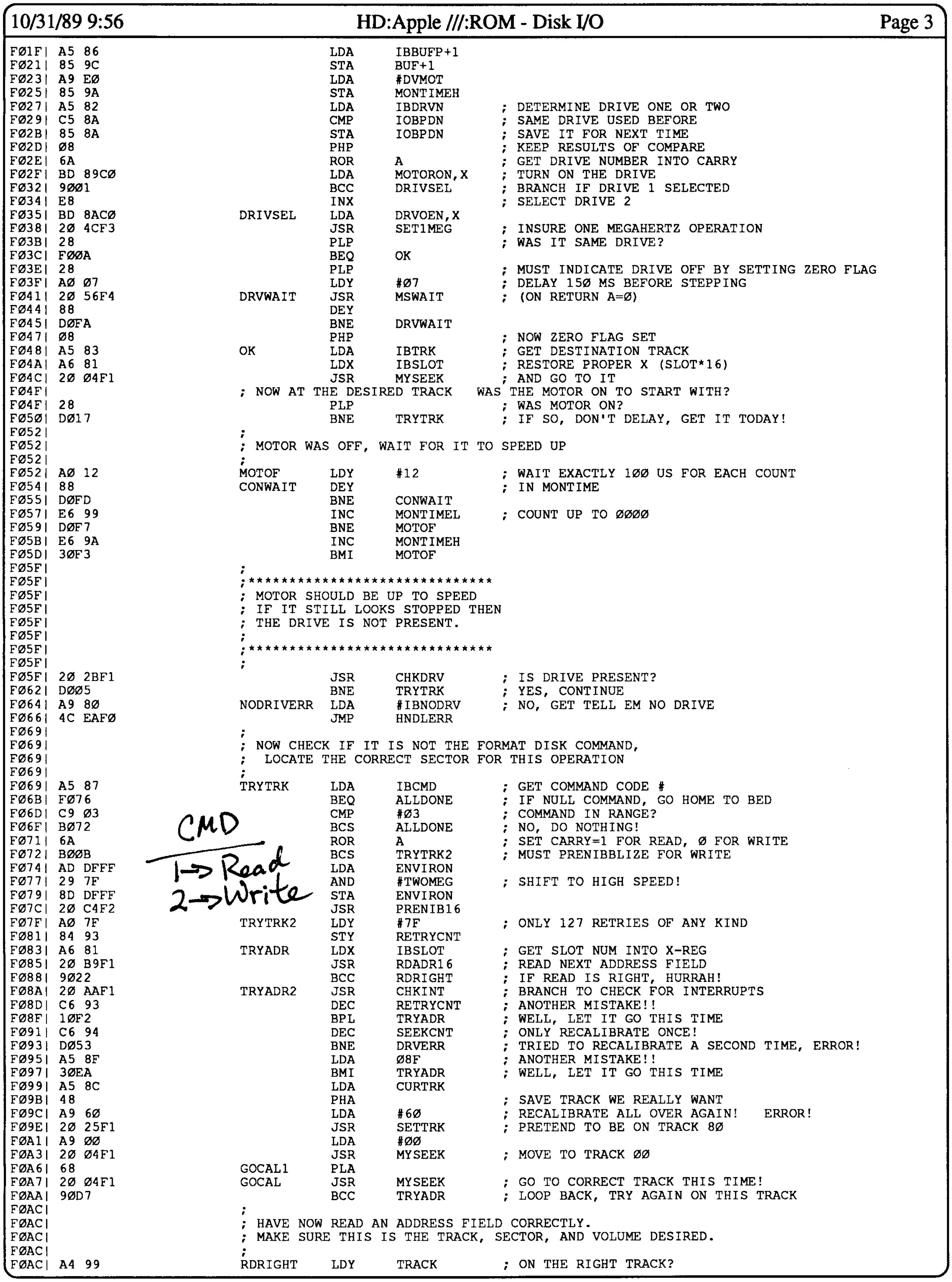
10/31/89 9:56 HD:Apple ///:ROM - Disk I/O Page 1

```
ØØØØ|                       ;
ØØØØ|                       ; APPLE /// ROM - DISK I/O ROUTINES
ØØØØ|                       ; COPYRIGHT 1979 BY APPLE COMPUTER, INC.
ØØØØ|                       ;
ØØØØ|
ØØØØ|                                  .ABSOLUTE
ØØØØ|                                  .PROC   DISKIO
ØØØØ|                                  .ORG    ØFØØØ
ØØØØ|
FØØØ|                       ;**************************
FØØØ|                       ;     CRITICAL TIMING     *
FØØØ|                       ;   REQUIRES PAGE BOUND   *
FØØØ|                       ;   CONSIDERATIONS FOR    *
FØØØ|                       ;      CODE AND DATA      *
FØØØ|                       ;     -----CODE-----      *
FØØØ|                       ;   VIRTUALLY THE ENTIRE  *
FØØØ|                       ;     'WRITE' ROUTINE     *
FØØØ|                       ;      MUST NOT CROSS     *
FØØØ|                       ;     PAGE BOUNDARIES     *
FØØØ|                       ;  CRITICAL BRANCHES IN   *
FØØØ|                       ;  THE 'WRITE', 'READ',   *
FØØØ|                       ;  AND 'READ ADR' SUBRS   *
FØØØ|                       ;  WHICH MUST NOT CROSS   *
FØØØ|                       ;  PAGE BOUNDARIES ARE    *
FØØØ|                       ;  NOTED IN COMMENTS      *
FØØØ|                       ;                         *
FØØØ|                       ;**************************
FØØØ|                       ;                         *
FØØØ|                       ;         EQUATES         *
FØØØ|                       ;                         *
FØØØ| Ø2ØØ                  NBUF1      .EQU    Ø2ØØ
FØØØ| Ø3Ø2                  NBUF2      .EQU    Ø3Ø2         ; (ZERO PAGE AT $3ØØ)
FØØØ|                       ;
FØØØ| ØØ8Ø                  HRDERRS    .EQU    8Ø
FØØØ| ØØEØ                  DVMOT      .EQU    ØEØ
FØØØ|                       ;
FØØØ| ØØ81                  IBSLOT     .EQU    81
FØØØ| ØØ82                  IBDRVN     .EQU    IBSLOT+1
FØØØ| ØØ83                  IBTRK      .EQU    IBSLOT+2
FØØØ| ØØ84                  IBSECT     .EQU    IBSLOT+3
FØØØ| ØØ85                  IBBUFP     .EQU    IBSLOT+4     ; & 5
FØØØ| ØØ87                  IBCMD      .EQU    IBSLOT+6
FØØØ| ØØ88                  IBSTAT     .EQU    IBSLOT+7
FØØØ| ØØ89                  IBSMOD     .EQU    IBSLOT+8
FØØØ| ØØ89                  CSUM       .EQU    IBSMOD       ; USED ALSO FOR ADDRESS HEADER CKSUM
FØØØ| ØØ8A                  IOBPDN     .EQU    IBSLOT+9
FØØØ| ØØ8B                  IMASK      .EQU    IBSLOT+ØA
FØØØ| ØØ8C                  CURTRK     .EQU    IBSLOT+ØB
FØØØ| ØØ85                  DRVOTRK    .EQU    CURTRK-7
FØØØ|                       ; SLOT 4, DRIVE 1
FØØØ|                       ; SLOT 4, DRIVE 2
FØØØ|                       ; SLOT 5, DRIVE 1
FØØØ|                       ; SLOT 5, DRIVE 2
FØØØ|                       ; SLOT 6, DRIVE 1
FØØØ|                       ; SLOT 6, DRIVE 2
FØØØ| ØØ93                  RETRYCNT   .EQU    IBSLOT+12
FØØØ| ØØ94                  SEEKCNT    .EQU    IBSLOT+13
FØØØ| ØØ9B                  BUF        .EQU    IBSLOT+1A
FØØØ| ØØ9F                  ENVTEMP    .EQU    IBSLOT+1E
FØØØ|                       ; IBSLOT+$1F NOT USED
FØØØ|                       ;
FØØØ|                       ;**************************
FØØØ|                       ;                         *
FØØØ|                       ;     ----READADR----     *
FØØØ|                       ;                         *
FØØØ|                       ;**************************
FØØØ|                       ;
FØØØ| ØØ95                  COUNT      .EQU    IBSLOT+14    ; 'MUST FIND' COUNT.
FØØØ| ØØ95                  LAST       .EQU    IBSLOT+14    ; 'ODD BIT' NIBLS.
FØØØ| ØØ96                  CKSUM      .EQU    IBSLOT+15    ; CHECKSUM BYTE.
FØØØ| ØØ97                  CSSTV      .EQU    IBSLOT+16    ; FOUR BYTES
FØØØ|                       ;          CHECKSUM, SECTOR, TRACK, AND VOLUME.
FØØØ|                       ;
FØØØ|                       ;**************************
FØØØ|                       ;                         *
FØØØ|                       ;     ----WRITE----       *
FØØØ|                       ;                         *
FØØØ|                       ;     USES ALL NBUFS      *
FØØØ|                       ;       AND 32-BYTE       *
FØØØ|                       ;    DATA TABLE 'NIBL'    *
FØØØ|                       ;                         *
FØØØ|                       ;**************************
FØØØ|                       ;
FØØØ|                       ;**************************
FØØØ|                       ;                         *
FØØØ|                       ;      ----READ----       *
FØØØ|                       ;                         *
FØØØ|                       ;     USES ALL NBUFS      *
FØØØ|                       ;   USES LAST 54 BYTES    *
FØØØ|                       ;   OF A CODE PAGE FOR    *
```



```
FØØØ|                      ;   SIGNIFICANT BYTES     *
FØØØ|                      ;   OF DNIBL TABLE.       *
FØØØ|                      ;                         *
FØØØ|                      ;**************************
FØØØ|                      ;
FØØØ|                      ;**************************
FØØØ|                      ;                         *
FØØØ|                      ;     ----SEEK----        *
FØØØ|                      ;                         *
FØØØ|                      ;**************************
FØØØ|                      ;
FØØØ| ØØ95                 TRKCNT     .EQU    COUNT        ; HALFTRACKS MOVED COUNT.
FØØØ| ØØ9D                 PRIOR      .EQU    IBSLOT+1C
FØØØ| ØØ9E                 TRKN       .EQU    IBSLOT+1D
FØØØ|                      ;
FØØØ|                      ;**************************
FØØØ|                      ;                         *
FØØØ|                      ;    ----MSWAIT----       *
FØØØ|                      ;                         *
FØØØ|                      ;**************************
FØØØ|                      ;
FØØØ| ØØ99                 MONTIMEL   .EQU    CSSTV+2      ; MOTOR-ON TIME
FØØØ| ØØ9A                 MONTIMEH   .EQU    MONTIMEL+1   ; COUNTERS.
FØØØ|                      ;
FØØØ|                      ;**************************
FØØØ|                      ;                         *
FØØØ|                      ;     DEVICE ADDRESS      *
FØØØ|                      ;      ASSIGNMENTS        *
FØØØ|                      ;                         *
FØØØ|                      ;**************************
FØØØ|                      ;
FØØØ| CØ8Ø                 PHASEOFF   .EQU    ØCØ8Ø        ; STEPPER PHASE OFF.
FØØØ| CØ81                 PHASEON    .EQU    ØCØ81        ; STEPPER PHASE ON.
FØØØ| CØ8C                 Q6L        .EQU    ØCØ8C        ; Q7L,Q6L=READ
FØØØ| CØ8D                 Q6H        .EQU    ØCØ8D        ; Q7L,Q6H=SENSE WPROT
FØØØ| CØ8E                 Q7L        .EQU    ØCØ8E        ; Q7H,Q6L=WRITE
FØØØ| CØ8F                 Q7H        .EQU    ØCØ8F        ; Q7H,Q6H=WRITE STORE
FØØØ| FFEF                 INTERUPT   .EQU    ØFFEF
FØØØ| FFDF                 ENVIRON    .EQU    ØFFDF
FØØØ| ØØ8Ø                 ONEMEG     .EQU    8Ø
FØØØ| ØØ7F                 TWOMEG     .EQU    7F
FØØØ|                      ;
FØØØ|                      ;******************************
FØØØ|                      ;
FØØØ|                      ; EQUATES FOR RWTS AND BLOCK
FØØØ|                      ;
FØØØ|                      ;******************************
FØØØ|                      ;
FØØØ| CØ88                 MOTOROFF   .EQU    ØCØ88
FØØØ| CØ89                 MOTORON    .EQU    ØCØ89
FØØØ| CØ8A                 DRVOEN     .EQU    ØCØ8A
FØØØ| CØ8B                 DRV1EN     .EQU    ØCØ8B
FØØØ| CØ81                 PHASON     .EQU    ØCØ81
FØØØ| CØ8Ø                 PHSOFF     .EQU    ØCØ8Ø
FØØØ| ØØ97                 TEMP       .EQU    CSSTV        ; PUT ADDRESS INFO HERE
FØØØ| ØØ97                 CSUM1      .EQU    TEMP
FØØØ| ØØ98                 SECT       .EQU    CSUM1+1
FØØØ| ØØ99                 TRACK      .EQU    SECT+1
FØØØ| ØØ99                 TRKN1      .EQU    TRACK
FØØØ| ØØ9A                 VOLUME     .EQU    TRACK+1
FØØØ| ØØ83                 IBRERR     .EQU    HRDERRS+3
FØØØ| ØØ82                 IBDERR     .EQU    HRDERRS+2
FØØØ| ØØ81                 IBWPER     .EQU    HRDERRS+1
FØØØ| ØØ8Ø                 IBNODRV    .EQU    HRDERRS
FØØØ|                      ;
FØØØ|                      ;******************************
FØØØ|                      ;                             *
FØØØ|                      ;     READ WRITE A            *
FØØØ|                      ;   TRACK AND SECTOR          *
FØØØ|                      ;                             *
FØØØ|                      ;******************************
FØØØ|                      ;
FØØØ| AØ Ø1                REGRWTS    LDY     #Ø1          ; RETRY COUNT
FØØ2| A6 81                           LDX     IBSLOT       ; GET SLOT # FOR THIS OPERATION
FØØ4| 84 94                           STY     SEEKCNT      ; ONLY ONE RECALIBRATE PER CALL
FØØ6| A9 Ø5                           LDA     #ØØ5
FØØ8| 85 8F                           STA     Ø8F
FØØA| Ø8                              PHP                  ; DETERMINE INTERRUPT STATUS
FØØB| 68                              PLA
FØØC| 6A                              ROR     A
FØØD| 6A                              ROR     A            ; GET INTERRUPT FLAG INTO BIT 7
FØØE| 6A                              ROR     A
FØØF| 6A                              ROR     A
FØ1Ø| 85 8B                           STA     IMASK
FØ12| AD DFFF                         LDA     ENVIRON      ; PRESERVE ENVIRONMENT
FØ15| 85 9F                           STA     ENVTEMP
FØ17| 2Ø 2BF1                         JSR     CHKDRV       ; SET ZERO FLAG IF MOTOR STOPPED
FØ1A| Ø8                              PHP                  ; SAVE TEST RESULTS
FØ1B| A5 85                           LDA     IBBUFP       ; MOVE OUT POINTER TO BUFFER INTO ZPAGE
FØ1D| 85 9B                           STA     BUF
```

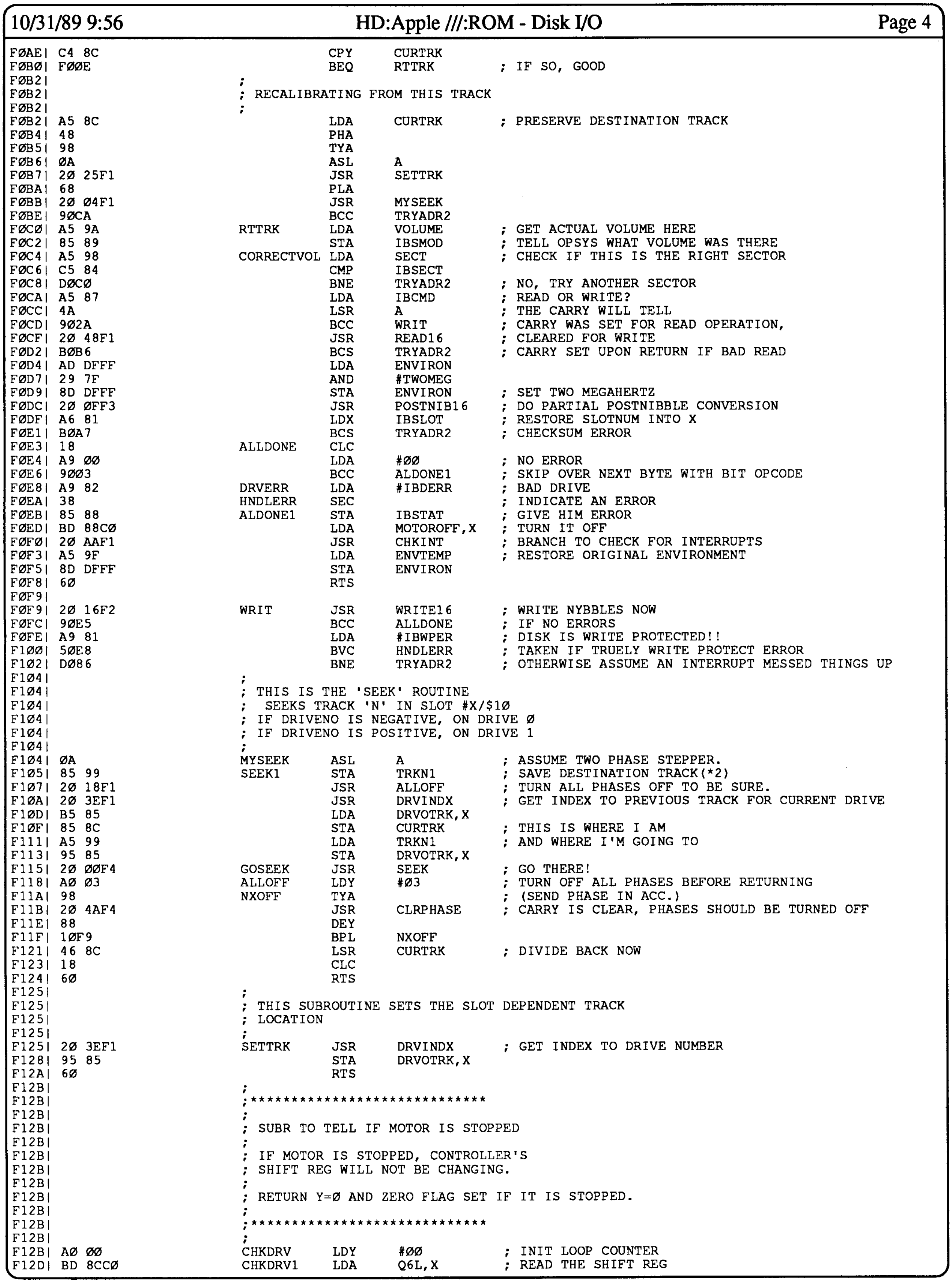
“APPLE_PAT_4_383_296_C_04” 166 KB 2000-02-27 dpi: 300h x 300v pix: 2312h x 3099v



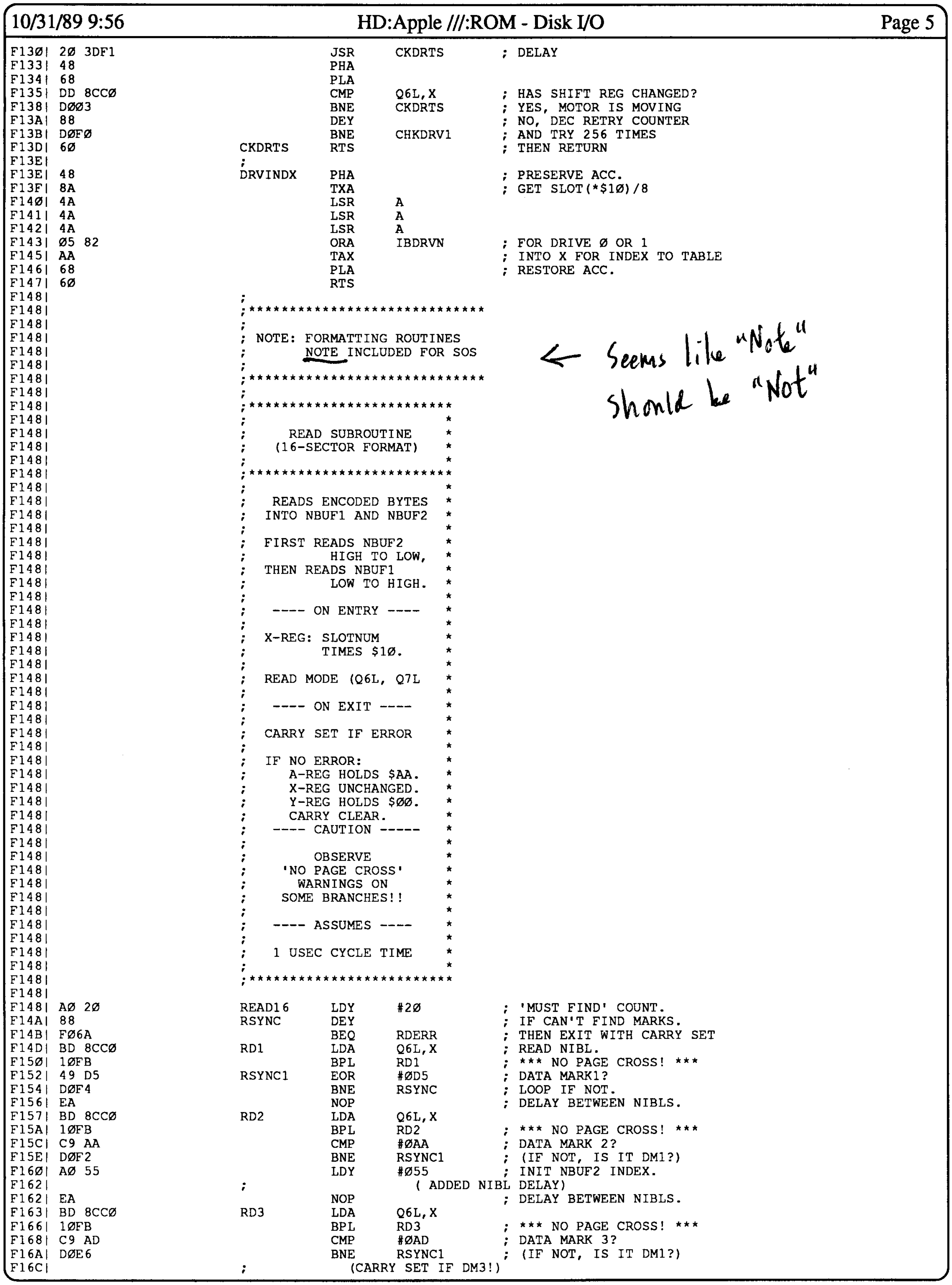
```
FØ1F| A5 86                            LDA     IBBUFP+1
FØ21| 85 9C                            STA     BUF+1
FØ23| A9 EØ                            LDA     #DVMOT
FØ25| 85 9A                            STA     MONTIMEH
FØ27| A5 82                            LDA     IBDRVN       ; DETERMINE DRIVE ONE OR TWO
FØ29| C5 8A                            CMP     IOBPDN       ; SAME DRIVE USED BEFORE
FØ2B| 85 8A                            STA     IOBPDN       ; SAVE IT FOR NEXT TIME
FØ2D| Ø8                               PHP                  ; KEEP RESULTS OF COMPARE
FØ2E| 6A                               ROR     A            ; GET DRIVE NUMBER INTO CARRY
FØ2F| BD 89CØ                          LDA     MOTORON,X    ; TURN ON THE DRIVE
FØ32| 9ØØ1                             BCC     DRIVSEL      ; BRANCH IF DRIVE 1 SELECTED
FØ34| E8                               INX                  ; SELECT DRIVE 2
FØ35| BD 8ACØ               DRIVSEL    LDA     DRVOEN,X
FØ38| 2Ø 4CF3                          JSR     SET1MEG      ; INSURE ONE MEGAHERTZ OPERATION
FØ3B| 28                               PLP                  ; WAS IT SAME DRIVE?
FØ3C| FØØA                             BEQ     OK
FØ3E| 28                               PLP                  ; MUST INDICATE DRIVE OFF BY SETTING ZERO FLAG
FØ3F| AØ Ø7                            LDY     #Ø7          ; DELAY 15Ø MS BEFORE STEPPING
FØ41| 2Ø 56F4               DRVWAIT    JSR     MSWAIT       ; (ON RETURN A=Ø)
FØ44| 88                               DEY
FØ45| DØFA                             BNE     DRVWAIT
FØ47| Ø8                               PHP                  ; NOW ZERO FLAG SET
FØ48| A5 83                 OK         LDA     IBTRK        ; GET DESTINATION TRACK
FØ4A| A6 81                            LDX     IBSLOT       ; RESTORE PROPER X (SLOT*16)
FØ4C| 2Ø Ø4F1                          JSR     MYSEEK       ; AND GO TO IT
FØ4F|                       ; NOW AT THE DESIRED TRACK   WAS THE MOTOR ON TO START WITH?
FØ4F| 28                               PLP                  ; WAS MOTOR ON?
FØ5Ø| DØ17                             BNE     TRYTRK       ; IF SO, DON'T DELAY, GET IT TODAY!
FØ52|                       ;
FØ52|                       ; MOTOR WAS OFF, WAIT FOR IT TO SPEED UP
FØ52|                       ;
FØ52| AØ 12                 MOTOF      LDY     #12          ; WAIT EXACTLY 1ØØ US FOR EACH COUNT
FØ54| 88                    CONWAIT    DEY                  ; IN MONTIME
FØ55| DØFD                             BNE     CONWAIT
FØ57| E6 99                            INC     MONTIMEL     ; COUNT UP TO ØØØØ
FØ59| DØF7                             BNE     MOTOF
FØ5B| E6 9A                            INC     MONTIMEH
FØ5D| 3ØF3                             BMI     MOTOF
FØ5F|                       ;
FØ5F|                       ;******************************
FØ5F|                       ; MOTOR SHOULD BE UP TO SPEED
FØ5F|                       ; IF IT STILL LOOKS STOPPED THEN
FØ5F|                       ; THE DRIVE IS NOT PRESENT.
FØ5F|                       ;
FØ5F|                       ;******************************
FØ5F|                       ;
FØ5F| 2Ø 2BF1                          JSR     CHKDRV       ; IS DRIVE PRESENT?
FØ62| DØØ5                             BNE     TRYTRK       ; YES, CONTINUE
FØ64| A9 8Ø                 NODRIVERR  LDA     #IBNODRV     ; NO, GET TELL EM NO DRIVE
FØ66| 4C EAFØ                          JMP     HNDLERR
FØ69|                       ;
FØ69|                       ; NOW CHECK IF IT IS NOT THE FORMAT DISK COMMAND,
FØ69|                       ;  LOCATE THE CORRECT SECTOR FOR THIS OPERATION
FØ69|                       ;
FØ69| A5 87                 TRYTRK     LDA     IBCMD        ; GET COMMAND CODE #
FØ6B| FØ76                             BEQ     ALLDONE      ; IF NULL COMMAND, GO HOME TO BED
FØ6D| C9 Ø3                            CMP     #Ø3          ; COMMAND IN RANGE?
FØ6F| BØ72                             BCS     ALLDONE      ; NO, DO NOTHING!
FØ71| 6A                               ROR     A            ; SET CARRY=1 FOR READ, Ø FOR WRITE
FØ72| BØØB                             BCS     TRYTRK2      ; MUST PRENIBBLIZE FOR WRITE
FØ74| AD DFFF                          LDA     ENVIRON
FØ77| 29 7F                            AND     #TWOMEG      ; SHIFT TO HIGH SPEED!
FØ79| 8D DFFF                          STA     ENVIRON
FØ7C| 2Ø C4F2                          JSR     PRENIB16
FØ7F| AØ 7F                 TRYTRK2    LDY     #7F          ; ONLY 127 RETRIES OF ANY KIND
FØ81| 84 93                            STY     RETRYCNT
FØ83| A6 81                 TRYADR     LDX     IBSLOT       ; GET SLOT NUM INTO X-REG
FØ85| 2Ø B9F1                          JSR     RDADR16      ; READ NEXT ADDRESS FIELD
FØ88| 9Ø22                             BCC     RDRIGHT      ; IF READ IS RIGHT, HURRAH!
FØ8A| 2Ø AAF1               TRYADR2    JSR     CHKINT       ; BRANCH TO CHECK FOR INTERRUPTS
FØ8D| C6 93                            DEC     RETRYCNT     ; ANOTHER MISTAKE!!
FØ8F| 1ØF2                             BPL     TRYADR       ; WELL, LET IT GO THIS TIME
FØ91| C6 94                            DEC     SEEKCNT      ; ONLY RECALIBRATE ONCE!
FØ93| DØ53                             BNE     DRVERR       ; TRIED TO RECALIBRATE A SECOND TIME, ERROR!
FØ95| A5 8F                            LDA     Ø8F          ; ANOTHER MISTAKE!!
FØ97| 3ØEA                             BMI     TRYADR       ; WELL, LET IT GO THIS TIME
FØ99| A5 8C                            LDA     CURTRK
FØ9B| 48                               PHA                  ; SAVE TRACK WE REALLY WANT
FØ9C| A9 6Ø                            LDA     #6Ø          ; RECALIBRATE ALL OVER AGAIN!   ERROR!
FØ9E| 2Ø 25F1                          JSR     SETTRK       ; PRETEND TO BE ON TRACK 8Ø
FØA1| A9 ØØ                            LDA     #ØØ
FØA3| 2Ø Ø4F1                          JSR     MYSEEK       ; MOVE TO TRACK ØØ
FØA6| 68                    GOCAL1     PLA
FØA7| 2Ø Ø4F1               GOCAL      JSR     MYSEEK       ; GO TO CORRECT TRACK THIS TIME!
FØAA| 9ØD7                             BCC     TRYADR       ; LOOP BACK, TRY AGAIN ON THIS TRACK
FØAC|                       ;
FØAC|                       ; HAVE NOW READ AN ADDRESS FIELD CORRECTLY.
FØAC|                       ; MAKE SURE THIS IS THE TRACK, SECTOR, AND VOLUME DESIRED.
FØAC|                       ;
FØAC| A4 99                 RDRIGHT    LDY     TRACK        ; ON THE RIGHT TRACK?
```

CMD
1 → Read
2 → Write

"APPLE_PAT_4_383_296_C_05" 224 KB 2000-02-28 dpi: 300h x 300v pix: 2324h x 3103v



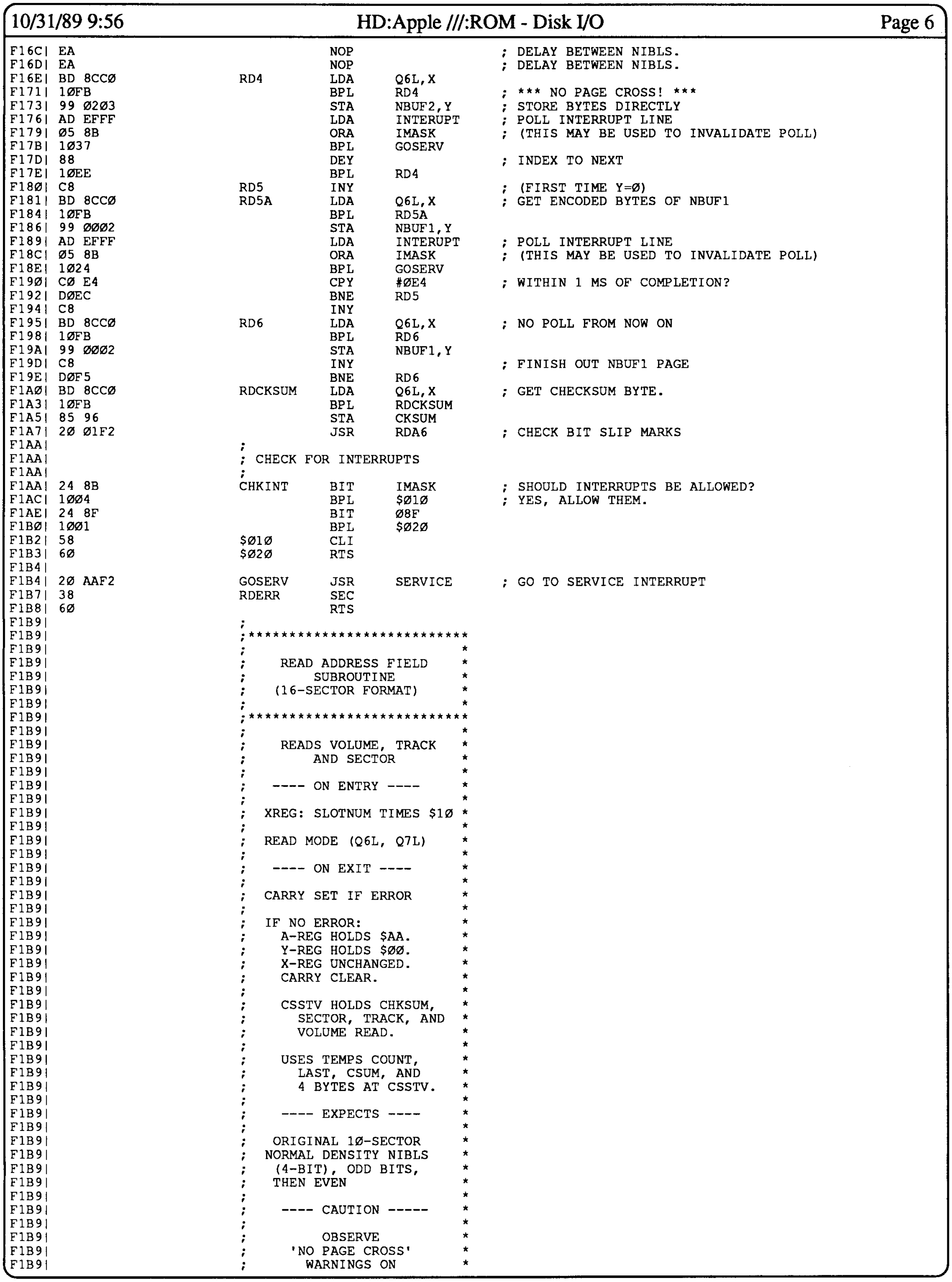
```
FØAE| C4 8C                         CPY      CURTRK
FØBØ| FØØE                          BEQ      RTTRK         ; IF SO, GOOD
FØB2|                    ;
FØB2|                    ; RECALIBRATING FROM THIS TRACK
FØB2|                    ;
FØB2| A5 8C                         LDA      CURTRK        ; PRESERVE DESTINATION TRACK
FØB4| 48                            PHA
FØB5| 98                            TYA
FØB6| ØA                            ASL      A
FØB7| 2Ø 25F1                       JSR      SETTRK
FØBA| 68                            PLA
FØBB| 2Ø Ø4F1                       JSR      MYSEEK
FØBE| 9ØCA                          BCC      TRYADR2
FØCØ| A5 9A              RTTRK      LDA      VOLUME        ; GET ACTUAL VOLUME HERE
FØC2| 85 89                         STA      IBSMOD        ; TELL OPSYS WHAT VOLUME WAS THERE
FØC4| A5 98              CORRECTVOL LDA      SECT          ; CHECK IF THIS IS THE RIGHT SECTOR
FØC6| C5 84                         CMP      IBSECT
FØC8| DØCØ                          BNE      TRYADR2       ; NO, TRY ANOTHER SECTOR
FØCA| A5 87                         LDA      IBCMD         ; READ OR WRITE?
FØCC| 4A                            LSR      A             ; THE CARRY WILL TELL
FØCD| 9Ø2A                          BCC      WRIT          ; CARRY WAS SET FOR READ OPERATION,
FØCF| 2Ø 48F1                       JSR      READ16        ; CLEARED FOR WRITE
FØD2| BØB6                          BCS      TRYADR2       ; CARRY SET UPON RETURN IF BAD READ
FØD4| AD DFFF                       LDA      ENVIRON
FØD7| 29 7F                         AND      #TWOMEG
FØD9| 8D DFFF                       STA      ENVIRON       ; SET TWO MEGAHERTZ
FØDC| 2Ø ØFF3                       JSR      POSTNIB16     ; DO PARTIAL POSTNIBBLE CONVERSION
FØDF| A6 81                         LDX      IBSLOT        ; RESTORE SLOTNUM INTO X
FØE1| BØA7                          BCS      TRYADR2       ; CHECKSUM ERROR
FØE3| 18                 ALLDONE    CLC
FØE4| A9 ØØ                         LDA      #ØØ           ; NO ERROR
FØE6| 9ØØ3                          BCC      ALDONE1       ; SKIP OVER NEXT BYTE WITH BIT OPCODE
FØE8| A9 82              DRVERR     LDA      #IBDERR       ; BAD DRIVE
FØEA| 38                 HNDLERR    SEC                    ; INDICATE AN ERROR
FØEB| 85 88              ALDONE1    STA      IBSTAT        ; GIVE HIM ERROR
FØED| BD 88CØ                       LDA      MOTOROFF,X    ; TURN IT OFF
FØFØ| 2Ø AAF1                       JSR      CHKINT        ; BRANCH TO CHECK FOR INTERRUPTS
FØF3| A5 9F                         LDA      ENVTEMP       ; RESTORE ORIGINAL ENVIRONMENT
FØF5| 8D DFFF                       STA      ENVIRON
FØF8| 6Ø                            RTS
FØF9|
FØF9| 2Ø 16F2            WRIT       JSR      WRITE16       ; WRITE NYBBLES NOW
FØFC| 9ØE5                          BCC      ALLDONE       ; IF NO ERRORS
FØFE| A9 81                         LDA      #IBWPER       ; DISK IS WRITE PROTECTED!!
F1ØØ| 5ØE8                          BVC      HNDLERR       ; TAKEN IF TRUELY WRITE PROTECT ERROR
F1Ø2| DØ86                          BNE      TRYADR2       ; OTHERWISE ASSUME AN INTERRUPT MESSED THINGS UP
F1Ø4|                    ;
F1Ø4|                    ; THIS IS THE 'SEEK' ROUTINE
F1Ø4|                    ;  SEEKS TRACK 'N' IN SLOT #X/$1Ø
F1Ø4|                    ; IF DRIVENO IS NEGATIVE, ON DRIVE Ø
F1Ø4|                    ; IF DRIVENO IS POSITIVE, ON DRIVE 1
F1Ø4|                    ;
F1Ø4| ØA                 MYSEEK     ASL      A             ; ASSUME TWO PHASE STEPPER.
F1Ø5| 85 99              SEEK1      STA      TRKN1         ; SAVE DESTINATION TRACK(*2)
F1Ø7| 2Ø 18F1                       JSR      ALLOFF        ; TURN ALL PHASES OFF TO BE SURE.
F1ØA| 2Ø 3EF1                       JSR      DRVINDX       ; GET INDEX TO PREVIOUS TRACK FOR CURRENT DRIVE
F1ØD| B5 85                         LDA      DRVOTRK,X
F1ØF| 85 8C                         STA      CURTRK        ; THIS IS WHERE I AM
F111| A5 99                         LDA      TRKN1         ; AND WHERE I'M GOING TO
F113| 95 85                         STA      DRVOTRK,X
F115| 2Ø ØØF4            GOSEEK     JSR      SEEK          ; GO THERE!
F118| AØ Ø3              ALLOFF     LDY      #Ø3           ; TURN OFF ALL PHASES BEFORE RETURNING
F11A| 98                 NXOFF      TYA                    ; (SEND PHASE IN ACC.)
F11B| 2Ø 4AF4                       JSR      CLRPHASE      ; CARRY IS CLEAR, PHASES SHOULD BE TURNED OFF
F11E| 88                            DEY
F11F| 1ØF9                          BPL      NXOFF
F121| 46 8C                         LSR      CURTRK        ; DIVIDE BACK NOW
F123| 18                            CLC
F124| 6Ø                            RTS
F125|                    ;
F125|                    ; THIS SUBROUTINE SETS THE SLOT DEPENDENT TRACK
F125|                    ; LOCATION
F125|                    ;
F125| 2Ø 3EF1            SETTRK     JSR      DRVINDX       ; GET INDEX TO DRIVE NUMBER
F128| 95 85                         STA      DRVOTRK,X
F12A| 6Ø                            RTS
F12B|                    ;
F12B|                    ;****************************
F12B|                    ;
F12B|                    ; SUBR TO TELL IF MOTOR IS STOPPED
F12B|                    ;
F12B|                    ; IF MOTOR IS STOPPED, CONTROLLER'S
F12B|                    ; SHIFT REG WILL NOT BE CHANGING.
F12B|                    ;
F12B|                    ; RETURN Y=Ø AND ZERO FLAG SET IF IT IS STOPPED.
F12B|                    ;
F12B|                    ;****************************
F12B|                    ;
F12B| AØ ØØ              CHKDRV     LDY      #ØØ           ; INIT LOOP COUNTER
F12D| BD 8CCØ            CHKDRV1    LDA      Q6L,X         ; READ THE SHIFT REG
```



10/31/89 9:56 HD:Apple ///:ROM - Disk I/O Page 5

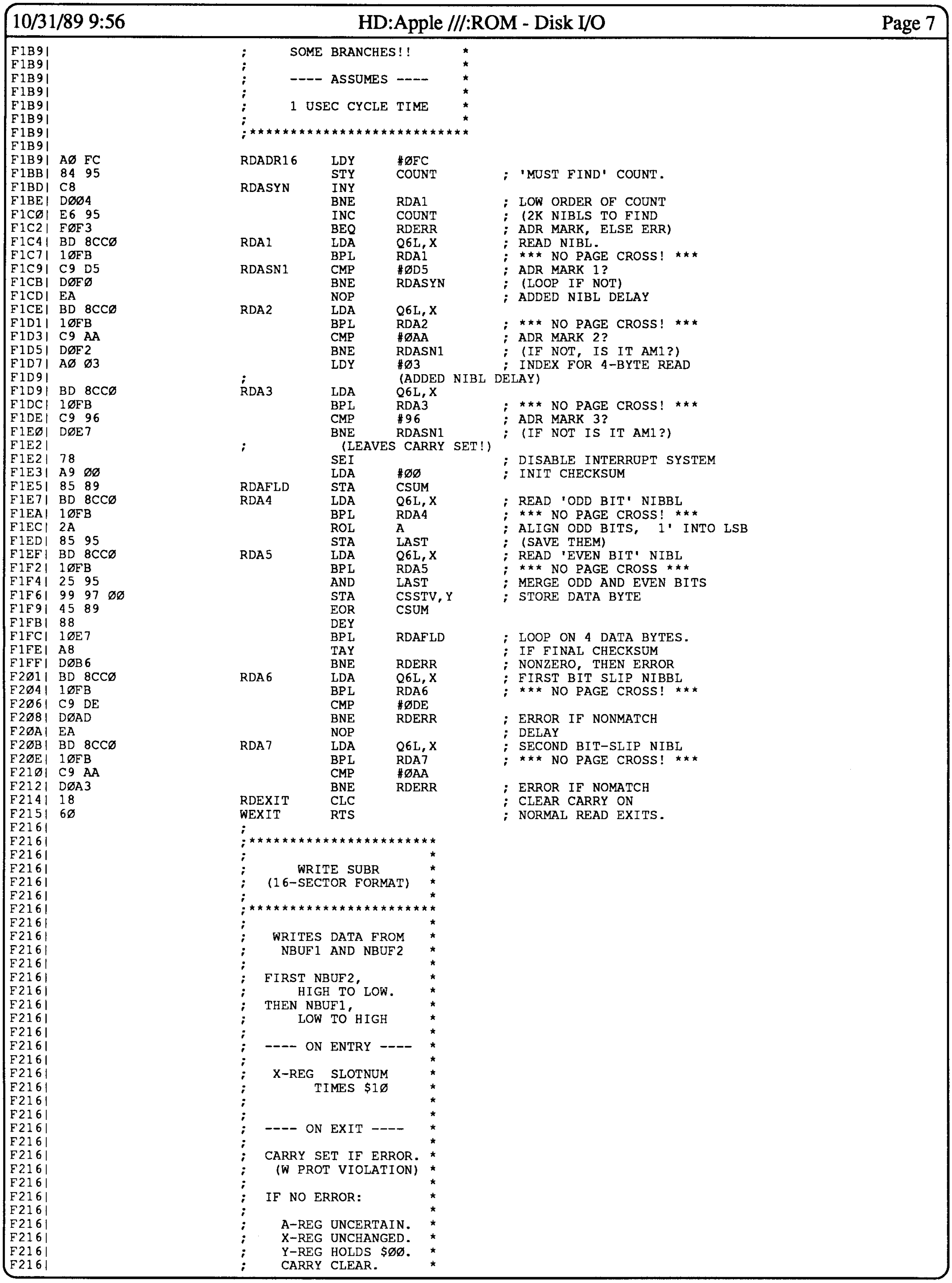
```
F13Ø| 2Ø 3DF1                     JSR     CKDRTS       ; DELAY
F133| 48                          PHA
F134| 68                          PLA
F135| DD 8CCØ                     CMP     Q6L,X        ; HAS SHIFT REG CHANGED?
F138| DØØ3                        BNE     CKDRTS       ; YES, MOTOR IS MOVING
F13A| 88                          DEY                  ; NO, DEC RETRY COUNTER
F13B| DØFØ                        BNE     CHKDRV1      ; AND TRY 256 TIMES
F13D| 6Ø               CKDRTS     RTS                  ; THEN RETURN
F13E|                  ;
F13E| 48               DRVINDX    PHA                  ; PRESERVE ACC.
F13F| 8A                          TXA                  ; GET SLOT(*$1Ø)/8
F14Ø| 4A                          LSR     A
F141| 4A                          LSR     A
F142| 4A                          LSR     A
F143| Ø5 82                       ORA     IBDRVN       ; FOR DRIVE Ø OR 1
F145| AA                          TAX                  ; INTO X FOR INDEX TO TABLE
F146| 68                          PLA                  ; RESTORE ACC.
F147| 6Ø                          RTS
F148|                  ;
F148|                  ;*****************************
F148|                  ;
F148|                  ; NOTE: FORMATTING ROUTINES
F148|                  ;       NOTE INCLUDED FOR SOS
F148|                  ;
F148|                  ;*****************************
F148|                  ;
F148|                  ;************************
F148|                  ;                       *
F148|                  ;     READ SUBROUTINE   *
F148|                  ;   (16-SECTOR FORMAT)  *
F148|                  ;                       *
F148|                  ;************************
F148|                  ;                       *
F148|                  ;   READS ENCODED BYTES *
F148|                  ;  INTO NBUF1 AND NBUF2 *
F148|                  ;                       *
F148|                  ;  FIRST READS NBUF2    *
F148|                  ;          HIGH TO LOW, *
F148|                  ;  THEN READS NBUF1     *
F148|                  ;          LOW TO HIGH. *
F148|                  ;                       *
F148|                  ;   ---- ON ENTRY ----  *
F148|                  ;                       *
F148|                  ;  X-REG: SLOTNUM       *
F148|                  ;         TIMES $1Ø.    *
F148|                  ;                       *
F148|                  ;  READ MODE (Q6L, Q7L  *
F148|                  ;                       *
F148|                  ;   ---- ON EXIT ----   *
F148|                  ;                       *
F148|                  ;  CARRY SET IF ERROR   *
F148|                  ;                       *
F148|                  ;  IF NO ERROR:         *
F148|                  ;     A-REG HOLDS $AA.  *
F148|                  ;     X-REG UNCHANGED.  *
F148|                  ;     Y-REG HOLDS $ØØ.  *
F148|                  ;     CARRY CLEAR.      *
F148|                  ;   ---- CAUTION -----  *
F148|                  ;                       *
F148|                  ;        OBSERVE        *
F148|                  ;    'NO PAGE CROSS'    *
F148|                  ;      WARNINGS ON      *
F148|                  ;    SOME BRANCHES!!    *
F148|                  ;                       *
F148|                  ;   ---- ASSUMES ----   *
F148|                  ;                       *
F148|                  ;   1 USEC CYCLE TIME   *
F148|                  ;                       *
F148|                  ;************************
F148|
F148| AØ 2Ø            READ16     LDY     #2Ø          ; 'MUST FIND' COUNT.
F14A| 88               RSYNC      DEY                  ; IF CAN'T FIND MARKS.
F14B| FØ6A                        BEQ     RDERR        ; THEN EXIT WITH CARRY SET
F14D| BD 8CCØ          RD1        LDA     Q6L,X        ; READ NIBL.
F15Ø| 1ØFB                        BPL     RD1          ; *** NO PAGE CROSS! ***
F152| 49 D5            RSYNC1     EOR     #ØD5         ; DATA MARK1?
F154| DØF4                        BNE     RSYNC        ; LOOP IF NOT.
F156| EA                          NOP                  ; DELAY BETWEEN NIBLS.
F157| BD 8CCØ          RD2        LDA     Q6L,X
F15A| 1ØFB                        BPL     RD2          ; *** NO PAGE CROSS! ***
F15C| C9 AA                       CMP     #ØAA         ; DATA MARK 2?
F15E| DØF2                        BNE     RSYNC1       ; (IF NOT, IS IT DM1?)
F16Ø| AØ 55                       LDY     #Ø55         ; INIT NBUF2 INDEX.
F162|                  ;                    ( ADDED NIBL DELAY)
F162| EA                          NOP                  ; DELAY BETWEEN NIBLS.
F163| BD 8CCØ          RD3        LDA     Q6L,X
F166| 1ØFB                        BPL     RD3          ; *** NO PAGE CROSS! ***
F168| C9 AD                       CMP     #ØAD         ; DATA MARK 3?
F16A| DØE6                        BNE     RSYNC1       ; (IF NOT, IS IT DM1?)
F16C|                  ;             (CARRY SET IF DM3!)
```

← Seems like "Note" should be "Not"

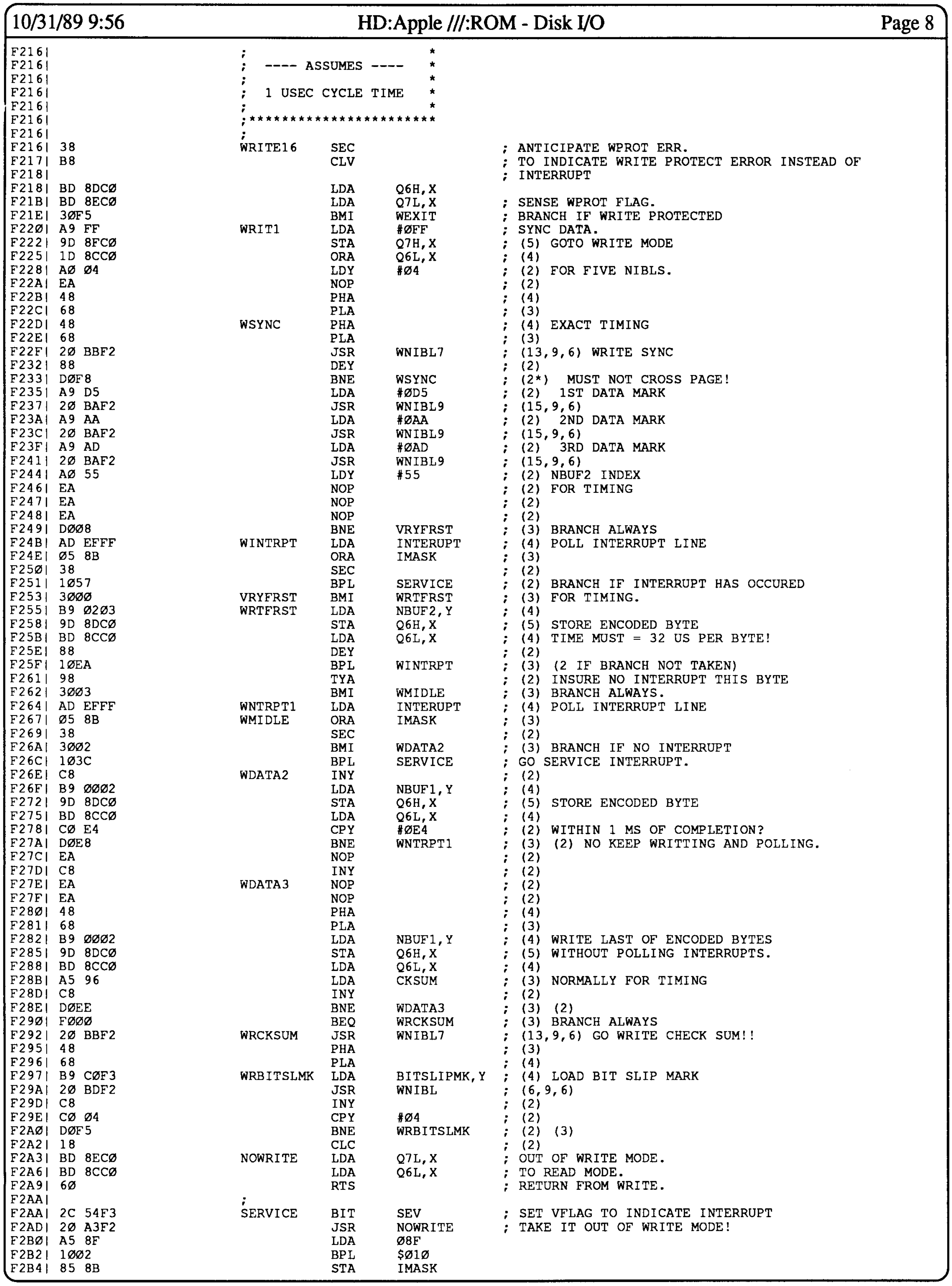


```
F16C| EA                               NOP                  ; DELAY BETWEEN NIBLS.
F16D| EA                               NOP                  ; DELAY BETWEEN NIBLS.
F16E| BD 8CCØ               RD4        LDA     Q6L,X
F171| 1ØFB                             BPL     RD4          ; *** NO PAGE CROSS! ***
F173| 99 Ø2Ø3                          STA     NBUF2,Y      ; STORE BYTES DIRECTLY
F176| AD EFFF                          LDA     INTERUPT     ; POLL INTERRUPT LINE
F179| Ø5 8B                            ORA     IMASK        ; (THIS MAY BE USED TO INVALIDATE POLL)
F17B| 1Ø37                             BPL     GOSERV
F17D| 88                               DEY                  ; INDEX TO NEXT
F17E| 1ØEE                             BPL     RD4
F18Ø| C8                    RD5        INY                  ; (FIRST TIME Y=Ø)
F181| BD 8CCØ               RD5A       LDA     Q6L,X        ; GET ENCODED BYTES OF NBUF1
F184| 1ØFB                             BPL     RD5A
F186| 99 ØØØ2                          STA     NBUF1,Y
F189| AD EFFF                          LDA     INTERUPT     ; POLL INTERRUPT LINE
F18C| Ø5 8B                            ORA     IMASK        ; (THIS MAY BE USED TO INVALIDATE POLL)
F18E| 1Ø24                             BPL     GOSERV
F19Ø| CØ E4                            CPY     #ØE4         ; WITHIN 1 MS OF COMPLETION?
F192| DØEC                             BNE     RD5
F194| C8                               INY
F195| BD 8CCØ               RD6        LDA     Q6L,X        ; NO POLL FROM NOW ON
F198| 1ØFB                             BPL     RD6
F19A| 99 ØØØ2                          STA     NBUF1,Y
F19D| C8                               INY                  ; FINISH OUT NBUF1 PAGE
F19E| DØF5                             BNE     RD6
F1AØ| BD 8CCØ               RDCKSUM    LDA     Q6L,X        ; GET CHECKSUM BYTE.
F1A3| 1ØFB                             BPL     RDCKSUM
F1A5| 85 96                            STA     CKSUM
F1A7| 2Ø Ø1F2                          JSR     RDA6         ; CHECK BIT SLIP MARKS
F1AA|                       ;
F1AA|                       ; CHECK FOR INTERRUPTS
F1AA|                       ;
F1AA| 24 8B                 CHKINT     BIT     IMASK        ; SHOULD INTERRUPTS BE ALLOWED?
F1AC| 1ØØ4                             BPL     $Ø1Ø         ; YES, ALLOW THEM.
F1AE| 24 8F                            BIT     Ø8F
F1BØ| 1ØØ1                             BPL     $Ø2Ø
F1B2| 58                    $Ø1Ø       CLI
F1B3| 6Ø                    $Ø2Ø       RTS
F1B4|
F1B4| 2Ø AAF2               GOSERV     JSR     SERVICE      ; GO TO SERVICE INTERRUPT
F1B7| 38                    RDERR      SEC
F1B8| 6Ø                               RTS
F1B9|                       ;
F1B9|                       ;****************************
F1B9|                       ;                           *
F1B9|                       ;    READ ADDRESS FIELD     *
F1B9|                       ;        SUBROUTINE         *
F1B9|                       ;   (16-SECTOR FORMAT)      *
F1B9|                       ;                           *
F1B9|                       ;****************************
F1B9|                       ;                           *
F1B9|                       ;    READS VOLUME, TRACK    *
F1B9|                       ;        AND SECTOR         *
F1B9|                       ;                           *
F1B9|                       ;   ---- ON ENTRY ----      *
F1B9|                       ;                           *
F1B9|                       ;  XREG: SLOTNUM TIMES $1Ø  *
F1B9|                       ;                           *
F1B9|                       ;  READ MODE (Q6L, Q7L)     *
F1B9|                       ;                           *
F1B9|                       ;   ---- ON EXIT ----       *
F1B9|                       ;                           *
F1B9|                       ;  CARRY SET IF ERROR       *
F1B9|                       ;                           *
F1B9|                       ;  IF NO ERROR:             *
F1B9|                       ;    A-REG HOLDS $AA.       *
F1B9|                       ;    Y-REG HOLDS $ØØ.       *
F1B9|                       ;    X-REG UNCHANGED.       *
F1B9|                       ;    CARRY CLEAR.           *
F1B9|                       ;                           *
F1B9|                       ;    CSSTV HOLDS CHKSUM,    *
F1B9|                       ;      SECTOR, TRACK, AND   *
F1B9|                       ;      VOLUME READ.         *
F1B9|                       ;                           *
F1B9|                       ;    USES TEMPS COUNT,      *
F1B9|                       ;      LAST, CSUM, AND      *
F1B9|                       ;      4 BYTES AT CSSTV.    *
F1B9|                       ;                           *
F1B9|                       ;    ---- EXPECTS ----      *
F1B9|                       ;                           *
F1B9|                       ;   ORIGINAL 1Ø-SECTOR      *
F1B9|                       ;  NORMAL DENSITY NIBLS     *
F1B9|                       ;   (4-BIT), ODD BITS,      *
F1B9|                       ;   THEN EVEN               *
F1B9|                       ;                           *
F1B9|                       ;    ---- CAUTION -----     *
F1B9|                       ;                           *
F1B9|                       ;         OBSERVE           *
F1B9|                       ;     'NO PAGE CROSS'       *
F1B9|                       ;       WARNINGS ON         *
```

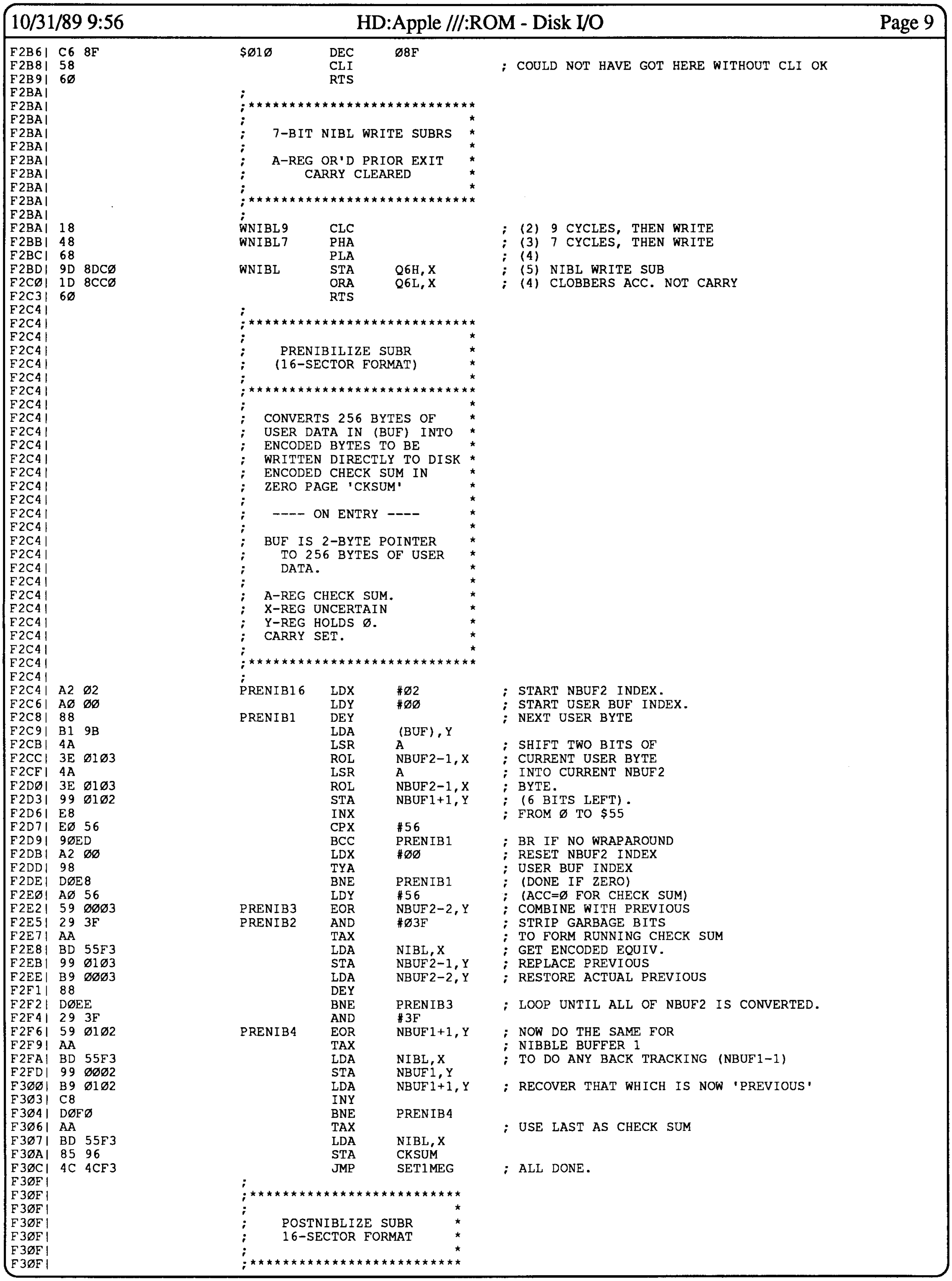
“APPLE_PAT_4_383_296_C_08” 160 KB 2000-02-28 dpi: 300h x 300v pix: 2325h x 3091v



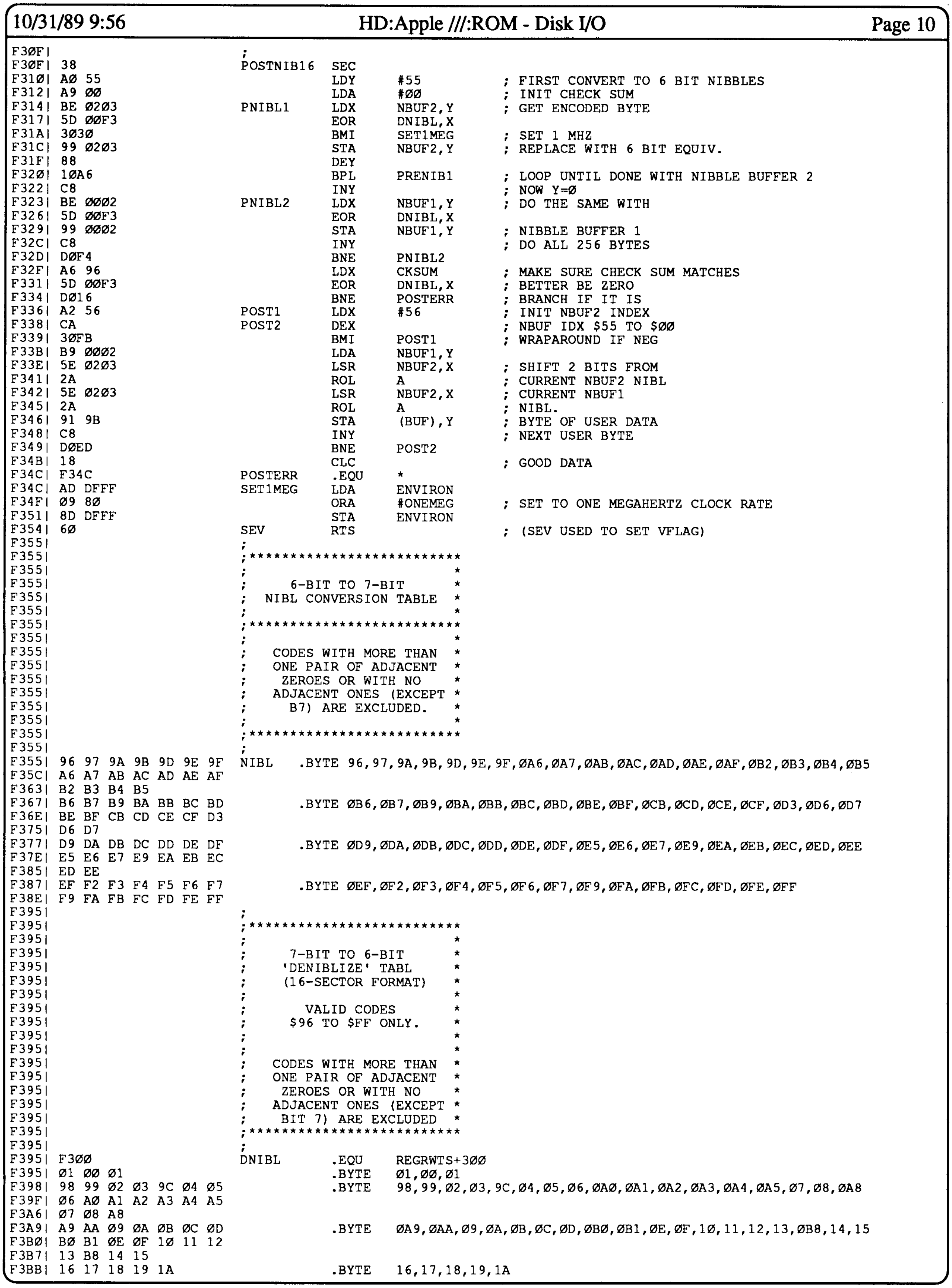
```
F1B9|                       ;     SOME BRANCHES!!      *
F1B9|                       ;                          *
F1B9|                       ;     ---- ASSUMES ----    *
F1B9|                       ;                          *
F1B9|                       ;     1 USEC CYCLE TIME    *
F1B9|                       ;                          *
F1B9|                       ;***************************
F1B9|
F1B9| AØ FC                 RDADR16    LDY     #ØFC
F1BB| 84 95                            STY     COUNT        ; 'MUST FIND' COUNT.
F1BD| C8                    RDASYN     INY
F1BE| DØØ4                             BNE     RDA1         ; LOW ORDER OF COUNT
F1CØ| E6 95                            INC     COUNT        ; (2K NIBLS TO FIND
F1C2| FØF3                             BEQ     RDERR        ; ADR MARK, ELSE ERR)
F1C4| BD 8CCØ               RDA1       LDA     Q6L,X        ; READ NIBL.
F1C7| 1ØFB                             BPL     RDA1         ; *** NO PAGE CROSS! ***
F1C9| C9 D5                 RDASN1     CMP     #ØD5         ; ADR MARK 1?
F1CB| DØFØ                             BNE     RDASYN       ; (LOOP IF NOT)
F1CD| EA                               NOP                  ; ADDED NIBL DELAY
F1CE| BD 8CCØ               RDA2       LDA     Q6L,X
F1D1| 1ØFB                             BPL     RDA2         ; *** NO PAGE CROSS! ***
F1D3| C9 AA                            CMP     #ØAA         ; ADR MARK 2?
F1D5| DØF2                             BNE     RDASN1       ; (IF NOT, IS IT AM1?)
F1D7| AØ Ø3                            LDY     #Ø3          ; INDEX FOR 4-BYTE READ
F1D9|                       ;                  (ADDED NIBL DELAY)
F1D9| BD 8CCØ               RDA3       LDA     Q6L,X
F1DC| 1ØFB                             BPL     RDA3         ; *** NO PAGE CROSS! ***
F1DE| C9 96                            CMP     #96          ; ADR MARK 3?
F1EØ| DØE7                             BNE     RDASN1       ; (IF NOT IS IT AM1?)
F1E2|                       ;           (LEAVES CARRY SET!)
F1E2| 78                               SEI                  ; DISABLE INTERRUPT SYSTEM
F1E3| A9 ØØ                            LDA     #ØØ          ; INIT CHECKSUM
F1E5| 85 89                 RDAFLD     STA     CSUM
F1E7| BD 8CCØ               RDA4       LDA     Q6L,X        ; READ 'ODD BIT' NIBBL
F1EA| 1ØFB                             BPL     RDA4         ; *** NO PAGE CROSS! ***
F1EC| 2A                               ROL     A            ; ALIGN ODD BITS,  1' INTO LSB
F1ED| 85 95                            STA     LAST         ; (SAVE THEM)
F1EF| BD 8CCØ               RDA5       LDA     Q6L,X        ; READ 'EVEN BIT' NIBL
F1F2| 1ØFB                             BPL     RDA5         ; *** NO PAGE CROSS ***
F1F4| 25 95                            AND     LAST         ; MERGE ODD AND EVEN BITS
F1F6| 99 97 ØØ                         STA     CSSTV,Y      ; STORE DATA BYTE
F1F9| 45 89                            EOR     CSUM
F1FB| 88                               DEY
F1FC| 1ØE7                             BPL     RDAFLD       ; LOOP ON 4 DATA BYTES.
F1FE| A8                               TAY                  ; IF FINAL CHECKSUM
F1FF| DØB6                             BNE     RDERR        ; NONZERO, THEN ERROR
F2Ø1| BD 8CCØ               RDA6       LDA     Q6L,X        ; FIRST BIT SLIP NIBBL
F2Ø4| 1ØFB                             BPL     RDA6         ; *** NO PAGE CROSS! ***
F2Ø6| C9 DE                            CMP     #ØDE
F2Ø8| DØAD                             BNE     RDERR        ; ERROR IF NONMATCH
F2ØA| EA                               NOP                  ; DELAY
F2ØB| BD 8CCØ               RDA7       LDA     Q6L,X        ; SECOND BIT-SLIP NIBL
F2ØE| 1ØFB                             BPL     RDA7         ; *** NO PAGE CROSS! ***
F21Ø| C9 AA                            CMP     #ØAA
F212| DØA3                             BNE     RDERR        ; ERROR IF NOMATCH
F214| 18                    RDEXIT     CLC                  ; CLEAR CARRY ON
F215| 6Ø                    WEXIT      RTS                  ; NORMAL READ EXITS.
F216|                       ;
F216|                       ;***********************
F216|                       ;                      *
F216|                       ;      WRITE SUBR      *
F216|                       ;  (16-SECTOR FORMAT)  *
F216|                       ;                      *
F216|                       ;***********************
F216|                       ;                      *
F216|                       ;   WRITES DATA FROM   *
F216|                       ;    NBUF1 AND NBUF2   *
F216|                       ;                      *
F216|                       ;  FIRST NBUF2,        *
F216|                       ;      HIGH TO LOW.    *
F216|                       ;  THEN NBUF1,         *
F216|                       ;      LOW TO HIGH     *
F216|                       ;                      *
F216|                       ;  ---- ON ENTRY ----  *
F216|                       ;                      *
F216|                       ;   X-REG  SLOTNUM     *
F216|                       ;        TIMES $1Ø     *
F216|                       ;                      *
F216|                       ;                      *
F216|                       ;  ---- ON EXIT ----   *
F216|                       ;                      *
F216|                       ;  CARRY SET IF ERROR. *
F216|                       ;   (W PROT VIOLATION) *
F216|                       ;                      *
F216|                       ;  IF NO ERROR:        *
F216|                       ;                      *
F216|                       ;    A-REG UNCERTAIN.  *
F216|                       ;    X-REG UNCHANGED.  *
F216|                       ;    Y-REG HOLDS $ØØ.  *
F216|                       ;    CARRY CLEAR.      *
```



```
F216|                       ;                      *
F216|                       ;  ---- ASSUMES ----   *
F216|                       ;                      *
F216|                       ;  1 USEC CYCLE TIME   *
F216|                       ;                      *
F216|                       ;***********************
F216|                       ;
F216| 38                    WRITE16    SEC                  ; ANTICIPATE WPROT ERR.
F217| B8                               CLV                  ; TO INDICATE WRITE PROTECT ERROR INSTEAD OF
F218|                                                       ; INTERRUPT
F218| BD 8DCØ                          LDA     Q6H,X
F21B| BD 8ECØ                          LDA     Q7L,X        ; SENSE WPROT FLAG.
F21E| 3ØF5                             BMI     WEXIT        ; BRANCH IF WRITE PROTECTED
F22Ø| A9 FF                 WRIT1      LDA     #ØFF         ; SYNC DATA.
F222| 9D 8FCØ                          STA     Q7H,X        ; (5) GOTO WRITE MODE
F225| 1D 8CCØ                          ORA     Q6L,X        ; (4)
F228| AØ Ø4                            LDY     #Ø4          ; (2) FOR FIVE NIBLS.
F22A| EA                               NOP                  ; (2)
F22B| 48                               PHA                  ; (4)
F22C| 68                               PLA                  ; (3)
F22D| 48                    WSYNC      PHA                  ; (4) EXACT TIMING
F22E| 68                               PLA                  ; (3)
F22F| 2Ø BBF2                          JSR     WNIBL7       ; (13,9,6) WRITE SYNC
F232| 88                               DEY                  ; (2)
F233| DØF8                             BNE     WSYNC        ; (2*)  MUST NOT CROSS PAGE!
F235| A9 D5                            LDA     #ØD5         ; (2)  1ST DATA MARK
F237| 2Ø BAF2                          JSR     WNIBL9       ; (15,9,6)
F23A| A9 AA                            LDA     #ØAA         ; (2)  2ND DATA MARK
F23C| 2Ø BAF2                          JSR     WNIBL9       ; (15,9,6)
F23F| A9 AD                            LDA     #ØAD         ; (2)  3RD DATA MARK
F241| 2Ø BAF2                          JSR     WNIBL9       ; (15,9,6)
F244| AØ 55                            LDY     #55          ; (2) NBUF2 INDEX
F246| EA                               NOP                  ; (2) FOR TIMING
F247| EA                               NOP                  ; (2)
F248| EA                               NOP                  ; (2)
F249| DØØ8                             BNE     VRYFRST      ; (3) BRANCH ALWAYS
F24B| AD EFFF               WINTRPT    LDA     INTERUPT     ; (4) POLL INTERRUPT LINE
F24E| Ø5 8B                            ORA     IMASK        ; (3)
F25Ø| 38                               SEC                  ; (2)
F251| 1Ø57                             BPL     SERVICE      ; (2) BRANCH IF INTERRUPT HAS OCCURED
F253| 3ØØØ                  VRYFRST    BMI     WRTFRST      ; (3) FOR TIMING.
F255| B9 Ø2Ø3               WRTFRST    LDA     NBUF2,Y      ; (4)
F258| 9D 8DCØ                          STA     Q6H,X        ; (5) STORE ENCODED BYTE
F25B| BD 8CCØ                          LDA     Q6L,X        ; (4) TIME MUST = 32 US PER BYTE!
F25E| 88                               DEY                  ; (2)
F25F| 1ØEA                             BPL     WINTRPT      ; (3) (2 IF BRANCH NOT TAKEN)
F261| 98                               TYA                  ; (2) INSURE NO INTERRUPT THIS BYTE
F262| 3ØØ3                             BMI     WMIDLE       ; (3) BRANCH ALWAYS.
F264| AD EFFF               WNTRPT1    LDA     INTERUPT     ; (4) POLL INTERRUPT LINE
F267| Ø5 8B                 WMIDLE     ORA     IMASK        ; (3)
F269| 38                               SEC                  ; (2)
F26A| 3ØØ2                             BMI     WDATA2       ; (3) BRANCH IF NO INTERRUPT
F26C| 1Ø3C                             BPL     SERVICE      ; GO SERVICE INTERRUPT.
F26E| C8                    WDATA2     INY                  ; (2)
F26F| B9 ØØØ2                          LDA     NBUF1,Y      ; (4)
F272| 9D 8DCØ                          STA     Q6H,X        ; (5) STORE ENCODED BYTE
F275| BD 8CCØ                          LDA     Q6L,X        ; (4)
F278| CØ E4                            CPY     #ØE4         ; (2) WITHIN 1 MS OF COMPLETION?
F27A| DØE8                             BNE     WNTRPT1      ; (3) (2) NO KEEP WRITTING AND POLLING.
F27C| EA                               NOP                  ; (2)
F27D| C8                               INY                  ; (2)
F27E| EA                    WDATA3     NOP                  ; (2)
F27F| EA                               NOP                  ; (2)
F28Ø| 48                               PHA                  ; (4)
F281| 68                               PLA                  ; (3)
F282| B9 ØØØ2                          LDA     NBUF1,Y      ; (4) WRITE LAST OF ENCODED BYTES
F285| 9D 8DCØ                          STA     Q6H,X        ; (5) WITHOUT POLLING INTERRUPTS.
F288| BD 8CCØ                          LDA     Q6L,X        ; (4)
F28B| A5 96                            LDA     CKSUM        ; (3) NORMALLY FOR TIMING
F28D| C8                               INY                  ; (2)
F28E| DØEE                             BNE     WDATA3       ; (3) (2)
F29Ø| FØØØ                             BEQ     WRCKSUM      ; (3) BRANCH ALWAYS
F292| 2Ø BBF2               WRCKSUM    JSR     WNIBL7       ; (13,9,6) GO WRITE CHECK SUM!!
F295| 48                               PHA                  ; (3)
F296| 68                               PLA                  ; (4)
F297| B9 CØF3               WRBITSLMK  LDA     BITSLIPMK,Y  ; (4) LOAD BIT SLIP MARK
F29A| 2Ø BDF2                          JSR     WNIBL        ; (6,9,6)
F29D| C8                               INY                  ; (2)
F29E| CØ Ø4                            CPY     #Ø4          ; (2)
F2AØ| DØF5                             BNE     WRBITSLMK    ; (2) (3)
F2A2| 18                               CLC                  ; (2)
F2A3| BD 8ECØ               NOWRITE    LDA     Q7L,X        ; OUT OF WRITE MODE.
F2A6| BD 8CCØ                          LDA     Q6L,X        ; TO READ MODE.
F2A9| 6Ø                               RTS                  ; RETURN FROM WRITE.
F2AA|                       ;
F2AA| 2C 54F3               SERVICE    BIT     SEV          ; SET VFLAG TO INDICATE INTERRUPT
F2AD| 2Ø A3F2                          JSR     NOWRITE      ; TAKE IT OUT OF WRITE MODE!
F2BØ| A5 8F                            LDA     Ø8F
F2B2| 1ØØ2                             BPL     $Ø1Ø
F2B4| 85 8B                            STA     IMASK
```

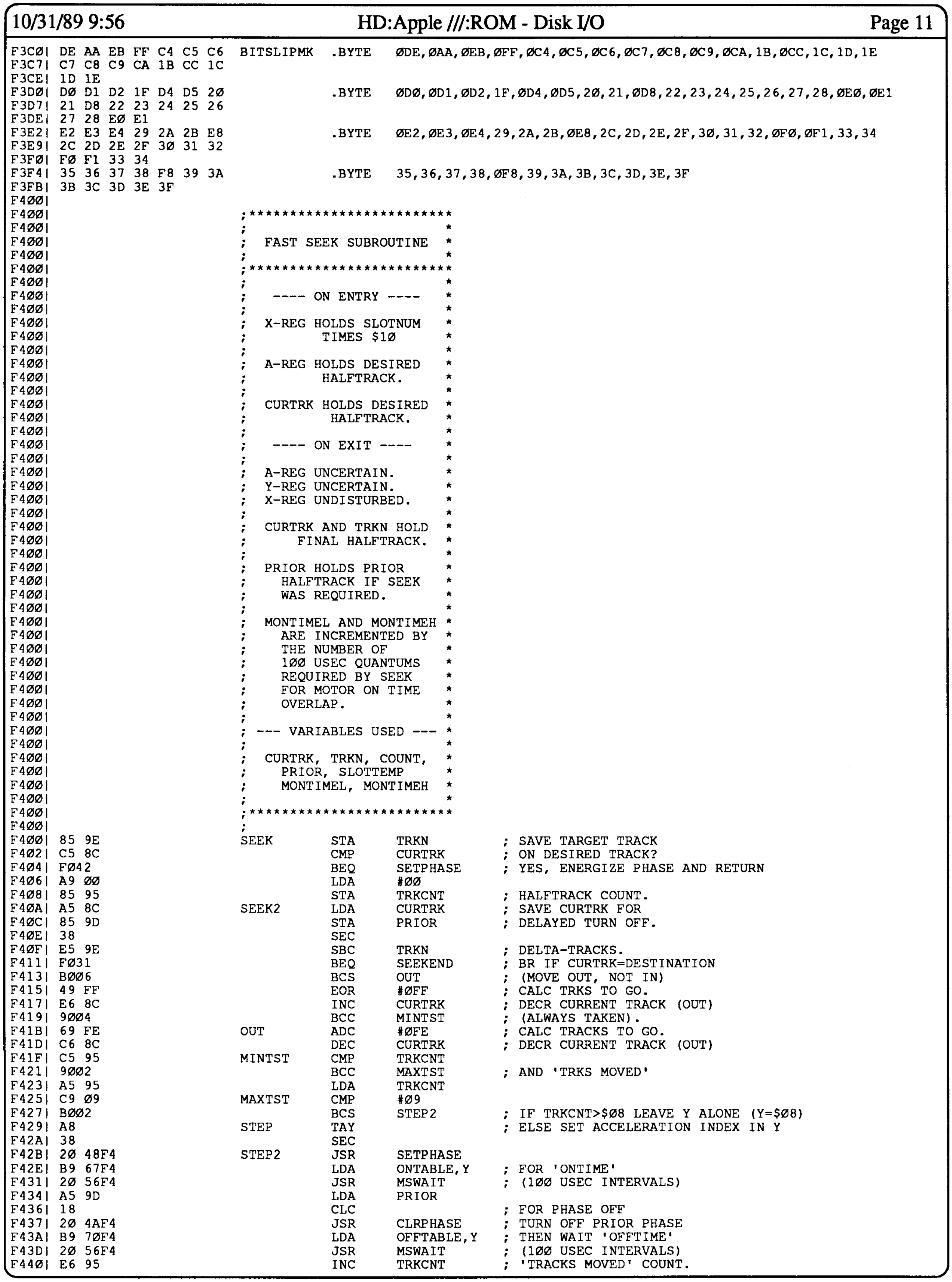


```
F2B6| C6 8F                 $Ø1Ø       DEC     Ø8F
F2B8| 58                               CLI                 ; COULD NOT HAVE GOT HERE WITHOUT CLI OK
F2B9| 6Ø                               RTS
F2BA|                       ;
F2BA|                       ;****************************
F2BA|                       ;                           *
F2BA|                       ;   7-BIT NIBL WRITE SUBRS  *
F2BA|                       ;                           *
F2BA|                       ;   A-REG OR'D PRIOR EXIT   *
F2BA|                       ;       CARRY CLEARED       *
F2BA|                       ;                           *
F2BA|                       ;****************************
F2BA|                       ;
F2BA| 18                    WNIBL9     CLC                 ; (2) 9 CYCLES, THEN WRITE
F2BB| 48                    WNIBL7     PHA                 ; (3) 7 CYCLES, THEN WRITE
F2BC| 68                               PLA                 ; (4)
F2BD| 9D 8DCØ               WNIBL      STA     Q6H,X       ; (5) NIBL WRITE SUB
F2CØ| 1D 8CCØ                          ORA     Q6L,X       ; (4) CLOBBERS ACC. NOT CARRY
F2C3| 6Ø                               RTS
F2C4|                       ;
F2C4|                       ;****************************
F2C4|                       ;                           *
F2C4|                       ;    PRENIBILIZE SUBR       *
F2C4|                       ;   (16-SECTOR FORMAT)      *
F2C4|                       ;                           *
F2C4|                       ;****************************
F2C4|                       ;                           *
F2C4|                       ;  CONVERTS 256 BYTES OF    *
F2C4|                       ;  USER DATA IN (BUF) INTO  *
F2C4|                       ;  ENCODED BYTES TO BE      *
F2C4|                       ;  WRITTEN DIRECTLY TO DISK *
F2C4|                       ;  ENCODED CHECK SUM IN     *
F2C4|                       ;  ZERO PAGE 'CKSUM'        *
F2C4|                       ;                           *
F2C4|                       ;   ---- ON ENTRY ----      *
F2C4|                       ;                           *
F2C4|                       ;  BUF IS 2-BYTE POINTER    *
F2C4|                       ;    TO 256 BYTES OF USER   *
F2C4|                       ;    DATA.                  *
F2C4|                       ;                           *
F2C4|                       ;  A-REG CHECK SUM.         *
F2C4|                       ;  X-REG UNCERTAIN          *
F2C4|                       ;  Y-REG HOLDS Ø.           *
F2C4|                       ;  CARRY SET.               *
F2C4|                       ;                           *
F2C4|                       ;****************************
F2C4|                       ;
F2C4| A2 Ø2                 PRENIB16   LDX     #Ø2         ; START NBUF2 INDEX.
F2C6| AØ ØØ                            LDY     #ØØ         ; START USER BUF INDEX.
F2C8| 88                    PRENIB1    DEY                 ; NEXT USER BYTE
F2C9| B1 9B                            LDA     (BUF),Y
F2CB| 4A                               LSR     A           ; SHIFT TWO BITS OF
F2CC| 3E Ø1Ø3                          ROL     NBUF2-1,X   ; CURRENT USER BYTE
F2CF| 4A                               LSR     A           ; INTO CURRENT NBUF2
F2DØ| 3E Ø1Ø3                          ROL     NBUF2-1,X   ; BYTE.
F2D3| 99 Ø1Ø2                          STA     NBUF1+1,Y   ; (6 BITS LEFT).
F2D6| E8                               INX                 ; FROM Ø TO $55
F2D7| EØ 56                            CPX     #56
F2D9| 9ØED                             BCC     PRENIB1     ; BR IF NO WRAPAROUND
F2DB| A2 ØØ                            LDX     #ØØ         ; RESET NBUF2 INDEX
F2DD| 98                               TYA                 ; USER BUF INDEX
F2DE| DØE8                             BNE     PRENIB1     ; (DONE IF ZERO)
F2EØ| AØ 56                            LDY     #56         ; (ACC=Ø FOR CHECK SUM)
F2E2| 59 ØØØ3               PRENIB3    EOR     NBUF2-2,Y   ; COMBINE WITH PREVIOUS
F2E5| 29 3F                 PRENIB2    AND     #Ø3F        ; STRIP GARBAGE BITS
F2E7| AA                               TAX                 ; TO FORM RUNNING CHECK SUM
F2E8| BD 55F3                          LDA     NIBL,X      ; GET ENCODED EQUIV.
F2EB| 99 Ø1Ø3                          STA     NBUF2-1,Y   ; REPLACE PREVIOUS
F2EE| B9 ØØØ3                          LDA     NBUF2-2,Y   ; RESTORE ACTUAL PREVIOUS
F2F1| 88                               DEY
F2F2| DØEE                             BNE     PRENIB3     ; LOOP UNTIL ALL OF NBUF2 IS CONVERTED.
F2F4| 29 3F                            AND     #3F
F2F6| 59 Ø1Ø2               PRENIB4    EOR     NBUF1+1,Y   ; NOW DO THE SAME FOR
F2F9| AA                               TAX                 ; NIBBLE BUFFER 1
F2FA| BD 55F3                          LDA     NIBL,X      ; TO DO ANY BACK TRACKING (NBUF1-1)
F2FD| 99 ØØØ2                          STA     NBUF1,Y
F3ØØ| B9 Ø1Ø2                          LDA     NBUF1+1,Y   ; RECOVER THAT WHICH IS NOW 'PREVIOUS'
F3Ø3| C8                               INY
F3Ø4| DØFØ                             BNE     PRENIB4
F3Ø6| AA                               TAX                 ; USE LAST AS CHECK SUM
F3Ø7| BD 55F3                          LDA     NIBL,X
F3ØA| 85 96                            STA     CKSUM
F3ØC| 4C 4CF3                          JMP     SET1MEG     ; ALL DONE.
F3ØF|                       ;
F3ØF|                       ;**************************
F3ØF|                       ;                         *
F3ØF|                       ;    POSTNIBLIZE SUBR     *
F3ØF|                       ;    16-SECTOR FORMAT     *
F3ØF|                       ;                         *
F3ØF|                       ;**************************
```



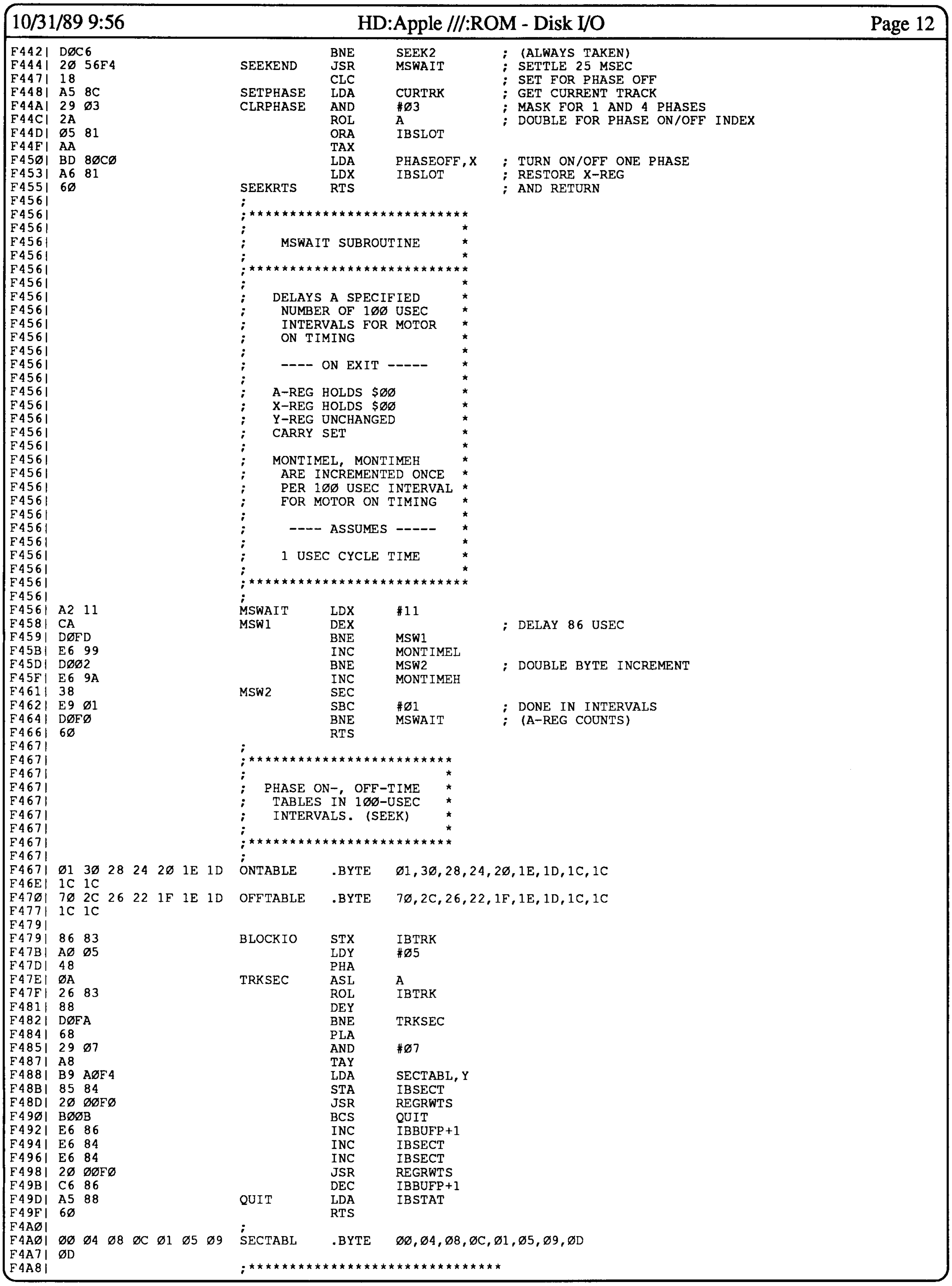
```
F30F|                      ;
F30F| 38                   POSTNIB16 SEC
F310| AØ 55                          LDY     #55          ; FIRST CONVERT TO 6 BIT NIBBLES
F312| A9 ØØ                          LDA     #ØØ          ; INIT CHECK SUM
F314| BE Ø2Ø3              PNIBL1    LDX     NBUF2,Y      ; GET ENCODED BYTE
F317| 5D ØØF3                        EOR     DNIBL,X
F31A| 3Ø3Ø                           BMI     SET1MEG      ; SET 1 MHZ
F31C| 99 Ø2Ø3                        STA     NBUF2,Y      ; REPLACE WITH 6 BIT EQUIV.
F31F| 88                             DEY
F32Ø| 1ØA6                           BPL     PRENIB1      ; LOOP UNTIL DONE WITH NIBBLE BUFFER 2
F322| C8                             INY                  ; NOW Y=Ø
F323| BE ØØØ2              PNIBL2    LDX     NBUF1,Y      ; DO THE SAME WITH
F326| 5D ØØF3                        EOR     DNIBL,X
F329| 99 ØØØ2                        STA     NBUF1,Y      ; NIBBLE BUFFER 1
F32C| C8                             INY                  ; DO ALL 256 BYTES
F32D| DØF4                           BNE     PNIBL2
F32F| A6 96                          LDX     CKSUM        ; MAKE SURE CHECK SUM MATCHES
F331| 5D ØØF3                        EOR     DNIBL,X      ; BETTER BE ZERO
F334| DØ16                           BNE     POSTERR      ; BRANCH IF IT IS
F336| A2 56                POST1     LDX     #56          ; INIT NBUF2 INDEX
F338| CA                   POST2     DEX                  ; NBUF IDX $55 TO $ØØ
F339| 3ØFB                           BMI     POST1        ; WRAPAROUND IF NEG
F33B| B9 ØØØ2                        LDA     NBUF1,Y
F33E| 5E Ø2Ø3                        LSR     NBUF2,X      ; SHIFT 2 BITS FROM
F341| 2A                             ROL     A            ; CURRENT NBUF2 NIBL
F342| 5E Ø2Ø3                        LSR     NBUF2,X      ; CURRENT NBUF1
F345| 2A                             ROL     A            ; NIBL.
F346| 91 9B                          STA     (BUF),Y      ; BYTE OF USER DATA
F348| C8                             INY                  ; NEXT USER BYTE
F349| DØED                           BNE     POST2
F34B| 18                             CLC                  ; GOOD DATA
F34C| F34C                 POSTERR   .EQU    *
F34C| AD DFFF              SET1MEG   LDA     ENVIRON
F34F| Ø9 8Ø                          ORA     #ONEMEG      ; SET TO ONE MEGAHERTZ CLOCK RATE
F351| 8D DFFF                        STA     ENVIRON
F354| 6Ø                   SEV       RTS                  ; (SEV USED TO SET VFLAG)
F355|                      ;
F355|                      ;*************************
F355|                      ;                        *
F355|                      ;     6-BIT TO 7-BIT     *
F355|                      ;  NIBL CONVERSION TABLE *
F355|                      ;                        *
F355|                      ;*************************
F355|                      ;                        *
F355|                      ;   CODES WITH MORE THAN *
F355|                      ;   ONE PAIR OF ADJACENT *
F355|                      ;    ZEROES OR WITH NO   *
F355|                      ;   ADJACENT ONES (EXCEPT *
F355|                      ;     B7) ARE EXCLUDED.  *
F355|                      ;                        *
F355|                      ;*************************
F355|                      ;
F355| 96 97 9A 9B 9D 9E 9F NIBL   .BYTE 96,97,9A,9B,9D,9E,9F,ØA6,ØA7,ØAB,ØAC,ØAD,ØAE,ØAF,ØB2,ØB3,ØB4,ØB5
F35C| A6 A7 AB AC AD AE AF
F363| B2 B3 B4 B5
F367| B6 B7 B9 BA BB BC BD        .BYTE ØB6,ØB7,ØB9,ØBA,ØBB,ØBC,ØBD,ØBE,ØBF,ØCB,ØCD,ØCE,ØCF,ØD3,ØD6,ØD7
F36E| BE BF CB CD CE CF D3
F375| D6 D7
F377| D9 DA DB DC DD DE DF        .BYTE ØD9,ØDA,ØDB,ØDC,ØDD,ØDE,ØDF,ØE5,ØE6,ØE7,ØE9,ØEA,ØEB,ØEC,ØED,ØEE
F37E| E5 E6 E7 E9 EA EB EC
F385| ED EE
F387| EF F2 F3 F4 F5 F6 F7        .BYTE ØEF,ØF2,ØF3,ØF4,ØF5,ØF6,ØF7,ØF9,ØFA,ØFB,ØFC,ØFD,ØFE,ØFF
F38E| F9 FA FB FC FD FE FF
F395|                      ;
F395|                      ;*************************
F395|                      ;                        *
F395|                      ;     7-BIT TO 6-BIT     *
F395|                      ;    'DENIBLIZE' TABL    *
F395|                      ;    (16-SECTOR FORMAT)  *
F395|                      ;                        *
F395|                      ;       VALID CODES      *
F395|                      ;     $96 TO $FF ONLY.   *
F395|                      ;                        *
F395|                      ;                        *
F395|                      ;   CODES WITH MORE THAN *
F395|                      ;   ONE PAIR OF ADJACENT *
F395|                      ;    ZEROES OR WITH NO   *
F395|                      ;   ADJACENT ONES (EXCEPT *
F395|                      ;    BIT 7) ARE EXCLUDED *
F395|                      ;*************************
F395|                      ;
F395| F3ØØ                 DNIBL     .EQU    REGRWTS+3ØØ
F395| Ø1 ØØ Ø1                       .BYTE   Ø1,ØØ,Ø1
F398| 98 99 Ø2 Ø3 9C Ø4 Ø5           .BYTE   98,99,Ø2,Ø3,9C,Ø4,Ø5,Ø6,ØAØ,ØA1,ØA2,ØA3,ØA4,ØA5,Ø7,Ø8,ØA8
F39F| Ø6 AØ A1 A2 A3 A4 A5
F3A6| Ø7 Ø8 A8
F3A9| A9 AA Ø9 ØA ØB ØC ØD           .BYTE   ØA9,ØAA,Ø9,ØA,ØB,ØC,ØD,ØBØ,ØB1,ØE,ØF,1Ø,11,12,13,ØB8,14,15
F3BØ| BØ B1 ØE ØF 1Ø 11 12
F3B7| 13 B8 14 15
F3BB| 16 17 18 19 1A                 .BYTE   16,17,18,19,1A
```

“APPLE_PAT_4_383_296_C_12” 192 KB 2000-02-28 dpi: 300h x 300v pix: 2318h x 3097v



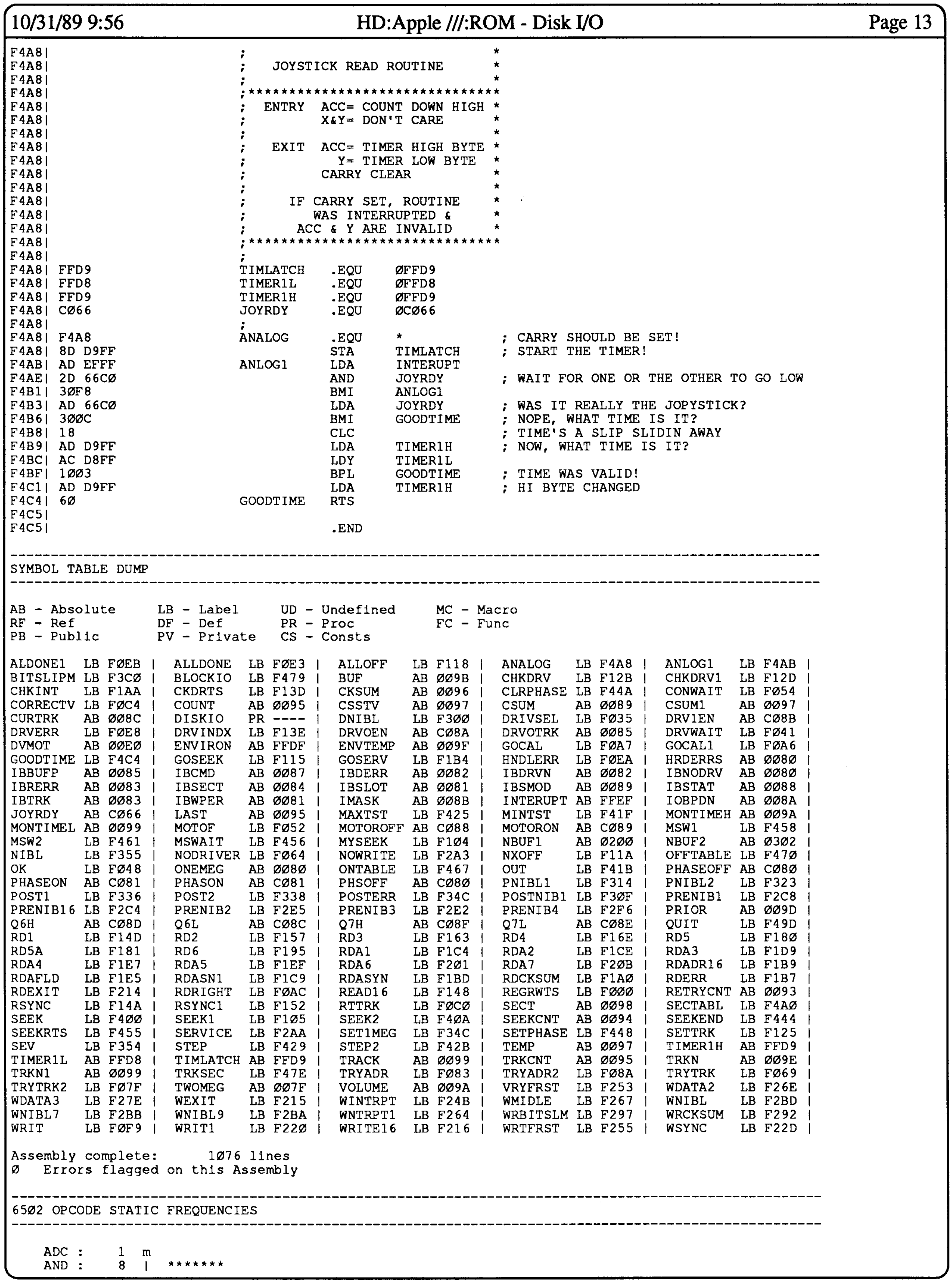
10/31/89 9:56 HD:Apple ///:ROM - Disk I/O Page 11

```
F3CØ| DE AA EB FF C4 C5 C6  BITSLIPMK  .BYTE   ØDE,ØAA,ØEB,ØFF,ØC4,ØC5,ØC6,ØC7,ØC8,ØC9,ØCA,1B,ØCC,1C,1D,1E
F3C7| C7 C8 C9 CA 1B CC 1C
F3CE| 1D 1E
F3DØ| DØ D1 D2 1F D4 D5 2Ø             .BYTE   ØDØ,ØD1,ØD2,1F,ØD4,ØD5,2Ø,21,ØD8,22,23,24,25,26,27,28,ØEØ,ØE1
F3D7| 21 D8 22 23 24 25 26
F3DE| 27 28 EØ E1
F3E2| E2 E3 E4 29 2A 2B E8             .BYTE   ØE2,ØE3,ØE4,29,2A,2B,ØE8,2C,2D,2E,2F,3Ø,31,32,ØFØ,ØF1,33,34
F3E9| 2C 2D 2E 2F 3Ø 31 32
F3FØ| FØ F1 33 34
F3F4| 35 36 37 38 F8 39 3A             .BYTE   35,36,37,38,ØF8,39,3A,3B,3C,3D,3E,3F
F3FB| 3B 3C 3D 3E 3F
F4ØØ|
F4ØØ|                       ;*************************
F4ØØ|                       ;                        *
F4ØØ|                       ;  FAST SEEK SUBROUTINE  *
F4ØØ|                       ;                        *
F4ØØ|                       ;*************************
F4ØØ|                       ;                        *
F4ØØ|                       ;   ---- ON ENTRY ----   *
F4ØØ|                       ;                        *
F4ØØ|                       ;  X-REG HOLDS SLOTNUM   *
F4ØØ|                       ;         TIMES $1Ø      *
F4ØØ|                       ;                        *
F4ØØ|                       ;  A-REG HOLDS DESIRED   *
F4ØØ|                       ;         HALFTRACK.     *
F4ØØ|                       ;                        *
F4ØØ|                       ;  CURTRK HOLDS DESIRED  *
F4ØØ|                       ;          HALFTRACK.    *
F4ØØ|                       ;                        *
F4ØØ|                       ;   ---- ON EXIT ----    *
F4ØØ|                       ;                        *
F4ØØ|                       ;  A-REG UNCERTAIN.      *
F4ØØ|                       ;  Y-REG UNCERTAIN.      *
F4ØØ|                       ;  X-REG UNDISTURBED.    *
F4ØØ|                       ;                        *
F4ØØ|                       ;  CURTRK AND TRKN HOLD  *
F4ØØ|                       ;      FINAL HALFTRACK.  *
F4ØØ|                       ;                        *
F4ØØ|                       ;  PRIOR HOLDS PRIOR     *
F4ØØ|                       ;    HALFTRACK IF SEEK   *
F4ØØ|                       ;    WAS REQUIRED.       *
F4ØØ|                       ;                        *
F4ØØ|                       ;  MONTIMEL AND MONTIMEH *
F4ØØ|                       ;    ARE INCREMENTED BY  *
F4ØØ|                       ;    THE NUMBER OF       *
F4ØØ|                       ;    1ØØ USEC QUANTUMS   *
F4ØØ|                       ;    REQUIRED BY SEEK    *
F4ØØ|                       ;    FOR MOTOR ON TIME   *
F4ØØ|                       ;    OVERLAP.            *
F4ØØ|                       ;                        *
F4ØØ|                       ; --- VARIABLES USED --- *
F4ØØ|                       ;                        *
F4ØØ|                       ;  CURTRK, TRKN, COUNT,  *
F4ØØ|                       ;    PRIOR, SLOTTEMP     *
F4ØØ|                       ;    MONTIMEL, MONTIMEH  *
F4ØØ|                       ;                        *
F4ØØ|                       ;*************************
F4ØØ|                       ;
F4ØØ| 85 9E                 SEEK       STA     TRKN         ; SAVE TARGET TRACK
F4Ø2| C5 8C                            CMP     CURTRK       ; ON DESIRED TRACK?
F4Ø4| FØ42                             BEQ     SETPHASE     ; YES, ENERGIZE PHASE AND RETURN
F4Ø6| A9 ØØ                            LDA     #ØØ
F4Ø8| 85 95                            STA     TRKCNT       ; HALFTRACK COUNT.
F4ØA| A5 8C                 SEEK2      LDA     CURTRK       ; SAVE CURTRK FOR
F4ØC| 85 9D                            STA     PRIOR        ; DELAYED TURN OFF.
F4ØE| 38                               SEC
F4ØF| E5 9E                            SBC     TRKN         ; DELTA-TRACKS.
F411| FØ31                             BEQ     SEEKEND      ; BR IF CURTRK=DESTINATION
F413| BØØ6                             BCS     OUT          ; (MOVE OUT, NOT IN)
F415| 49 FF                            EOR     #ØFF         ; CALC TRKS TO GO.
F417| E6 8C                            INC     CURTRK       ; DECR CURRENT TRACK (OUT)
F419| 9ØØ4                             BCC     MINTST       ; (ALWAYS TAKEN).
F41B| 69 FE                 OUT        ADC     #ØFE         ; CALC TRACKS TO GO.
F41D| C6 8C                            DEC     CURTRK       ; DECR CURRENT TRACK (OUT)
F41F| C5 95                 MINTST     CMP     TRKCNT
F421| 9ØØ2                             BCC     MAXTST       ; AND 'TRKS MOVED'
F423| A5 95                            LDA     TRKCNT
F425| C9 Ø9                 MAXTST     CMP     #Ø9
F427| BØØ2                             BCS     STEP2        ; IF TRKCNT>$Ø8 LEAVE Y ALONE (Y=$Ø8)
F429| A8                    STEP       TAY                  ; ELSE SET ACCELERATION INDEX IN Y
F42A| 38                               SEC
F42B| 2Ø 48F4               STEP2      JSR     SETPHASE
F42E| B9 67F4                          LDA     ONTABLE,Y    ; FOR 'ONTIME'
F431| 2Ø 56F4                          JSR     MSWAIT       ; (1ØØ USEC INTERVALS)
F434| A5 9D                            LDA     PRIOR
F436| 18                               CLC                  ; FOR PHASE OFF
F437| 2Ø 4AF4                          JSR     CLRPHASE     ; TURN OFF PRIOR PHASE
F43A| B9 7ØF4                          LDA     OFFTABLE,Y   ; THEN WAIT 'OFFTIME'
F43D| 2Ø 56F4                          JSR     MSWAIT       ; (1ØØ USEC INTERVALS)
F44Ø| E6 95                            INC     TRKCNT       ; 'TRACKS MOVED' COUNT.
```



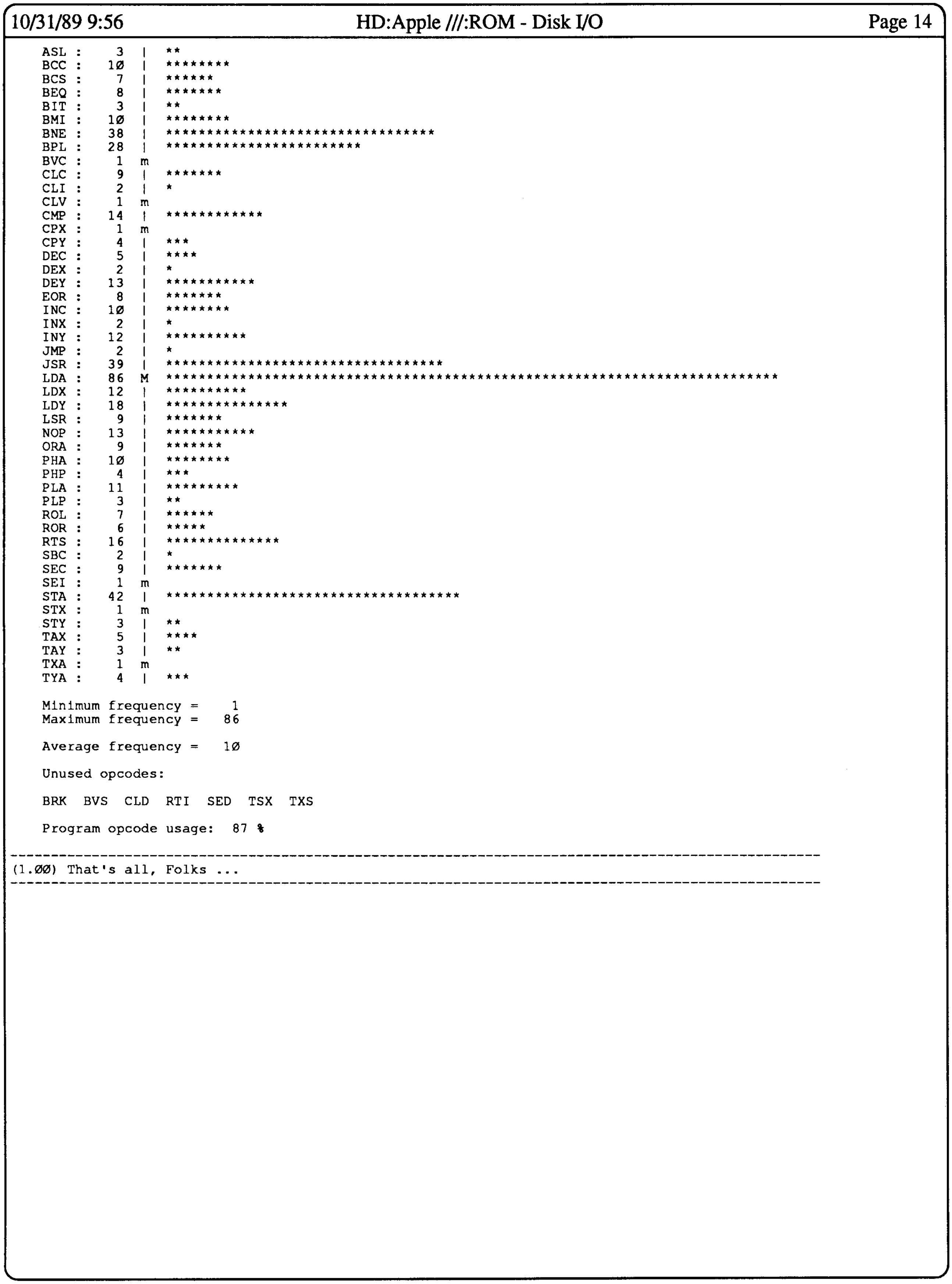
```
F442| DØC6                         BNE    SEEK2        ; (ALWAYS TAKEN)
F444| 2Ø 56F4          SEEKEND     JSR    MSWAIT       ; SETTLE 25 MSEC
F447| 18                           CLC                 ; SET FOR PHASE OFF
F448| A5 8C            SETPHASE    LDA    CURTRK       ; GET CURRENT TRACK
F44A| 29 Ø3            CLRPHASE    AND    #Ø3          ; MASK FOR 1 AND 4 PHASES
F44C| 2A                           ROL    A            ; DOUBLE FOR PHASE ON/OFF INDEX
F44D| Ø5 81                        ORA    IBSLOT
F44F| AA                           TAX
F45Ø| BD 8ØCØ                      LDA    PHASEOFF,X   ; TURN ON/OFF ONE PHASE
F453| A6 81                        LDX    IBSLOT       ; RESTORE X-REG
F455| 6Ø               SEEKRTS     RTS                 ; AND RETURN
F456|                  ;
F456|                  ;**************************
F456|                  ;                         *
F456|                  ;    MSWAIT SUBROUTINE    *
F456|                  ;                         *
F456|                  ;**************************
F456|                  ;                         *
F456|                  ;   DELAYS A SPECIFIED    *
F456|                  ;    NUMBER OF 1ØØ USEC   *
F456|                  ;    INTERVALS FOR MOTOR  *
F456|                  ;    ON TIMING            *
F456|                  ;                         *
F456|                  ;    ---- ON EXIT -----   *
F456|                  ;                         *
F456|                  ;   A-REG HOLDS $ØØ       *
F456|                  ;   X-REG HOLDS $ØØ       *
F456|                  ;   Y-REG UNCHANGED       *
F456|                  ;   CARRY SET             *
F456|                  ;                         *
F456|                  ;   MONTIMEL, MONTIMEH    *
F456|                  ;    ARE INCREMENTED ONCE *
F456|                  ;    PER 1ØØ USEC INTERVAL *
F456|                  ;    FOR MOTOR ON TIMING  *
F456|                  ;                         *
F456|                  ;     ---- ASSUMES -----  *
F456|                  ;                         *
F456|                  ;    1 USEC CYCLE TIME    *
F456|                  ;                         *
F456|                  ;**************************
F456|                  ;
F456| A2 11            MSWAIT      LDX    #11
F458| CA               MSW1        DEX                 ; DELAY 86 USEC
F459| DØFD                         BNE    MSW1
F45B| E6 99                        INC    MONTIMEL
F45D| DØØ2                         BNE    MSW2         ; DOUBLE BYTE INCREMENT
F45F| E6 9A                        INC    MONTIMEH
F461| 38               MSW2        SEC
F462| E9 Ø1                        SBC    #Ø1          ; DONE IN INTERVALS
F464| DØFØ                         BNE    MSWAIT       ; (A-REG COUNTS)
F466| 6Ø                           RTS
F467|                  ;
F467|                  ;************************
F467|                  ;                       *
F467|                  ;  PHASE ON-, OFF-TIME  *
F467|                  ;   TABLES IN 1ØØ-USEC  *
F467|                  ;   INTERVALS. (SEEK)   *
F467|                  ;                       *
F467|                  ;************************
F467|                  ;
F467| Ø1 3Ø 28 24 2Ø 1E 1D  ONTABLE  .BYTE  Ø1,3Ø,28,24,2Ø,1E,1D,1C,1C
F46E| 1C 1C
F47Ø| 7Ø 2C 26 22 1F 1E 1D  OFFTABLE .BYTE  7Ø,2C,26,22,1F,1E,1D,1C,1C
F477| 1C 1C
F479|
F479| 86 83            BLOCKIO     STX    IBTRK
F47B| AØ Ø5                        LDY    #Ø5
F47D| 48                           PHA
F47E| ØA               TRKSEC      ASL    A
F47F| 26 83                        ROL    IBTRK
F481| 88                           DEY
F482| DØFA                         BNE    TRKSEC
F484| 68                           PLA
F485| 29 Ø7                        AND    #Ø7
F487| A8                           TAY
F488| B9 AØF4                      LDA    SECTABL,Y
F48B| 85 84                        STA    IBSECT
F48D| 2Ø ØØFØ                      JSR    REGRWTS
F49Ø| BØØB                         BCS    QUIT
F492| E6 86                        INC    IBBUFP+1
F494| E6 84                        INC    IBSECT
F496| E6 84                        INC    IBSECT
F498| 2Ø ØØFØ                      JSR    REGRWTS
F49B| C6 86                        DEC    IBBUFP+1
F49D| A5 88            QUIT        LDA    IBSTAT
F49F| 6Ø                           RTS
F4AØ|                  ;
F4AØ| ØØ Ø4 Ø8 ØC Ø1 Ø5 Ø9  SECTABL  .BYTE  ØØ,Ø4,Ø8,ØC,Ø1,Ø5,Ø9,ØD
F4A7| ØD
F4A8|                  ;******************************
```

"APPLE_PAT_4_383_296_C_14" 156 KB 2000-02-28 dpi: 300h x 300v pix: 2337h x 3103v



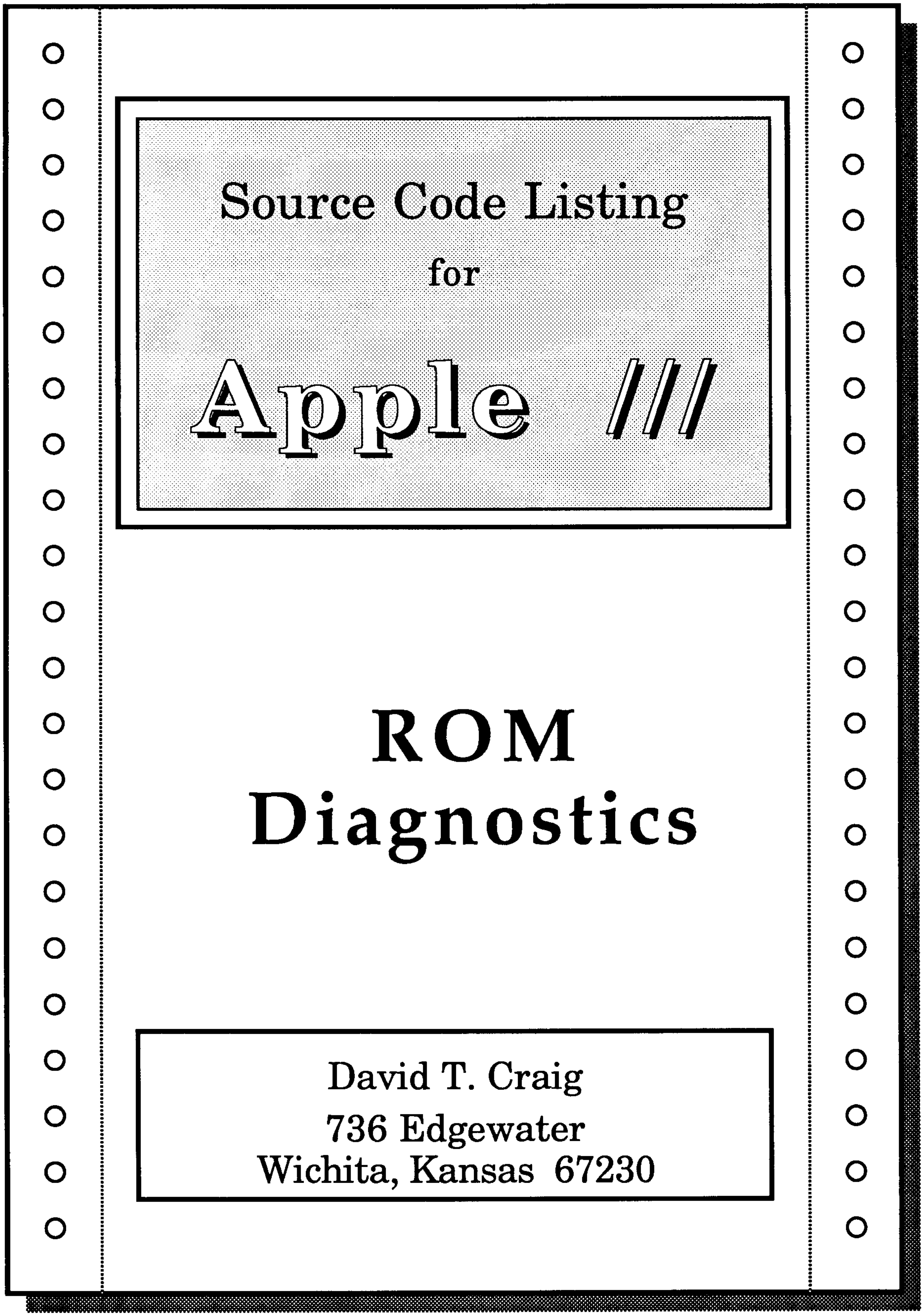
```
F4A8|                          ;                              *
F4A8|                          ;   JOYSTICK READ ROUTINE      *
F4A8|                          ;                              *
F4A8|                          ;*******************************
F4A8|                          ;  ENTRY  ACC= COUNT DOWN HIGH *
F4A8|                          ;         X&Y= DON'T CARE      *
F4A8|                          ;                              *
F4A8|                          ;   EXIT  ACC= TIMER HIGH BYTE *
F4A8|                          ;           Y= TIMER LOW BYTE  *
F4A8|                          ;         CARRY CLEAR          *
F4A8|                          ;                              *
F4A8|                          ;     IF CARRY SET, ROUTINE    *
F4A8|                          ;        WAS INTERRUPTED &     *
F4A8|                          ;      ACC & Y ARE INVALID     *
F4A8|                          ;*******************************
F4A8|                          ;
F4A8| FFD9                     TIMLATCH   .EQU    ØFFD9
F4A8| FFD8                     TIMER1L    .EQU    ØFFD8
F4A8| FFD9                     TIMER1H    .EQU    ØFFD9
F4A8| CØ66                     JOYRDY     .EQU    ØCØ66
F4A8|                          ;
F4A8| F4A8                     ANALOG     .EQU    *             ; CARRY SHOULD BE SET!
F4A8| 8D D9FF                             STA     TIMLATCH      ; START THE TIMER!
F4AB| AD EFFF                  ANLOG1     LDA     INTERUPT
F4AE| 2D 66CØ                             AND     JOYRDY        ; WAIT FOR ONE OR THE OTHER TO GO LOW
F4B1| 3ØF8                                BMI     ANLOG1
F4B3| AD 66CØ                             LDA     JOYRDY        ; WAS IT REALLY THE JOPYSTICK?
F4B6| 3ØØC                                BMI     GOODTIME      ; NOPE, WHAT TIME IS IT?
F4B8| 18                                  CLC                   ; TIME'S A SLIP SLIDIN AWAY
F4B9| AD D9FF                             LDA     TIMER1H       ; NOW, WHAT TIME IS IT?
F4BC| AC D8FF                             LDY     TIMER1L
F4BF| 1ØØ3                                BPL     GOODTIME      ; TIME WAS VALID!
F4C1| AD D9FF                             LDA     TIMER1H       ; HI BYTE CHANGED
F4C4| 6Ø                       GOODTIME   RTS
F4C5|
F4C5|                                     .END

-----------------------------------------------------------------------------------------------------
SYMBOL TABLE DUMP
-----------------------------------------------------------------------------------------------------

AB - Absolute      LB - Label      UD - Undefined      MC - Macro
RF - Ref           DF - Def        PR - Proc           FC - Func
PB - Public        PV - Private    CS - Consts

ALDONE1   LB FØEB |  ALLDONE   LB FØE3 |  ALLOFF    LB F118 |  ANALOG    LB F4A8 |  ANLOG1    LB F4AB |
BITSLIPM  LB F3CØ |  BLOCKIO   LB F479 |  BUF       AB ØØ9B |  CHKDRV    LB F12B |  CHKDRV1   LB F12D |
CHKINT    LB F1AA |  CKDRTS    LB F13D |  CKSUM     AB ØØ96 |  CLRPHASE  LB F44A |  CONWAIT   LB FØ54 |
CORRECTV  LB FØC4 |  COUNT     AB ØØ95 |  CSSTV     AB ØØ97 |  CSUM      AB ØØ89 |  CSUM1     AB ØØ97 |
CURTRK    AB ØØ8C |  DISKIO    PR ---- |  DNIBL     LB F3ØØ |  DRIVSEL   LB FØ35 |  DRV1EN    AB CØ8B |
DRVERR    LB FØE8 |  DRVINDX   LB F13E |  DRVOEN    AB CØ8A |  DRVOTRK   AB ØØ85 |  DRVWAIT   LB FØ41 |
DVMOT     AB ØØEØ |  ENVIRON   AB FFDF |  ENVTEMP   AB ØØ9F |  GOCAL     LB FØA7 |  GOCAL1    LB FØA6 |
GOODTIME  LB F4C4 |  GOSEEK    LB F115 |  GOSERV    LB F1B4 |  HNDLERR   LB FØEA |  HRDERRS   AB ØØ8Ø |
IBBUFP    AB ØØ85 |  IBCMD     AB ØØ87 |  IBDERR    AB ØØ82 |  IBDRVN    AB ØØ82 |  IBNODRV   AB ØØ8Ø |
IBRERR    AB ØØ83 |  IBSECT    AB ØØ84 |  IBSLOT    AB ØØ81 |  IBSMOD    AB ØØ89 |  IBSTAT    AB ØØ88 |
IBTRK     AB ØØ83 |  IBWPER    AB ØØ81 |  IMASK     AB ØØ8B |  INTERUPT  AB FFEF |  IOBPDN    AB ØØ8A |
JOYRDY    AB CØ66 |  LAST      AB ØØ95 |  MAXTST    LB F425 |  MINTST    LB F41F |  MONTIMEH  AB ØØ9A |
MONTIMEL  AB ØØ99 |  MOTOF     LB FØ52 |  MOTOROFF  AB CØ88 |  MOTORON   AB CØ89 |  MSW1      LB F458 |
MSW2      LB F461 |  MSWAIT    LB F456 |  MYSEEK    LB F1Ø4 |  NBUF1     AB Ø2ØØ |  NBUF2     AB Ø3Ø2 |
NIBL      LB F355 |  NODRIVER  LB FØ64 |  NOWRITE   LB F2A3 |  NXOFF     LB F11A |  OFFTABLE  LB F47Ø |
OK        LB FØ48 |  ONEMEG    AB ØØ8Ø |  ONTABLE   LB F467 |  OUT       LB F41B |  PHASEOFF  AB CØ8Ø |
PHASEON   AB CØ81 |  PHASON    AB CØ81 |  PHSOFF    AB CØ8Ø |  PNIBL1    LB F314 |  PNIBL2    LB F323 |
POST1     LB F336 |  POST2     LB F338 |  POSTERR   LB F34C |  POSTNIB1  LB F3ØF |  PRENIB1   LB F2C8 |
PRENIB16  LB F2C4 |  PRENIB2   LB F2E5 |  PRENIB3   LB F2E2 |  PRENIB4   LB F2F6 |  PRIOR     AB ØØ9D |
Q6H       AB CØ8D |  Q6L       AB CØ8C |  Q7H       AB CØ8F |  Q7L       AB CØ8E |  QUIT      LB F49D |
RD1       LB F14D |  RD2       LB F157 |  RD3       LB F163 |  RD4       LB F16E |  RD5       LB F18Ø |
RD5A      LB F181 |  RD6       LB F195 |  RDA1      LB F1C4 |  RDA2      LB F1CE |  RDA3      LB F1D9 |
RDA4      LB F1E7 |  RDA5      LB F1EF |  RDA6      LB F2Ø1 |  RDA7      LB F2ØB |  RDADR16   LB F1B9 |
RDAFLD    LB F1E5 |  RDASN1    LB F1C9 |  RDASYN    LB F1BD |  RDCKSUM   LB F1AØ |  RDERR     LB F1B7 |
RDEXIT    LB F214 |  RDRIGHT   LB FØAC |  READ16    LB F148 |  REGRWTS   LB FØØØ |  RETRYCNT  AB ØØ93 |
RSYNC     LB F14A |  RSYNC1    LB F152 |  RTTRK     LB FØCØ |  SECT      AB ØØ98 |  SECTABL   LB F4AØ |
SEEK      LB F4ØØ |  SEEK1     LB F1Ø5 |  SEEK2     LB F4ØA |  SEEKCNT   AB ØØ94 |  SEEKEND   LB F444 |
SEEKRTS   LB F455 |  SERVICE   LB F2AA |  SET1MEG   LB F34C |  SETPHASE  LB F448 |  SETTRK    LB F125 |
SEV       LB F354 |  STEP      LB F429 |  STEP2     LB F42B |  TEMP      AB ØØ97 |  TIMER1H   AB FFD9 |
TIMER1L   AB FFD8 |  TIMLATCH  AB FFD9 |  TRACK     AB ØØ99 |  TRKCNT    AB ØØ95 |  TRKN      AB ØØ9E |
TRKN1     AB ØØ99 |  TRKSEC    LB F47E |  TRYADR    LB FØ83 |  TRYADR2   LB FØ8A |  TRYTRK    LB FØ69 |
TRYTRK2   LB FØ7F |  TWOMEG    AB ØØ7F |  VOLUME    AB ØØ9A |  VRYFRST   LB F253 |  WDATA2    LB F26E |
WDATA3    LB F27E |  WEXIT     LB F215 |  WINTRPT   LB F24B |  WMIDLE    LB F267 |  WNIBL     LB F2BD |
WNIBL7    LB F2BB |  WNIBL9    LB F2BA |  WNTRPT1   LB F264 |  WRBITSLM  LB F297 |  WRCKSUM   LB F292 |
WRIT      LB FØF9 |  WRIT1     LB F22Ø |  WRITE16   LB F216 |  WRTFRST   LB F255 |  WSYNC     LB F22D |

Assembly complete:      1Ø76 lines
Ø   Errors flagged on this Assembly

-----------------------------------------------------------------------------------------------------
65Ø2 OPCODE STATIC FREQUENCIES
-----------------------------------------------------------------------------------------------------

    ADC :     1  m
    AND :     8  |   *******
```

"APPLE_PAT_4_383_296_C_15" 262 KB 2000-02-28 dpi: 300h x 300v pix: 2324h x 3097v



10/31/89 9:56 HD:Apple ///:ROM - Disk I/O Page 14

```
    ASL :    3  |  **
    BCC :   1Ø  |  ********
    BCS :    7  |  ******
    BEQ :    8  |  *******
    BIT :    3  |  **
    BMI :   1Ø  |  ********
    BNE :   38  |  *********************************
    BPL :   28  |  ************************
    BVC :    1  m
    CLC :    9  |  *******
    CLI :    2  |  *
    CLV :    1  m
    CMP :   14  |  ************
    CPX :    1  m
    CPY :    4  |  ***
    DEC :    5  |  ****
    DEX :    2  |  *
    DEY :   13  |  ***********
    EOR :    8  |  *******
    INC :   1Ø  |  ********
    INX :    2  |  *
    INY :   12  |  **********
    JMP :    2  |  *
    JSR :   39  |  **********************************
    LDA :   86  M  **************************************************************************
    LDX :   12  |  **********
    LDY :   18  |  ***************
    LSR :    9  |  *******
    NOP :   13  |  ***********
    ORA :    9  |  *******
    PHA :   1Ø  |  ********
    PHP :    4  |  ***
    PLA :   11  |  *********
    PLP :    3  |  **
    ROL :    7  |  ******
    ROR :    6  |  *****
    RTS :   16  |  **************
    SBC :    2  |  *
    SEC :    9  |  *******
    SEI :    1  m
    STA :   42  |  ************************************
    STX :    1  m
    STY :    3  |  **
    TAX :    5  |  ****
    TAY :    3  |  **
    TXA :    1  m
    TYA :    4  |  ***

    Minimum frequency =    1
    Maximum frequency =   86

    Average frequency =   1Ø

    Unused opcodes:

    BRK  BVS  CLD  RTI  SED  TSX  TXS

    Program opcode usage:  87 %

---------------------------------------------------------------------------------------------------
(1.ØØ) That's all, Folks ...
---------------------------------------------------------------------------------------------------
```



Source Code Listing

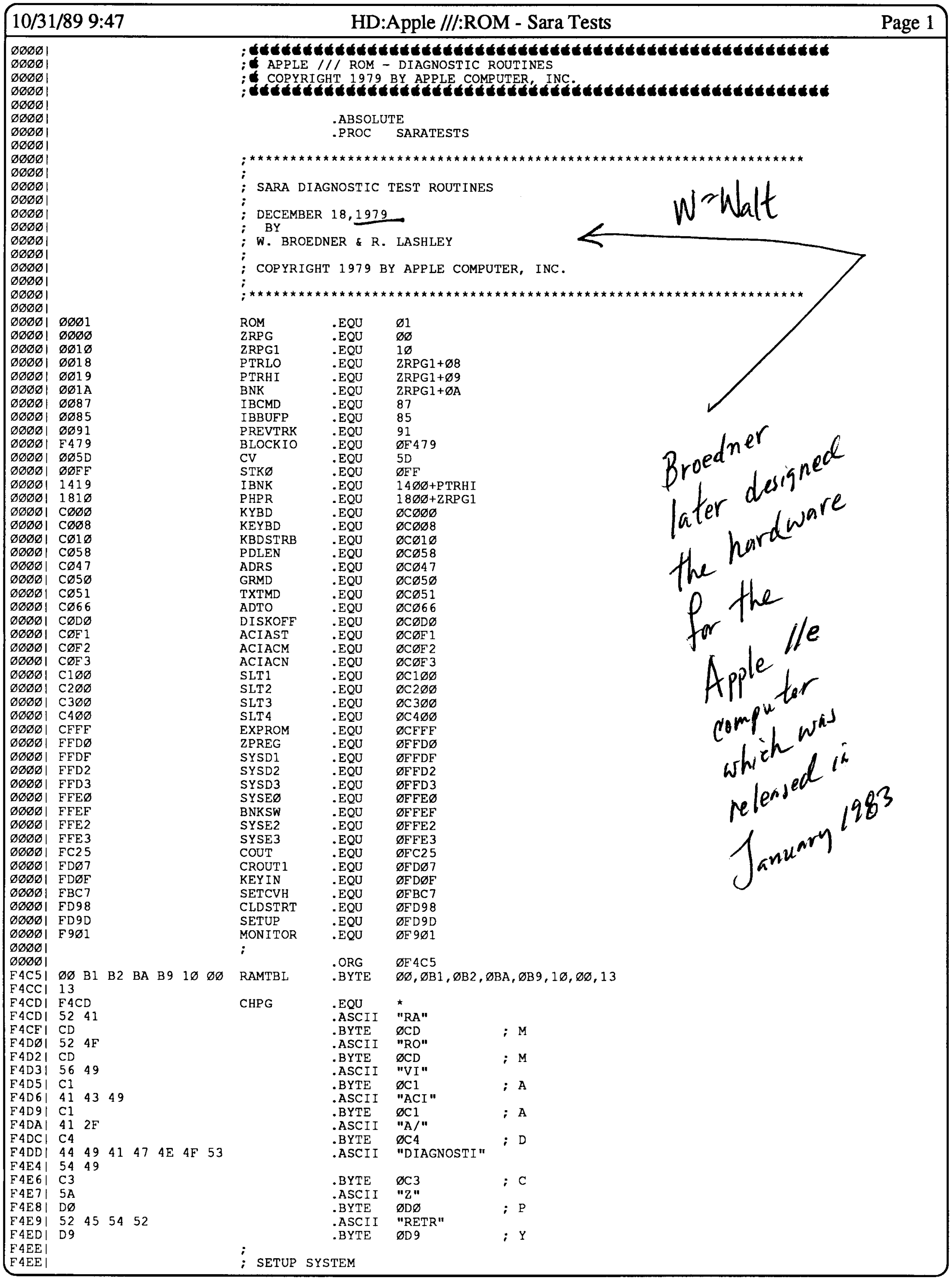
for

Apple ///

ROM Diagnostics

David T. Craig
736 Edgewater
Wichita, Kansas 67230

"APPLE_PAT_4_383_296_C_17" 165 KB 2000-02-28 dpi: 300h x 300v pix: 1928h x 2695v



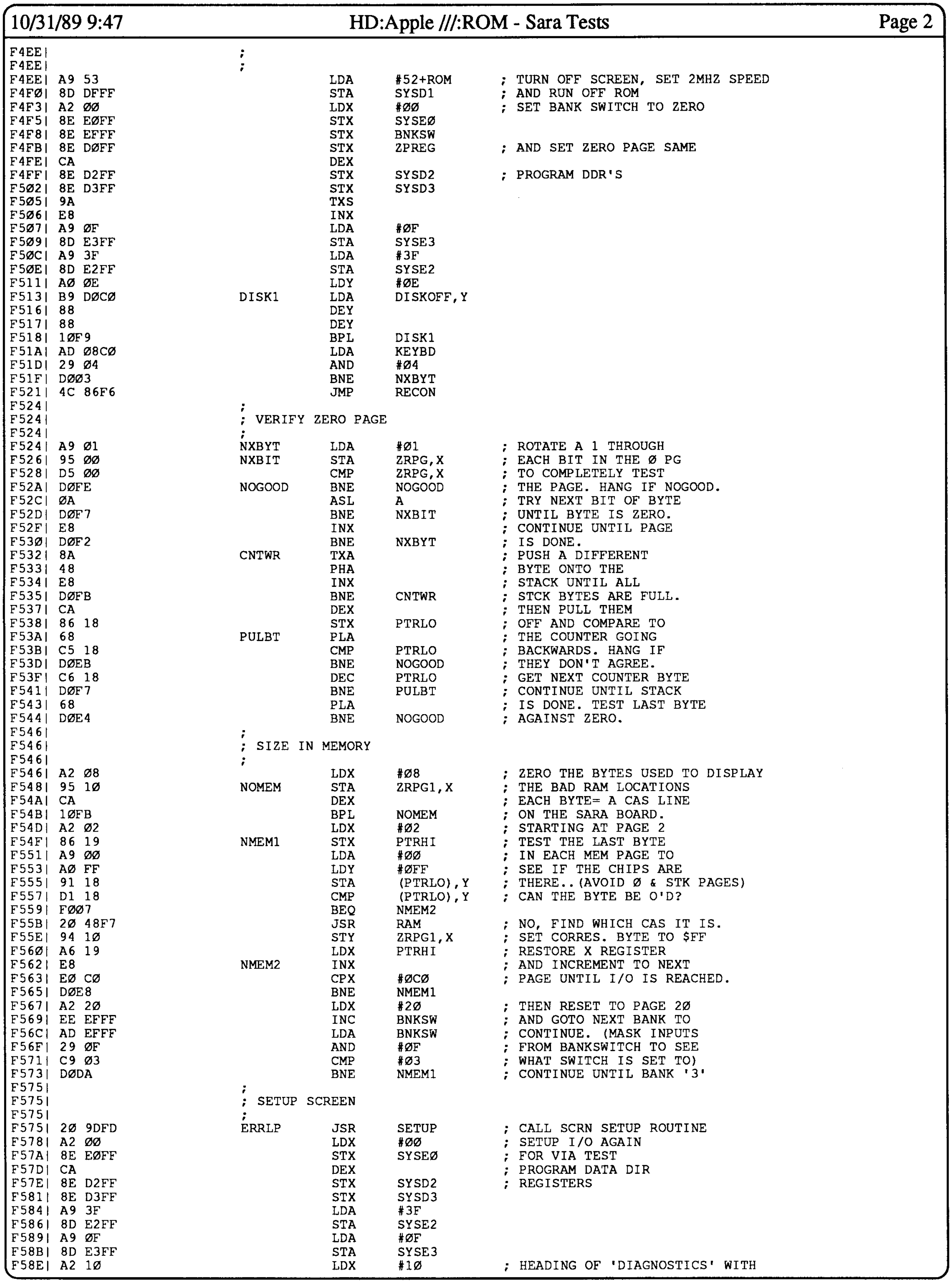
10/31/89 9:47 HD:Apple ///:ROM - Sara Tests Page 1

```
ØØØØ|                          ;
ØØØØ|                          ; APPLE /// ROM - DIAGNOSTIC ROUTINES
ØØØØ|                          ; COPYRIGHT 1979 BY APPLE COMPUTER, INC.
ØØØØ|                          ;
ØØØØ|
ØØØØ|                                    .ABSOLUTE
ØØØØ|                                    .PROC    SARATESTS
ØØØØ|
ØØØØ|                          ;********************************************************************
ØØØØ|                          ;
ØØØØ|                          ; SARA DIAGNOSTIC TEST ROUTINES
ØØØØ|                          ;
ØØØØ|                          ; DECEMBER 18,1979
ØØØØ|                          ;  BY
ØØØØ|                          ; W. BROEDNER & R. LASHLEY
ØØØØ|                          ;
ØØØØ|                          ; COPYRIGHT 1979 BY APPLE COMPUTER, INC.
ØØØØ|                          ;
ØØØØ|                          ;********************************************************************
ØØØØ|
ØØØØ| ØØØ1                     ROM        .EQU     Ø1
ØØØØ| ØØØØ                     ZRPG       .EQU     ØØ
ØØØØ| ØØ1Ø                     ZRPG1      .EQU     1Ø
ØØØØ| ØØ18                     PTRLO      .EQU     ZRPG1+Ø8
ØØØØ| ØØ19                     PTRHI      .EQU     ZRPG1+Ø9
ØØØØ| ØØ1A                     BNK        .EQU     ZRPG1+ØA
ØØØØ| ØØ87                     IBCMD      .EQU     87
ØØØØ| ØØ85                     IBBUFP     .EQU     85
ØØØØ| ØØ91                     PREVTRK    .EQU     91
ØØØØ| F479                     BLOCKIO    .EQU     ØF479
ØØØØ| ØØ5D                     CV         .EQU     5D
ØØØØ| ØØFF                     STKØ       .EQU     ØFF
ØØØØ| 1419                     IBNK       .EQU     14ØØ+PTRHI
ØØØØ| 181Ø                     PHPR       .EQU     18ØØ+ZRPG1
ØØØØ| CØØØ                     KYBD       .EQU     ØCØØØ
ØØØØ| CØØ8                     KEYBD      .EQU     ØCØØ8
ØØØØ| CØ1Ø                     KBDSTRB    .EQU     ØCØ1Ø
ØØØØ| CØ58                     PDLEN      .EQU     ØCØ58
ØØØØ| CØ47                     ADRS       .EQU     ØCØ47
ØØØØ| CØ5Ø                     GRMD       .EQU     ØCØ5Ø
ØØØØ| CØ51                     TXTMD      .EQU     ØCØ51
ØØØØ| CØ66                     ADTO       .EQU     ØCØ66
ØØØØ| CØDØ                     DISKOFF    .EQU     ØCØDØ
ØØØØ| CØF1                     ACIAST     .EQU     ØCØF1
ØØØØ| CØF2                     ACIACM     .EQU     ØCØF2
ØØØØ| CØF3                     ACIACN     .EQU     ØCØF3
ØØØØ| C1ØØ                     SLT1       .EQU     ØC1ØØ
ØØØØ| C2ØØ                     SLT2       .EQU     ØC2ØØ
ØØØØ| C3ØØ                     SLT3       .EQU     ØC3ØØ
ØØØØ| C4ØØ                     SLT4       .EQU     ØC4ØØ
ØØØØ| CFFF                     EXPROM     .EQU     ØCFFF
ØØØØ| FFDØ                     ZPREG      .EQU     ØFFDØ
ØØØØ| FFDF                     SYSD1      .EQU     ØFFDF
ØØØØ| FFD2                     SYSD2      .EQU     ØFFD2
ØØØØ| FFD3                     SYSD3      .EQU     ØFFD3
ØØØØ| FFEØ                     SYSEØ      .EQU     ØFFEØ
ØØØØ| FFEF                     BNKSW      .EQU     ØFFEF
ØØØØ| FFE2                     SYSE2      .EQU     ØFFE2
ØØØØ| FFE3                     SYSE3      .EQU     ØFFE3
ØØØØ| FC25                     COUT       .EQU     ØFC25
ØØØØ| FDØ7                     CROUT1     .EQU     ØFDØ7
ØØØØ| FDØF                     KEYIN      .EQU     ØFDØF
ØØØØ| FBC7                     SETCVH     .EQU     ØFBC7
ØØØØ| FD98                     CLDSTRT    .EQU     ØFD98
ØØØØ| FD9D                     SETUP      .EQU     ØFD9D
ØØØØ| F9Ø1                     MONITOR    .EQU     ØF9Ø1
ØØØØ|                          ;
ØØØØ|                                     .ORG     ØF4C5
F4C5| ØØ B1 B2 BA B9 1Ø ØØ     RAMTBL     .BYTE    ØØ,ØB1,ØB2,ØBA,ØB9,1Ø,ØØ,13
F4CC| 13
F4CD| F4CD                     CHPG       .EQU     *
F4CD| 52 41                               .ASCII   "RA"
F4CF| CD                                  .BYTE    ØCD          ; M
F4DØ| 52 4F                               .ASCII   "RO"
F4D2| CD                                  .BYTE    ØCD          ; M
F4D3| 56 49                               .ASCII   "VI"
F4D5| C1                                  .BYTE    ØC1          ; A
F4D6| 41 43 49                            .ASCII   "ACI"
F4D9| C1                                  .BYTE    ØC1          ; A
F4DA| 41 2F                               .ASCII   "A/"
F4DC| C4                                  .BYTE    ØC4          ; D
F4DD| 44 49 41 47 4E 4F 53                .ASCII   "DIAGNOSTI"
F4E4| 54 49
F4E6| C3                                  .BYTE    ØC3          ; C
F4E7| 5A                                  .ASCII   "Z"
F4E8| DØ                                  .BYTE    ØDØ          ; P
F4E9| 52 45 54 52                         .ASCII   "RETR"
F4ED| D9                                  .BYTE    ØD9          ; Y
F4EE|                          ;
F4EE|                          ; SETUP SYSTEM
```

W=Walt

Broedner later designed the hardware for the Apple //e computer which was released in January 1983

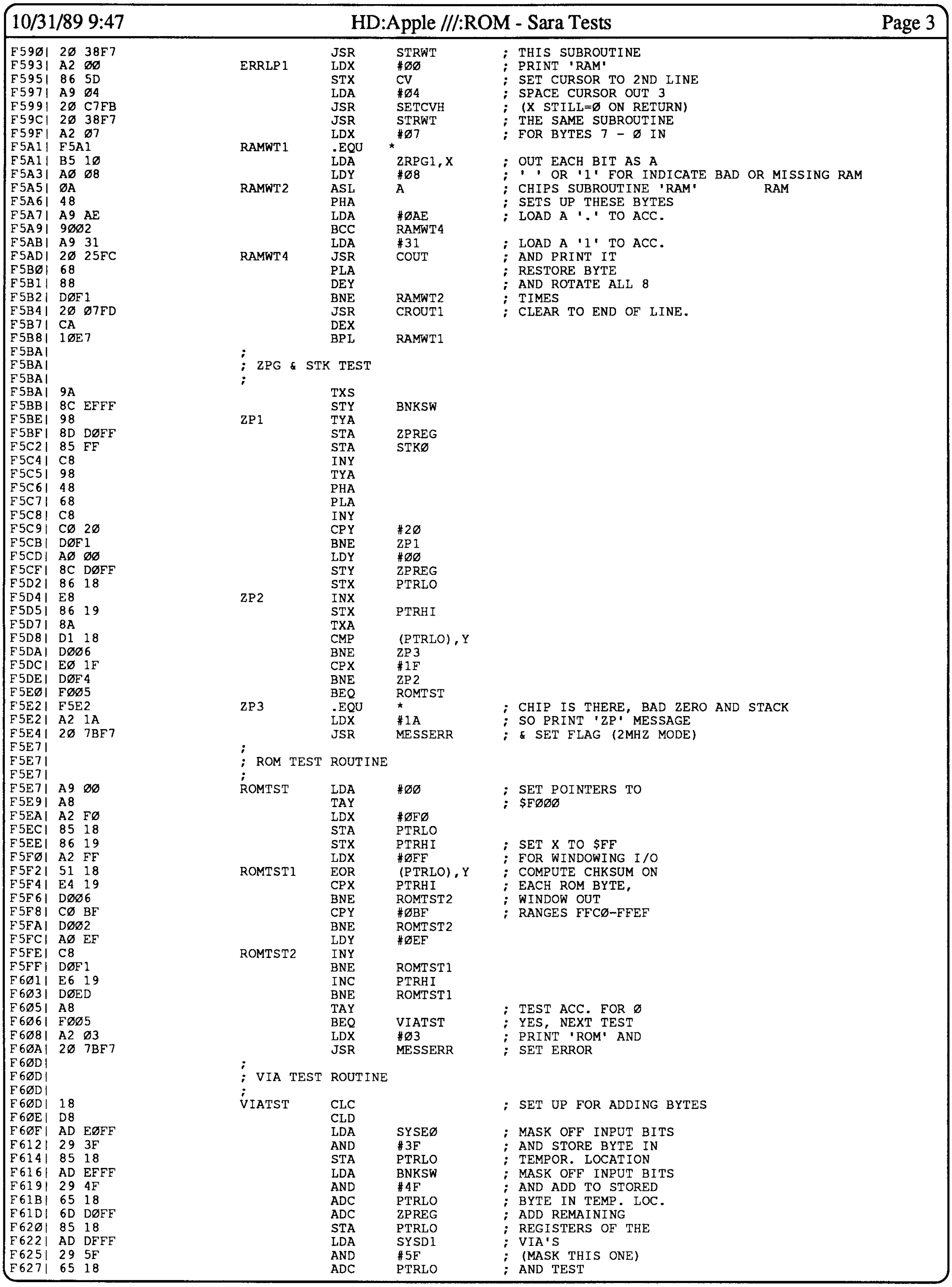
"APPLE_PAT_4_383_296_C_18" 192 KB 2000-02-28 dpi: 300h x 300v pix: 2324h x 3103v



10/31/89 9:47 HD:Apple ///:ROM - Sara Tests Page 2

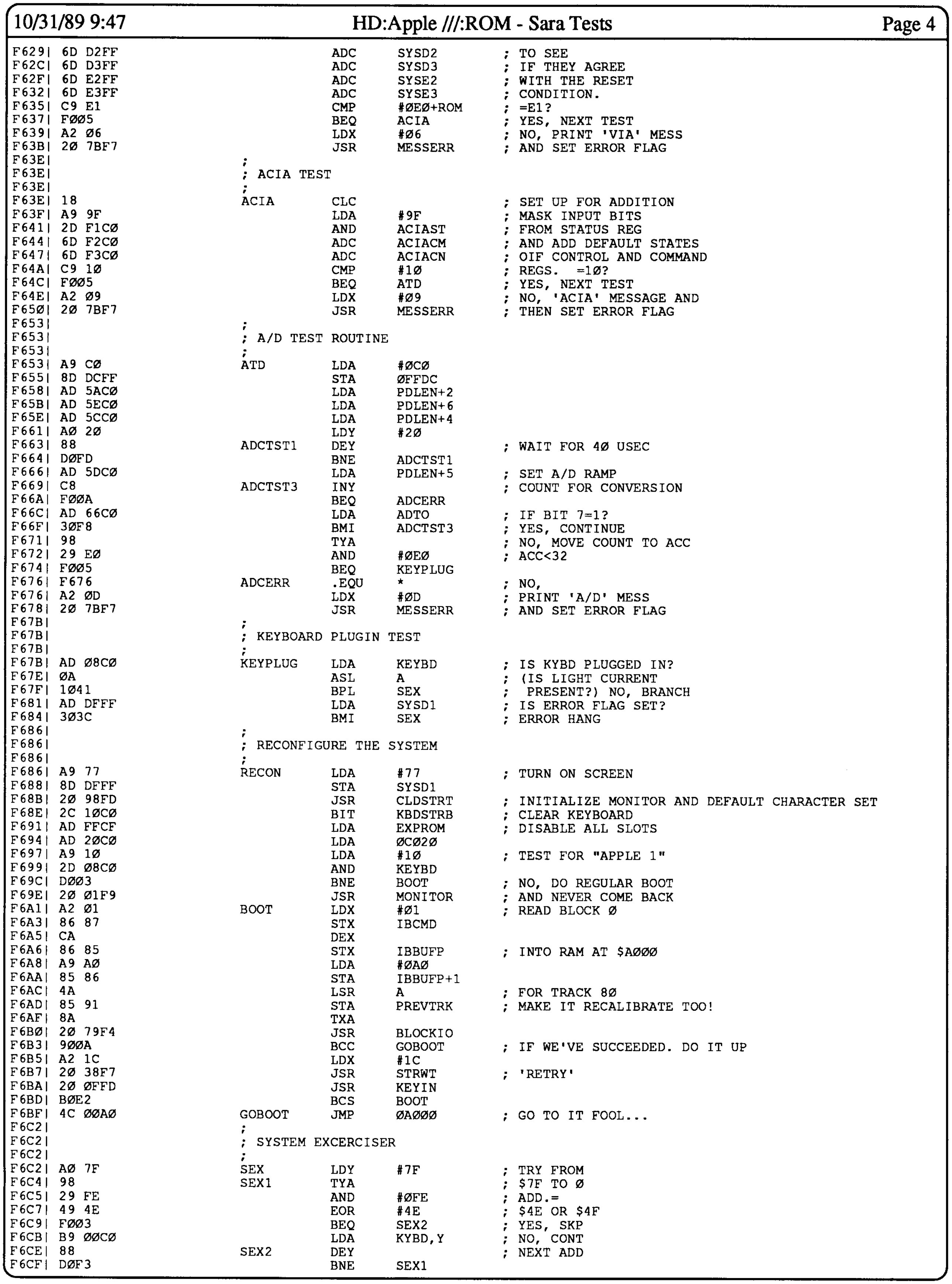
```
F4EE|                       ;
F4EE|                       ;
F4EE| A9 53                            LDA     #52+ROM      ; TURN OFF SCREEN, SET 2MHZ SPEED
F4FØ| 8D DFFF                          STA     SYSD1        ; AND RUN OFF ROM
F4F3| A2 ØØ                            LDX     #ØØ          ; SET BANK SWITCH TO ZERO
F4F5| 8E EØFF                          STX     SYSEØ
F4F8| 8E EFFF                          STX     BNKSW
F4FB| 8E DØFF                          STX     ZPREG        ; AND SET ZERO PAGE SAME
F4FE| CA                               DEX
F4FF| 8E D2FF                          STX     SYSD2        ; PROGRAM DDR'S
F5Ø2| 8E D3FF                          STX     SYSD3
F5Ø5| 9A                               TXS
F5Ø6| E8                               INX
F5Ø7| A9 ØF                            LDA     #ØF
F5Ø9| 8D E3FF                          STA     SYSE3
F5ØC| A9 3F                            LDA     #3F
F5ØE| 8D E2FF                          STA     SYSE2
F511| AØ ØE                            LDY     #ØE
F513| B9 DØCØ               DISK1      LDA     DISKOFF,Y
F516| 88                               DEY
F517| 88                               DEY
F518| 1ØF9                             BPL     DISK1
F51A| AD Ø8CØ                          LDA     KEYBD
F51D| 29 Ø4                            AND     #Ø4
F51F| DØØ3                             BNE     NXBYT
F521| 4C 86F6                          JMP     RECON
F524|                       ;
F524|                       ; VERIFY ZERO PAGE
F524|                       ;
F524| A9 Ø1                 NXBYT      LDA     #Ø1          ; ROTATE A 1 THROUGH
F526| 95 ØØ                 NXBIT      STA     ZRPG,X       ; EACH BIT IN THE Ø PG
F528| D5 ØØ                            CMP     ZRPG,X       ; TO COMPLETELY TEST
F52A| DØFE                  NOGOOD     BNE     NOGOOD       ; THE PAGE. HANG IF NOGOOD.
F52C| ØA                               ASL     A            ; TRY NEXT BIT OF BYTE
F52D| DØF7                             BNE     NXBIT        ; UNTIL BYTE IS ZERO.
F52F| E8                               INX                  ; CONTINUE UNTIL PAGE
F53Ø| DØF2                             BNE     NXBYT        ; IS DONE.
F532| 8A                    CNTWR      TXA                  ; PUSH A DIFFERENT
F533| 48                               PHA                  ; BYTE ONTO THE
F534| E8                               INX                  ; STACK UNTIL ALL
F535| DØFB                             BNE     CNTWR        ; STCK BYTES ARE FULL.
F537| CA                               DEX                  ; THEN PULL THEM
F538| 86 18                            STX     PTRLO        ; OFF AND COMPARE TO
F53A| 68                    PULBT      PLA                  ; THE COUNTER GOING
F53B| C5 18                            CMP     PTRLO        ; BACKWARDS. HANG IF
F53D| DØEB                             BNE     NOGOOD       ; THEY DON'T AGREE.
F53F| C6 18                            DEC     PTRLO        ; GET NEXT COUNTER BYTE
F541| DØF7                             BNE     PULBT        ; CONTINUE UNTIL STACK
F543| 68                               PLA                  ; IS DONE. TEST LAST BYTE
F544| DØE4                             BNE     NOGOOD       ; AGAINST ZERO.
F546|                       ;
F546|                       ; SIZE IN MEMORY
F546|                       ;
F546| A2 Ø8                            LDX     #Ø8          ; ZERO THE BYTES USED TO DISPLAY
F548| 95 1Ø                 NOMEM      STA     ZRPG1,X      ; THE BAD RAM LOCATIONS
F54A| CA                               DEX                  ; EACH BYTE= A CAS LINE
F54B| 1ØFB                             BPL     NOMEM        ; ON THE SARA BOARD.
F54D| A2 Ø2                            LDX     #Ø2          ; STARTING AT PAGE 2
F54F| 86 19                 NMEM1      STX     PTRHI        ; TEST THE LAST BYTE
F551| A9 ØØ                            LDA     #ØØ          ; IN EACH MEM PAGE TO
F553| AØ FF                            LDY     #ØFF         ; SEE IF THE CHIPS ARE
F555| 91 18                            STA     (PTRLO),Y    ; THERE..(AVOID Ø & STK PAGES)
F557| D1 18                            CMP     (PTRLO),Y    ; CAN THE BYTE BE O'D?
F559| FØØ7                             BEQ     NMEM2
F55B| 2Ø 48F7                          JSR     RAM          ; NO, FIND WHICH CAS IT IS.
F55E| 94 1Ø                            STY     ZRPG1,X      ; SET CORRES. BYTE TO $FF
F56Ø| A6 19                            LDX     PTRHI        ; RESTORE X REGISTER
F562| E8                    NMEM2      INX                  ; AND INCREMENT TO NEXT
F563| EØ CØ                            CPX     #ØCØ         ; PAGE UNTIL I/O IS REACHED.
F565| DØE8                             BNE     NMEM1
F567| A2 2Ø                            LDX     #2Ø          ; THEN RESET TO PAGE 2Ø
F569| EE EFFF                          INC     BNKSW        ; AND GOTO NEXT BANK TO
F56C| AD EFFF                          LDA     BNKSW        ; CONTINUE. (MASK INPUTS
F56F| 29 ØF                            AND     #ØF          ; FROM BANKSWITCH TO SEE
F571| C9 Ø3                            CMP     #Ø3          ; WHAT SWITCH IS SET TO)
F573| DØDA                             BNE     NMEM1        ; CONTINUE UNTIL BANK '3'
F575|                       ;
F575|                       ; SETUP SCREEN
F575|                       ;
F575| 2Ø 9DFD               ERRLP      JSR     SETUP        ; CALL SCRN SETUP ROUTINE
F578| A2 ØØ                            LDX     #ØØ          ; SETUP I/O AGAIN
F57A| 8E EØFF                          STX     SYSEØ        ; FOR VIA TEST
F57D| CA                               DEX                  ; PROGRAM DATA DIR
F57E| 8E D2FF                          STX     SYSD2        ; REGISTERS
F581| 8E D3FF                          STX     SYSD3
F584| A9 3F                            LDA     #3F
F586| 8D E2FF                          STA     SYSE2
F589| A9 ØF                            LDA     #ØF
F58B| 8D E3FF                          STA     SYSE3
F58E| A2 1Ø                            LDX     #1Ø          ; HEADING OF 'DIAGNOSTICS' WITH
```

"APPLE_PAT_4_383_296_C_19" 196 KB 2000-02-28 dpi: 300h x 300v pix: 2330h x 3103v



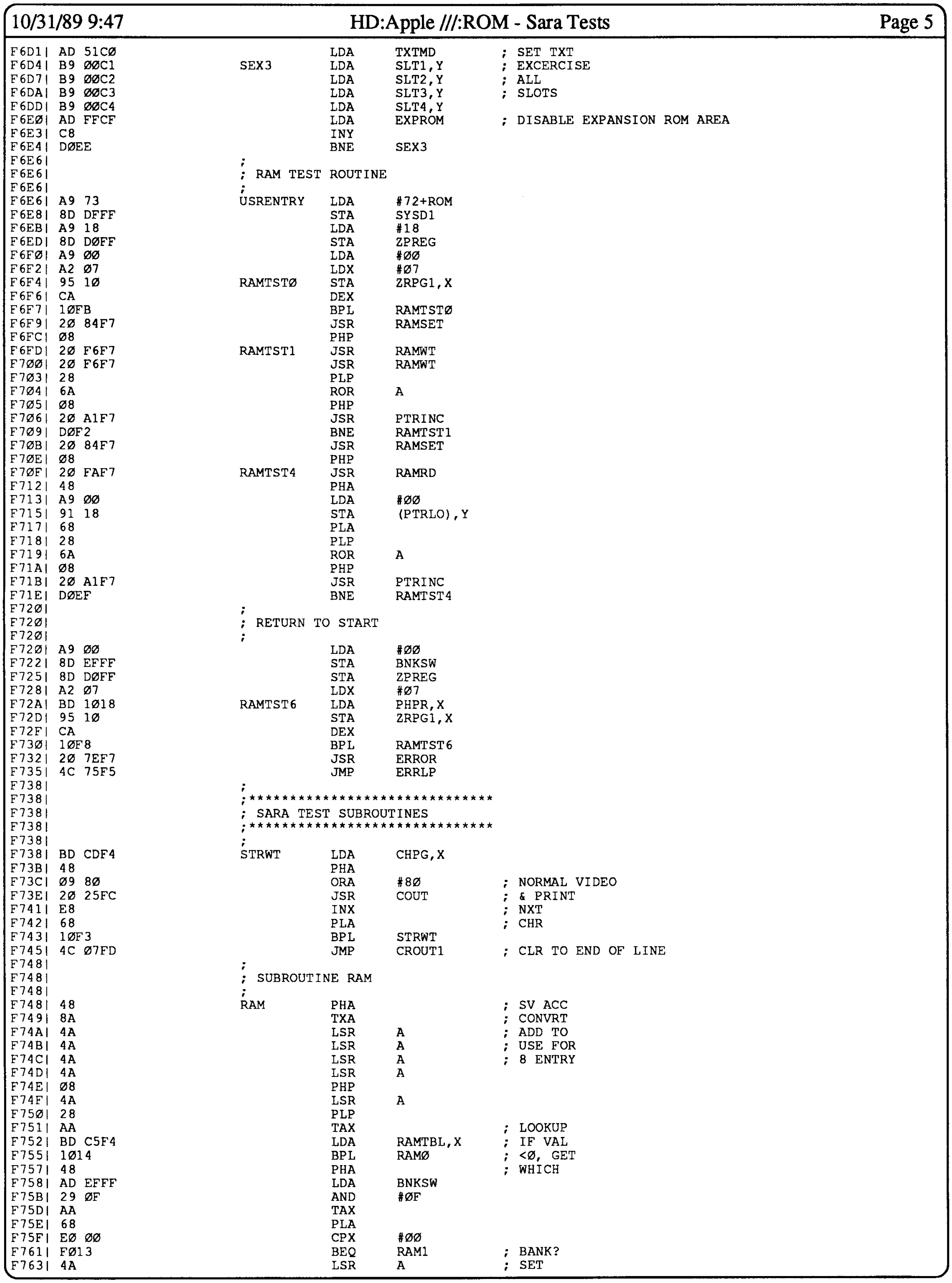
```
F59Ø| 2Ø 38F7                      JSR      STRWT         ; THIS SUBROUTINE
F593| A2 ØØ           ERRLP1       LDX      #ØØ           ; PRINT 'RAM'
F595| 86 5D                        STX      CV            ; SET CURSOR TO 2ND LINE
F597| A9 Ø4                        LDA      #Ø4           ; SPACE CURSOR OUT 3
F599| 2Ø C7FB                      JSR      SETCVH        ; (X STILL=Ø ON RETURN)
F59C| 2Ø 38F7                      JSR      STRWT         ; THE SAME SUBROUTINE
F59F| A2 Ø7                        LDX      #Ø7           ; FOR BYTES 7 - Ø IN
F5A1| F5A1            RAMWT1       .EQU   *
F5A1| B5 1Ø                        LDA      ZRPG1,X       ; OUT EACH BIT AS A
F5A3| AØ Ø8                        LDY      #Ø8           ; ' ' OR '1' FOR INDICATE BAD OR MISSING RAM
F5A5| ØA              RAMWT2       ASL      A             ; CHIPS SUBROUTINE 'RAM'        RAM
F5A6| 48                           PHA                    ; SETS UP THESE BYTES
F5A7| A9 AE                        LDA      #ØAE          ; LOAD A '.' TO ACC.
F5A9| 9ØØ2                         BCC      RAMWT4
F5AB| A9 31                        LDA      #31           ; LOAD A '1' TO ACC.
F5AD| 2Ø 25FC         RAMWT4       JSR      COUT          ; AND PRINT IT
F5BØ| 68                           PLA                    ; RESTORE BYTE
F5B1| 88                           DEY                    ; AND ROTATE ALL 8
F5B2| DØF1                         BNE      RAMWT2        ; TIMES
F5B4| 2Ø Ø7FD                      JSR      CROUT1        ; CLEAR TO END OF LINE.
F5B7| CA                           DEX
F5B8| 1ØE7                         BPL      RAMWT1
F5BA|                 ;
F5BA|                 ; ZPG & STK TEST
F5BA|                 ;
F5BA| 9A                           TXS
F5BB| 8C EFFF                      STY      BNKSW
F5BE| 98              ZP1          TYA
F5BF| 8D DØFF                      STA      ZPREG
F5C2| 85 FF                        STA      STKØ
F5C4| C8                           INY
F5C5| 98                           TYA
F5C6| 48                           PHA
F5C7| 68                           PLA
F5C8| C8                           INY
F5C9| CØ 2Ø                        CPY      #2Ø
F5CB| DØF1                         BNE      ZP1
F5CD| AØ ØØ                        LDY      #ØØ
F5CF| 8C DØFF                      STY      ZPREG
F5D2| 86 18                        STX      PTRLO
F5D4| E8              ZP2          INX
F5D5| 86 19                        STX      PTRHI
F5D7| 8A                           TXA
F5D8| D1 18                        CMP      (PTRLO),Y
F5DA| DØØ6                         BNE      ZP3
F5DC| EØ 1F                        CPX      #1F
F5DE| DØF4                         BNE      ZP2
F5EØ| FØØ5                         BEQ      ROMTST
F5E2| F5E2            ZP3          .EQU     *             ; CHIP IS THERE, BAD ZERO AND STACK
F5E2| A2 1A                        LDX      #1A           ; SO PRINT 'ZP' MESSAGE
F5E4| 2Ø 7BF7                      JSR      MESSERR       ; & SET FLAG (2MHZ MODE)
F5E7|                 ;
F5E7|                 ; ROM TEST ROUTINE
F5E7|                 ;
F5E7| A9 ØØ           ROMTST       LDA      #ØØ           ; SET POINTERS TO
F5E9| A8                           TAY                    ; $FØØØ
F5EA| A2 FØ                        LDX      #ØFØ
F5EC| 85 18                        STA      PTRLO
F5EE| 86 19                        STX      PTRHI         ; SET X TO $FF
F5FØ| A2 FF                        LDX      #ØFF          ; FOR WINDOWING I/O
F5F2| 51 18           ROMTST1      EOR      (PTRLO),Y     ; COMPUTE CHKSUM ON
F5F4| E4 19                        CPX      PTRHI         ; EACH ROM BYTE,
F5F6| DØØ6                         BNE      ROMTST2       ; WINDOW OUT
F5F8| CØ BF                        CPY      #ØBF          ; RANGES FFCØ-FFEF
F5FA| DØØ2                         BNE      ROMTST2
F5FC| AØ EF                        LDY      #ØEF
F5FE| C8              ROMTST2      INY
F5FF| DØF1                         BNE      ROMTST1
F6Ø1| E6 19                        INC      PTRHI
F6Ø3| DØED                         BNE      ROMTST1
F6Ø5| A8                           TAY                    ; TEST ACC. FOR Ø
F6Ø6| FØØ5                         BEQ      VIATST        ; YES, NEXT TEST
F6Ø8| A2 Ø3                        LDX      #Ø3           ; PRINT 'ROM' AND
F6ØA| 2Ø 7BF7                      JSR      MESSERR       ; SET ERROR
F6ØD|                 ;
F6ØD|                 ; VIA TEST ROUTINE
F6ØD|                 ;
F6ØD| 18              VIATST       CLC                    ; SET UP FOR ADDING BYTES
F6ØE| D8                           CLD
F6ØF| AD EØFF                      LDA      SYSEØ         ; MASK OFF INPUT BITS
F612| 29 3F                        AND      #3F           ; AND STORE BYTE IN
F614| 85 18                        STA      PTRLO         ; TEMPOR. LOCATION
F616| AD EFFF                      LDA      BNKSW         ; MASK OFF INPUT BITS
F619| 29 4F                        AND      #4F           ; AND ADD TO STORED
F61B| 65 18                        ADC      PTRLO         ; BYTE IN TEMP. LOC.
F61D| 6D DØFF                      ADC      ZPREG         ; ADD REMAINING
F62Ø| 85 18                        STA      PTRLO         ; REGISTERS OF THE
F622| AD DFFF                      LDA      SYSD1         ; VIA'S
F625| 29 5F                        AND      #5F           ; (MASK THIS ONE)
F627| 65 18                        ADC      PTRLO         ; AND TEST
```

“APPLE_PAT_4_383_296_C_20” 178 KB 2000-02-28 dpi: 300h x 300v pix: 2318h x 3097v



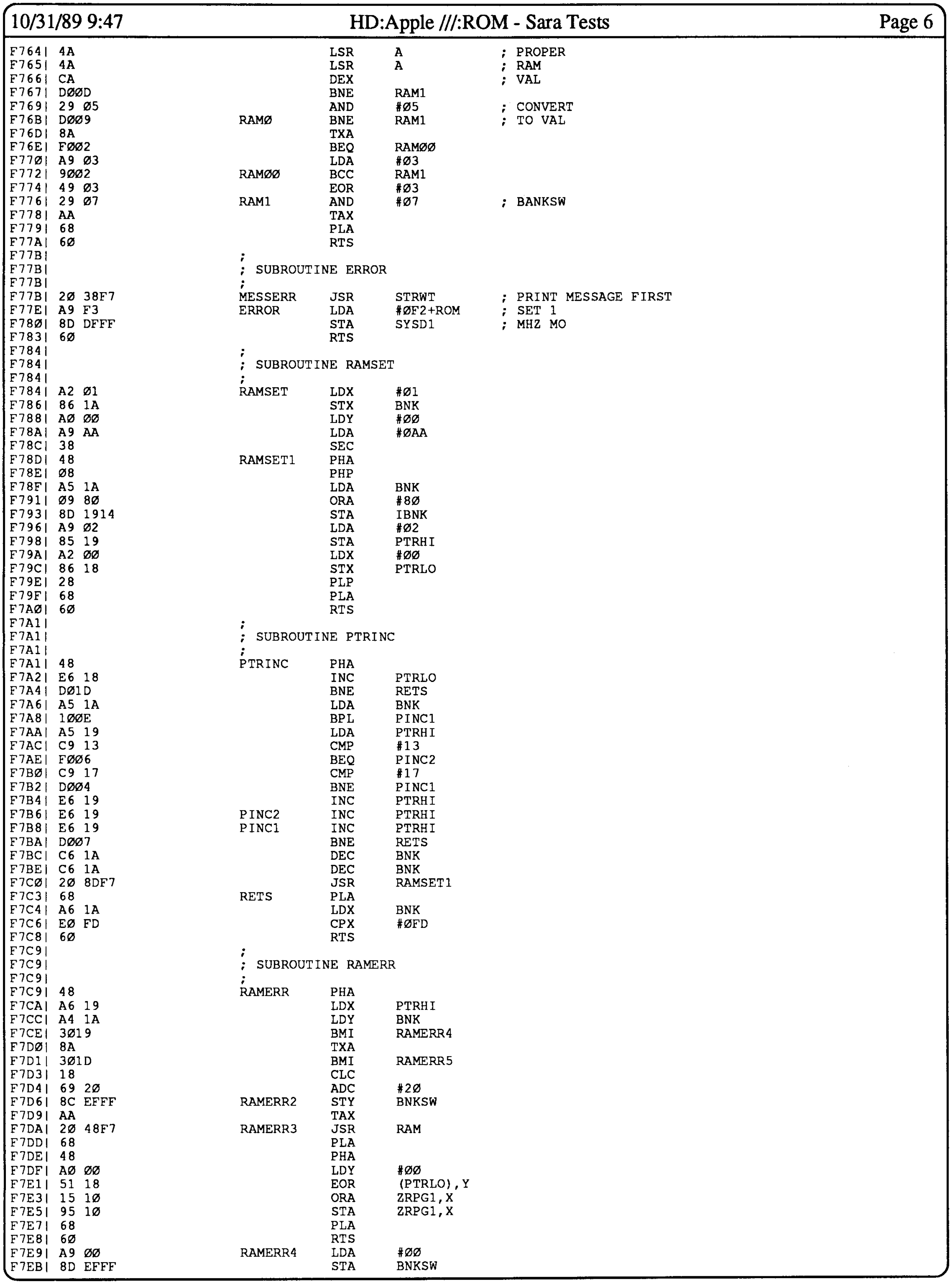
10/31/89 9:47 HD:Apple ///:ROM - Sara Tests Page 4

```
F629| 6D D2FF                    ADC     SYSD2          ; TO SEE
F62C| 6D D3FF                    ADC     SYSD3          ; IF THEY AGREE
F62F| 6D E2FF                    ADC     SYSE2          ; WITH THE RESET
F632| 6D E3FF                    ADC     SYSE3          ; CONDITION.
F635| C9 E1                      CMP     #ØEØ+ROM       ; =E1?
F637| FØØ5                       BEQ     ACIA           ; YES, NEXT TEST
F639| A2 Ø6                      LDX     #Ø6            ; NO, PRINT 'VIA' MESS
F63B| 2Ø 7BF7                    JSR     MESSERR        ; AND SET ERROR FLAG
F63E|                  ;
F63E|                  ; ACIA TEST
F63E|                  ;
F63E| 18               ACIA      CLC                    ; SET UP FOR ADDITION
F63F| A9 9F                      LDA     #9F            ; MASK INPUT BITS
F641| 2D F1CØ                    AND     ACIAST         ; FROM STATUS REG
F644| 6D F2CØ                    ADC     ACIACM         ; AND ADD DEFAULT STATES
F647| 6D F3CØ                    ADC     ACIACN         ; OIF CONTROL AND COMMAND
F64A| C9 1Ø                      CMP     #1Ø            ; REGS.  =1Ø?
F64C| FØØ5                       BEQ     ATD            ; YES, NEXT TEST
F64E| A2 Ø9                      LDX     #Ø9            ; NO, 'ACIA' MESSAGE AND
F65Ø| 2Ø 7BF7                    JSR     MESSERR        ; THEN SET ERROR FLAG
F653|                  ;
F653|                  ; A/D TEST ROUTINE
F653|                  ;
F653| A9 CØ            ATD       LDA     #ØCØ
F655| 8D DCFF                    STA     ØFFDC
F658| AD 5ACØ                    LDA     PDLEN+2
F65B| AD 5ECØ                    LDA     PDLEN+6
F65E| AD 5CCØ                    LDA     PDLEN+4
F661| AØ 2Ø                      LDY     #2Ø
F663| 88               ADCTST1   DEY                    ; WAIT FOR 4Ø USEC
F664| DØFD                       BNE     ADCTST1
F666| AD 5DCØ                    LDA     PDLEN+5        ; SET A/D RAMP
F669| C8               ADCTST3   INY                    ; COUNT FOR CONVERSION
F66A| FØØA                       BEQ     ADCERR
F66C| AD 66CØ                    LDA     ADTO           ; IF BIT 7=1?
F66F| 3ØF8                       BMI     ADCTST3        ; YES, CONTINUE
F671| 98                         TYA                    ; NO, MOVE COUNT TO ACC
F672| 29 EØ                      AND     #ØEØ           ; ACC<32
F674| FØØ5                       BEQ     KEYPLUG
F676| F676             ADCERR    .EQU    *              ; NO,
F676| A2 ØD                      LDX     #ØD            ; PRINT 'A/D' MESS
F678| 2Ø 7BF7                    JSR     MESSERR        ; AND SET ERROR FLAG
F67B|                  ;
F67B|                  ; KEYBOARD PLUGIN TEST
F67B|                  ;
F67B| AD Ø8CØ          KEYPLUG   LDA     KEYBD          ; IS KYBD PLUGGED IN?
F67E| ØA                         ASL     A              ; (IS LIGHT CURRENT
F67F| 1Ø41                       BPL     SEX            ;  PRESENT?) NO, BRANCH
F681| AD DFFF                    LDA     SYSD1          ; IS ERROR FLAG SET?
F684| 3Ø3C                       BMI     SEX            ; ERROR HANG
F686|                  ;
F686|                  ; RECONFIGURE THE SYSTEM
F686|                  ;
F686| A9 77            RECON     LDA     #77            ; TURN ON SCREEN
F688| 8D DFFF                    STA     SYSD1
F68B| 2Ø 98FD                    JSR     CLDSTRT        ; INITIALIZE MONITOR AND DEFAULT CHARACTER SET
F68E| 2C 1ØCØ                    BIT     KBDSTRB        ; CLEAR KEYBOARD
F691| AD FFCF                    LDA     EXPROM         ; DISABLE ALL SLOTS
F694| AD 2ØCØ                    LDA     ØCØ2Ø
F697| A9 1Ø                      LDA     #1Ø            ; TEST FOR "APPLE 1"
F699| 2D Ø8CØ                    AND     KEYBD
F69C| DØØ3                       BNE     BOOT           ; NO, DO REGULAR BOOT
F69E| 2Ø Ø1F9                    JSR     MONITOR        ; AND NEVER COME BACK
F6A1| A2 Ø1            BOOT      LDX     #Ø1            ; READ BLOCK Ø
F6A3| 86 87                      STX     IBCMD
F6A5| CA                         DEX
F6A6| 86 85                      STX     IBBUFP         ; INTO RAM AT $AØØØ
F6A8| A9 AØ                      LDA     #ØAØ
F6AA| 85 86                      STA     IBBUFP+1
F6AC| 4A                         LSR     A              ; FOR TRACK 8Ø
F6AD| 85 91                      STA     PREVTRK        ; MAKE IT RECALIBRATE TOO!
F6AF| 8A                         TXA
F6BØ| 2Ø 79F4                    JSR     BLOCKIO
F6B3| 9ØØA                       BCC     GOBOOT         ; IF WE'VE SUCCEEDED. DO IT UP
F6B5| A2 1C                      LDX     #1C
F6B7| 2Ø 38F7                    JSR     STRWT          ; 'RETRY'
F6BA| 2Ø ØFFD                    JSR     KEYIN
F6BD| BØE2                       BCS     BOOT
F6BF| 4C ØØAØ          GOBOOT    JMP     ØAØØØ          ; GO TO IT FOOL...
F6C2|                  ;
F6C2|                  ; SYSTEM EXCERCISER
F6C2|                  ;
F6C2| AØ 7F            SEX       LDY     #7F            ; TRY FROM
F6C4| 98               SEX1      TYA                    ; $7F TO Ø
F6C5| 29 FE                      AND     #ØFE           ; ADD.=
F6C7| 49 4E                      EOR     #4E            ; $4E OR $4F
F6C9| FØØ3                       BEQ     SEX2           ; YES, SKP
F6CB| B9 ØØCØ                    LDA     KYBD,Y         ; NO, CONT
F6CE| 88               SEX2      DEY                    ; NEXT ADD
F6CF| DØF3                       BNE     SEX1
```

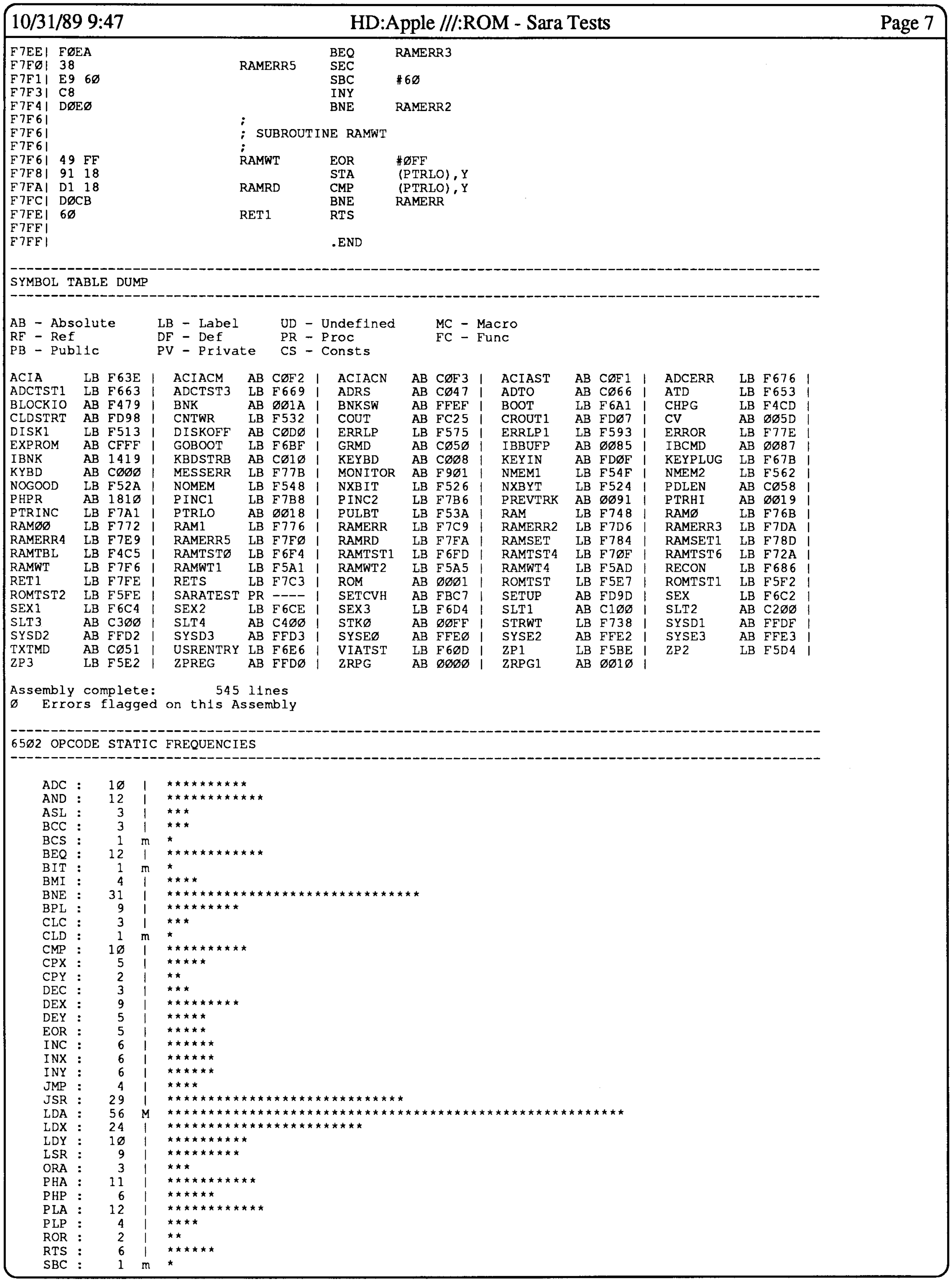


10/31/89 9:47   HD:Apple ///:ROM - Sara Tests   Page 5

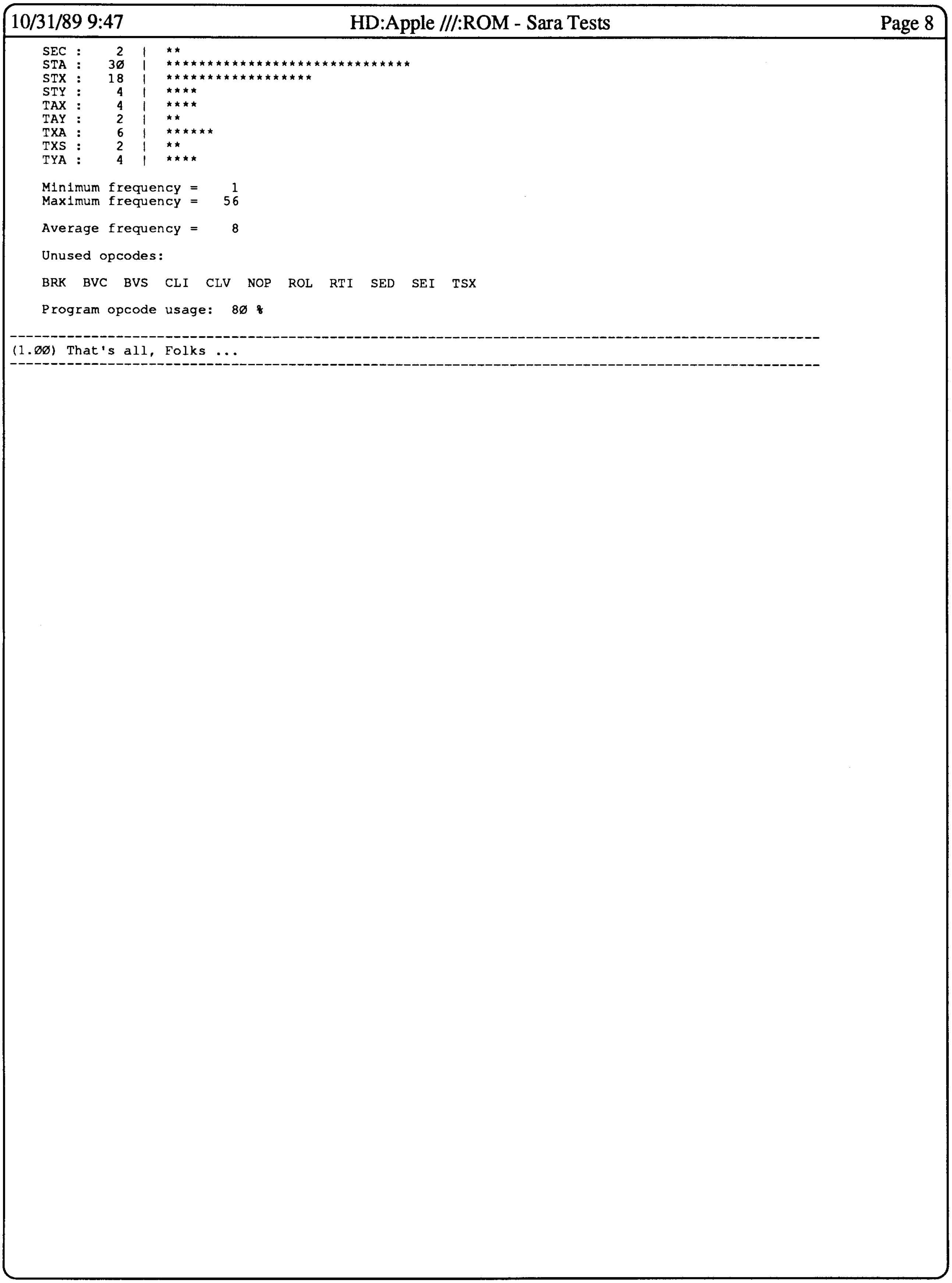
```
F6D1| AD 51CØ                          LDA     TXTMD        ; SET TXT
F6D4| B9 ØØC1               SEX3       LDA     SLT1,Y       ; EXCERCISE
F6D7| B9 ØØC2                          LDA     SLT2,Y       ; ALL
F6DA| B9 ØØC3                          LDA     SLT3,Y       ; SLOTS
F6DD| B9 ØØC4                          LDA     SLT4,Y
F6EØ| AD FFCF                          LDA     EXPROM       ; DISABLE EXPANSION ROM AREA
F6E3| C8                               INY
F6E4| DØEE                             BNE     SEX3
F6E6|                       ;
F6E6|                       ; RAM TEST ROUTINE
F6E6|                       ;
F6E6| A9 73                 USRENTRY   LDA     #72+ROM
F6E8| 8D DFFF                          STA     SYSD1
F6EB| A9 18                            LDA     #18
F6ED| 8D DØFF                          STA     ZPREG
F6FØ| A9 ØØ                            LDA     #ØØ
F6F2| A2 Ø7                            LDX     #Ø7
F6F4| 95 1Ø                 RAMTSTØ    STA     ZRPG1,X
F6F6| CA                               DEX
F6F7| 1ØFB                             BPL     RAMTSTØ
F6F9| 2Ø 84F7                          JSR     RAMSET
F6FC| Ø8                               PHP
F6FD| 2Ø F6F7               RAMTST1    JSR     RAMWT
F7ØØ| 2Ø F6F7                          JSR     RAMWT
F7Ø3| 28                               PLP
F7Ø4| 6A                               ROR     A
F7Ø5| Ø8                               PHP
F7Ø6| 2Ø A1F7                          JSR     PTRINC
F7Ø9| DØF2                             BNE     RAMTST1
F7ØB| 2Ø 84F7                          JSR     RAMSET
F7ØE| Ø8                               PHP
F7ØF| 2Ø FAF7               RAMTST4    JSR     RAMRD
F712| 48                               PHA
F713| A9 ØØ                            LDA     #ØØ
F715| 91 18                            STA     (PTRLO),Y
F717| 68                               PLA
F718| 28                               PLP
F719| 6A                               ROR     A
F71A| Ø8                               PHP
F71B| 2Ø A1F7                          JSR     PTRINC
F71E| DØEF                             BNE     RAMTST4
F72Ø|                       ;
F72Ø|                       ; RETURN TO START
F72Ø|                       ;
F72Ø| A9 ØØ                            LDA     #ØØ
F722| 8D EFFF                          STA     BNKSW
F725| 8D DØFF                          STA     ZPREG
F728| A2 Ø7                            LDX     #Ø7
F72A| BD 1Ø18               RAMTST6    LDA     PHPR,X
F72D| 95 1Ø                            STA     ZRPG1,X
F72F| CA                               DEX
F73Ø| 1ØF8                             BPL     RAMTST6
F732| 2Ø 7EF7                          JSR     ERROR
F735| 4C 75F5                          JMP     ERRLP
F738|                       ;
F738|                       ;******************************
F738|                       ; SARA TEST SUBROUTINES
F738|                       ;******************************
F738|                       ;
F738| BD CDF4               STRWT      LDA     CHPG,X
F73B| 48                               PHA
F73C| Ø9 8Ø                            ORA     #8Ø          ; NORMAL VIDEO
F73E| 2Ø 25FC                          JSR     COUT         ; & PRINT
F741| E8                               INX                  ; NXT
F742| 68                               PLA                  ; CHR
F743| 1ØF3                             BPL     STRWT
F745| 4C Ø7FD                          JMP     CROUT1       ; CLR TO END OF LINE
F748|                       ;
F748|                       ; SUBROUTINE RAM
F748|                       ;
F748| 48                    RAM        PHA                  ; SV ACC
F749| 8A                               TXA                  ; CONVRT
F74A| 4A                               LSR     A            ; ADD TO
F74B| 4A                               LSR     A            ; USE FOR
F74C| 4A                               LSR     A            ; 8 ENTRY
F74D| 4A                               LSR     A
F74E| Ø8                               PHP
F74F| 4A                               LSR     A
F75Ø| 28                               PLP
F751| AA                               TAX                  ; LOOKUP
F752| BD C5F4                          LDA     RAMTBL,X     ; IF VAL
F755| 1Ø14                             BPL     RAMØ         ; <Ø, GET
F757| 48                               PHA                  ; WHICH
F758| AD EFFF                          LDA     BNKSW
F75B| 29 ØF                            AND     #ØF
F75D| AA                               TAX
F75E| 68                               PLA
F75F| EØ ØØ                            CPX     #ØØ
F761| FØ13                             BEQ     RAM1         ; BANK?
F763| 4A                               LSR     A            ; SET
```



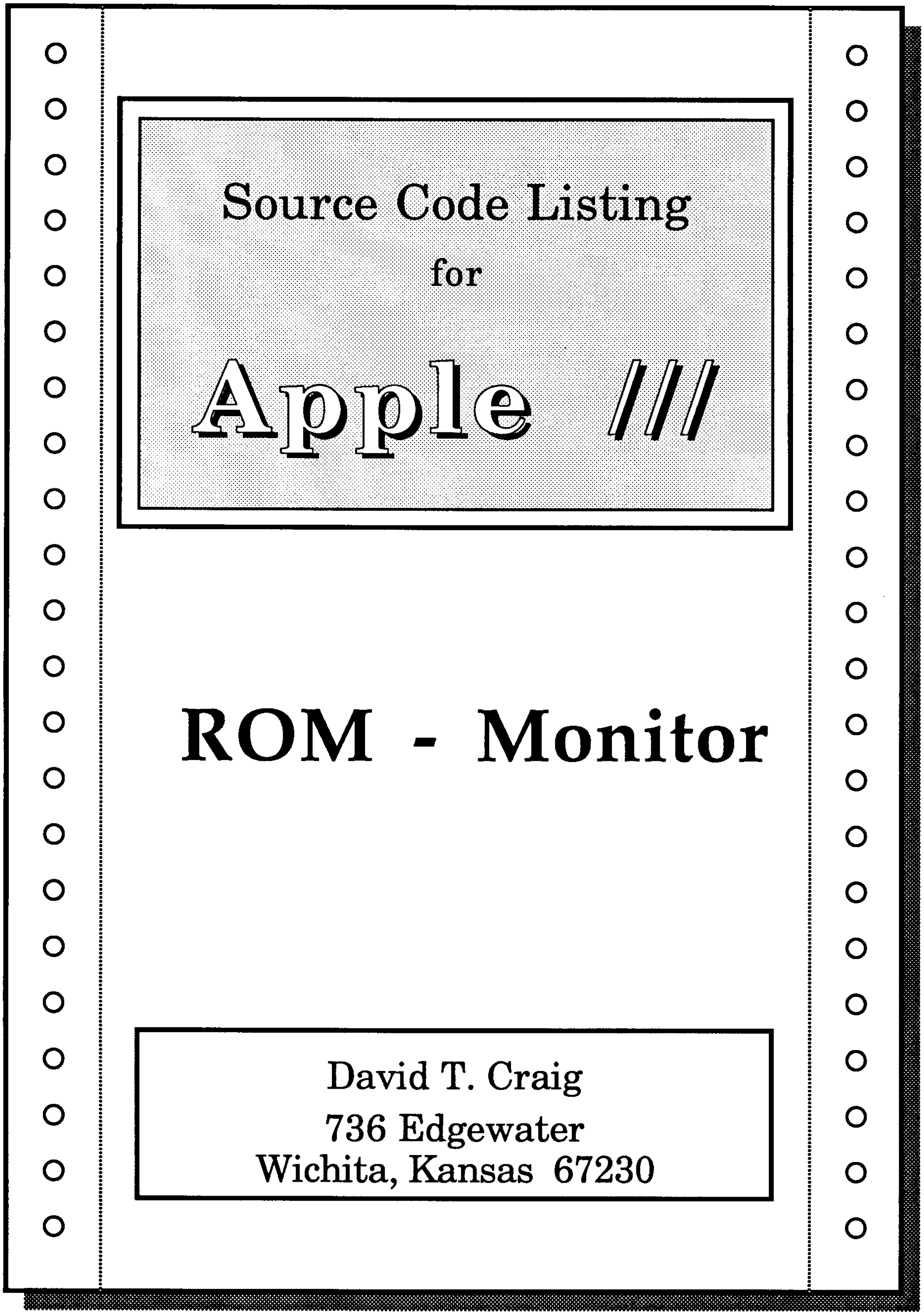
```
F764| 4A                               LSR     A            ; PROPER
F765| 4A                               LSR     A            ; RAM
F766| CA                               DEX                  ; VAL
F767| DØØD                             BNE     RAM1
F769| 29 Ø5                            AND     #Ø5          ; CONVERT
F76B| DØØ9                  RAMØ       BNE     RAM1         ; TO VAL
F76D| 8A                               TXA
F76E| FØØ2                             BEQ     RAMØØ
F77Ø| A9 Ø3                            LDA     #Ø3
F772| 9ØØ2                  RAMØØ      BCC     RAM1
F774| 49 Ø3                            EOR     #Ø3
F776| 29 Ø7                 RAM1       AND     #Ø7          ; BANKSW
F778| AA                               TAX
F779| 68                               PLA
F77A| 6Ø                               RTS
F77B|                       ;
F77B|                       ; SUBROUTINE ERROR
F77B|                       ;
F77B| 2Ø 38F7               MESSERR    JSR     STRWT        ; PRINT MESSAGE FIRST
F77E| A9 F3                 ERROR      LDA     #ØF2+ROM     ; SET 1
F78Ø| 8D DFFF                          STA     SYSD1        ; MHZ MO
F783| 6Ø                               RTS
F784|                       ;
F784|                       ; SUBROUTINE RAMSET
F784|                       ;
F784| A2 Ø1                 RAMSET     LDX     #Ø1
F786| 86 1A                            STX     BNK
F788| AØ ØØ                            LDY     #ØØ
F78A| A9 AA                            LDA     #ØAA
F78C| 38                               SEC
F78D| 48                    RAMSET1    PHA
F78E| Ø8                               PHP
F78F| A5 1A                            LDA     BNK
F791| Ø9 8Ø                            ORA     #8Ø
F793| 8D 1914                          STA     IBNK
F796| A9 Ø2                            LDA     #Ø2
F798| 85 19                            STA     PTRHI
F79A| A2 ØØ                            LDX     #ØØ
F79C| 86 18                            STX     PTRLO
F79E| 28                               PLP
F79F| 68                               PLA
F7AØ| 6Ø                               RTS
F7A1|                       ;
F7A1|                       ; SUBROUTINE PTRINC
F7A1|                       ;
F7A1| 48                    PTRINC     PHA
F7A2| E6 18                            INC     PTRLO
F7A4| DØ1D                             BNE     RETS
F7A6| A5 1A                            LDA     BNK
F7A8| 1ØØE                             BPL     PINC1
F7AA| A5 19                            LDA     PTRHI
F7AC| C9 13                            CMP     #13
F7AE| FØØ6                             BEQ     PINC2
F7BØ| C9 17                            CMP     #17
F7B2| DØØ4                             BNE     PINC1
F7B4| E6 19                            INC     PTRHI
F7B6| E6 19                 PINC2      INC     PTRHI
F7B8| E6 19                 PINC1      INC     PTRHI
F7BA| DØØ7                             BNE     RETS
F7BC| C6 1A                            DEC     BNK
F7BE| C6 1A                            DEC     BNK
F7CØ| 2Ø 8DF7                          JSR     RAMSET1
F7C3| 68                    RETS       PLA
F7C4| A6 1A                            LDX     BNK
F7C6| EØ FD                            CPX     #ØFD
F7C8| 6Ø                               RTS
F7C9|                       ;
F7C9|                       ; SUBROUTINE RAMERR
F7C9|                       ;
F7C9| 48                    RAMERR     PHA
F7CA| A6 19                            LDX     PTRHI
F7CC| A4 1A                            LDY     BNK
F7CE| 3Ø19                             BMI     RAMERR4
F7DØ| 8A                               TXA
F7D1| 3Ø1D                             BMI     RAMERR5
F7D3| 18                               CLC
F7D4| 69 2Ø                            ADC     #2Ø
F7D6| 8C EFFF               RAMERR2    STY     BNKSW
F7D9| AA                               TAX
F7DA| 2Ø 48F7               RAMERR3    JSR     RAM
F7DD| 68                               PLA
F7DE| 48                               PHA
F7DF| AØ ØØ                            LDY     #ØØ
F7E1| 51 18                            EOR     (PTRLO),Y
F7E3| 15 1Ø                            ORA     ZRPG1,X
F7E5| 95 1Ø                            STA     ZRPG1,X
F7E7| 68                               PLA
F7E8| 6Ø                               RTS
F7E9| A9 ØØ                 RAMERR4    LDA     #ØØ
F7EB| 8D EFFF                          STA     BNKSW
```



```
F7EE| FØEA                             BEQ        RAMERR3
F7FØ| 38                   RAMERR5     SEC
F7F1| E9 6Ø                            SBC        #6Ø
F7F3| C8                               INY
F7F4| DØEØ                             BNE        RAMERR2
F7F6|                      ;
F7F6|                      ; SUBROUTINE RAMWT
F7F6|                      ;
F7F6| 49 FF                RAMWT       EOR        #ØFF
F7F8| 91 18                            STA        (PTRLO),Y
F7FA| D1 18                RAMRD       CMP        (PTRLO),Y
F7FC| DØCB                             BNE        RAMERR
F7FE| 6Ø                   RET1        RTS
F7FF|
F7FF|                                  .END

-------------------------------------------------------------------------------------------------
SYMBOL TABLE DUMP
-------------------------------------------------------------------------------------------------

AB - Absolute       LB - Label      UD - Undefined      MC - Macro
RF - Ref            DF - Def        PR - Proc           FC - Func
PB - Public         PV - Private    CS - Consts

ACIA     LB F63E | ACIACM   AB CØF2 | ACIACN   AB CØF3 | ACIAST   AB CØF1 | ADCERR   LB F676 |
ADCTST1  LB F663 | ADCTST3  LB F669 | ADRS     AB CØ47 | ADTO     AB CØ66 | ATD      LB F653 |
BLOCKIO  AB F479 | BNK      AB ØØ1A | BNKSW    AB FFEF | BOOT     LB F6A1 | CHPG     LB F4CD |
CLDSTRT  AB FD98 | CNTWR    LB F532 | COUT     AB FC25 | CROUT1   AB FDØ7 | CV       AB ØØ5D |
DISK1    LB F513 | DISKOFF  AB CØDØ | ERRLP    LB F575 | ERRLP1   LB F593 | ERROR    LB F77E |
EXPROM   AB CFFF | GOBOOT   LB F6BF | GRMD     AB CØ5Ø | IBBUFP   AB ØØ85 | IBCMD    AB ØØ87 |
IBNK     AB 1419 | KBDSTRB  AB CØ1Ø | KEYBD    AB CØØ8 | KEYIN    AB FDØF | KEYPLUG  LB F67B |
KYBD     AB CØØØ | MESSERR  LB F77B | MONITOR  AB F9Ø1 | NMEM1    LB F54F | NMEM2    LB F562 |
NOGOOD   LB F52A | NOMEM    LB F548 | NXBIT    LB F526 | NXBYT    LB F524 | PDLEN    AB CØ58 |
PHPR     AB 181Ø | PINC1    LB F7B8 | PINC2    LB F7B6 | PREVTRK  AB ØØ91 | PTRHI    AB ØØ19 |
PTRINC   LB F7A1 | PTRLO    AB ØØ18 | PULBT    LB F53A | RAM      LB F748 | RAMØ     LB F76B |
RAMØØ    LB F772 | RAM1     LB F776 | RAMERR   LB F7C9 | RAMERR2  LB F7D6 | RAMERR3  LB F7DA |
RAMERR4  LB F7E9 | RAMERR5  LB F7FØ | RAMRD    LB F7FA | RAMSET   LB F784 | RAMSET1  LB F78D |
RAMTBL   LB F4C5 | RAMTSTØ  LB F6F4 | RAMTST1  LB F6FD | RAMTST4  LB F7ØF | RAMTST6  LB F72A |
RAMWT    LB F7F6 | RAMWT1   LB F5A1 | RAMWT2   LB F5A5 | RAMWT4   LB F5AD | RECON    LB F686 |
RET1     LB F7FE | RETS     LB F7C3 | ROM      AB ØØØ1 | ROMTST   LB F5E7 | ROMTST1  LB F5F2 |
ROMTST2  LB F5FE | SARATEST PR ---- | SETCVH   AB FBC7 | SETUP    AB FD9D | SEX      LB F6C2 |
SEX1     LB F6C4 | SEX2     LB F6CE | SEX3     LB F6D4 | SLT1     AB C1ØØ | SLT2     AB C2ØØ |
SLT3     AB C3ØØ | SLT4     AB C4ØØ | STKØ     AB ØØFF | STRWT    LB F738 | SYSD1    AB FFDF |
SYSD2    AB FFD2 | SYSD3    AB FFD3 | SYSEØ    AB FFEØ | SYSE2    AB FFE2 | SYSE3    AB FFE3 |
TXTMD    AB CØ51 | USRENTRY LB F6E6 | VIATST   LB F6ØD | ZP1      LB F5BE | ZP2      LB F5D4 |
ZP3      LB F5E2 | ZPREG    AB FFDØ | ZRPG     AB ØØØØ | ZRPG1    AB ØØ1Ø |

Assembly complete:       545 lines
Ø   Errors flagged on this Assembly

-------------------------------------------------------------------------------------------------
65Ø2 OPCODE STATIC FREQUENCIES
-------------------------------------------------------------------------------------------------

    ADC :   1Ø  |  **********
    AND :   12  |  ************
    ASL :    3  |  ***
    BCC :    3  |  ***
    BCS :    1  m  *
    BEQ :   12  |  ************
    BIT :    1  m  *
    BMI :    4  |  ****
    BNE :   31  |  *******************************
    BPL :    9  |  *********
    CLC :    3  |  ***
    CLD :    1  m  *
    CMP :   1Ø  |  **********
    CPX :    5  |  *****
    CPY :    2  |  **
    DEC :    3  |  ***
    DEX :    9  |  *********
    DEY :    5  |  *****
    EOR :    5  |  *****
    INC :    6  |  ******
    INX :    6  |  ******
    INY :    6  |  ******
    JMP :    4  |  ****
    JSR :   29  |  *****************************
    LDA :   56  M  ********************************************************
    LDX :   24  |  ************************
    LDY :   1Ø  |  **********
    LSR :    9  |  *********
    ORA :    3  |  ***
    PHA :   11  |  ***********
    PHP :    6  |  ******
    PLA :   12  |  ************
    PLP :    4  |  ****
    ROR :    2  |  **
    RTS :    6  |  ******
    SBC :    1  m  *
```



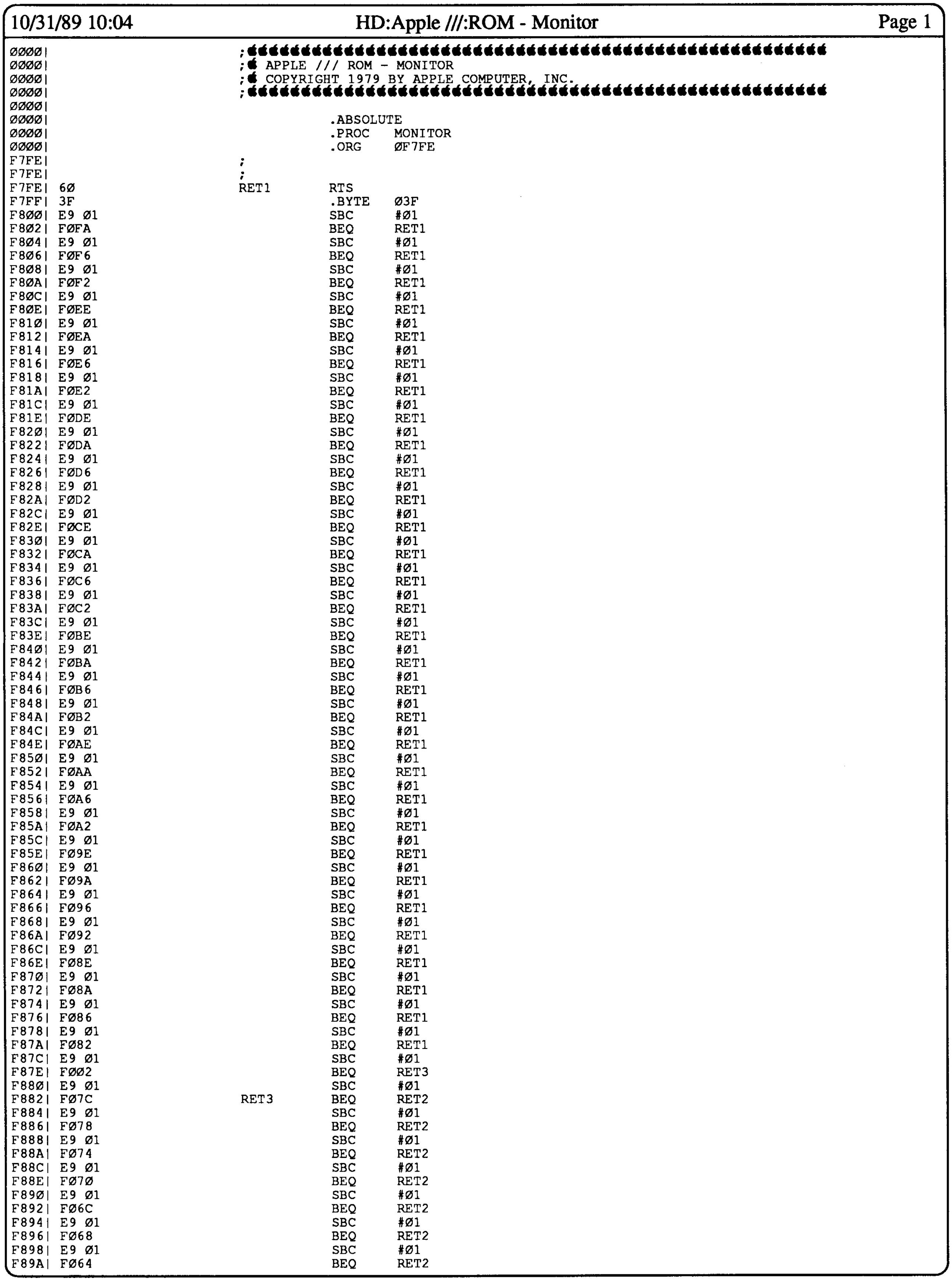
```
    SEC :    2  |  **
    STA :   3Ø  |  ******************************
    STX :   18  |  ******************
    STY :    4  |  ****
    TAX :    4  |  ****
    TAY :    2  |  **
    TXA :    6  |  ******
    TXS :    2  |  **
    TYA :    4  |  ****

    Minimum frequency =    1
    Maximum frequency =   56

    Average frequency =    8

    Unused opcodes:

    BRK  BVC  BVS  CLI  CLV  NOP  ROL  RTI  SED  SEI  TSX

    Program opcode usage:  8Ø %

--------------------------------------------------------------------------------------------------
(1.ØØ) That's all, Folks ...
--------------------------------------------------------------------------------------------------
```



# Source Code Listing
for
Apple ///

ROM - Monitor

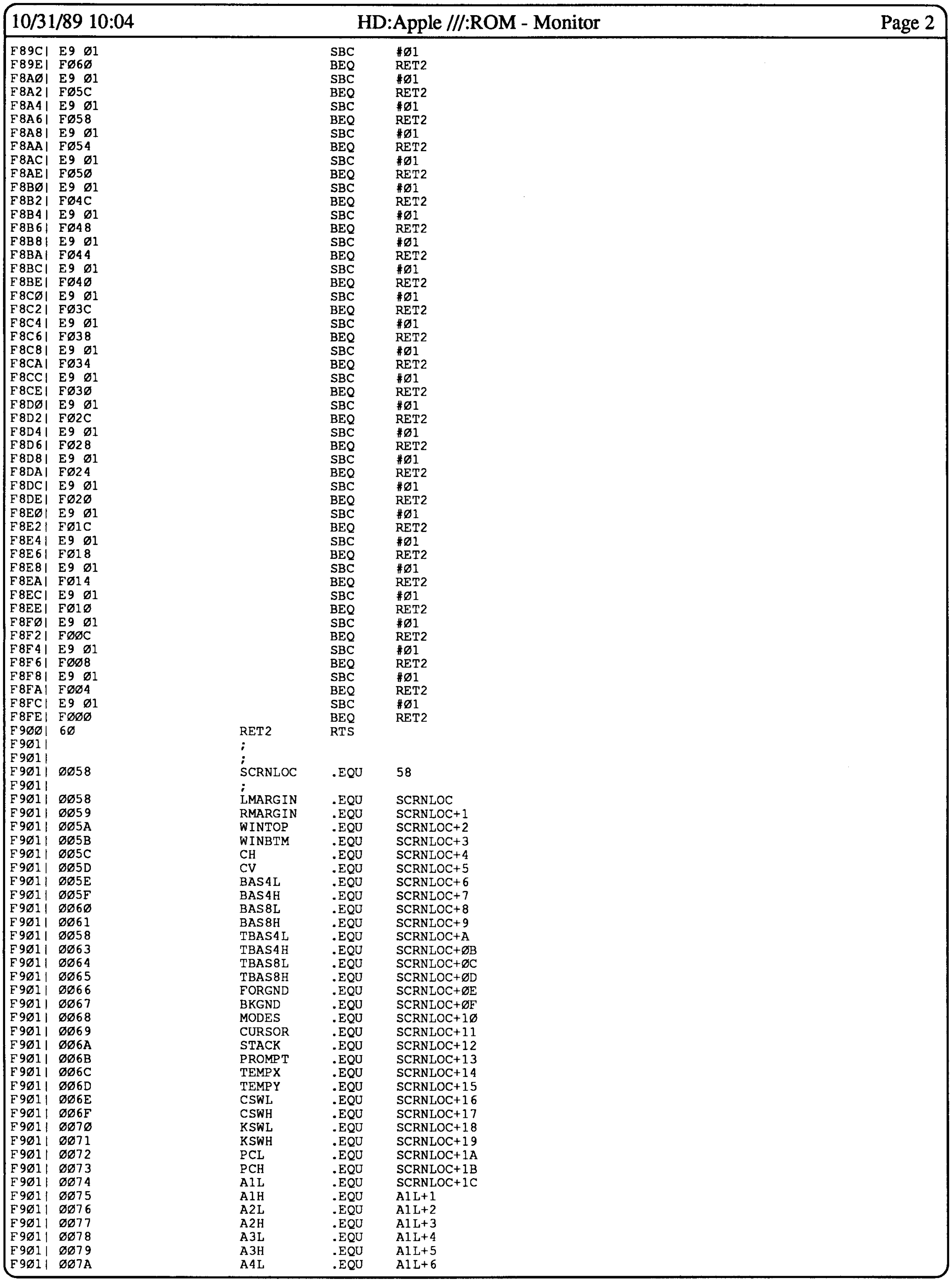
David T. Craig
736 Edgewater
Wichita, Kansas 67230



10/31/89 10:04 HD:Apple ///:ROM - Monitor Page 1

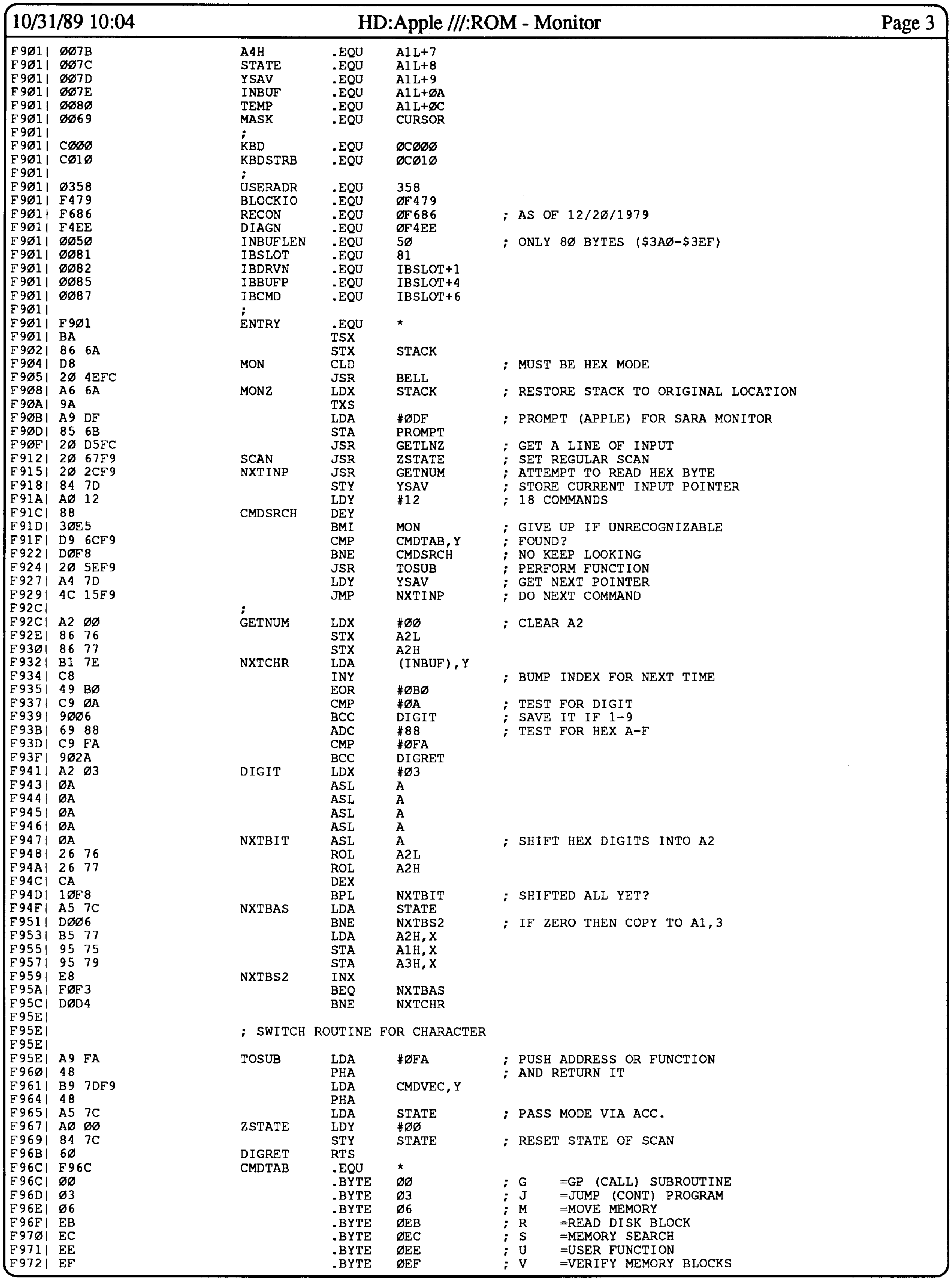
```
ØØØØ|                         ;
ØØØØ|                         ; APPLE /// ROM - MONITOR
ØØØØ|                         ; COPYRIGHT 1979 BY APPLE COMPUTER, INC.
ØØØØ|                         ;
ØØØØ|
ØØØØ|                                     .ABSOLUTE
ØØØØ|                                     .PROC   MONITOR
ØØØØ|                                     .ORG    ØF7FE
F7FE|                         ;
F7FE|                         ;
F7FE| 6Ø                      RET1        RTS
F7FF| 3F                                  .BYTE   Ø3F
F8ØØ| E9 Ø1                               SBC     #Ø1
F8Ø2| FØFA                                BEQ     RET1
F8Ø4| E9 Ø1                               SBC     #Ø1
F8Ø6| FØF6                                BEQ     RET1
F8Ø8| E9 Ø1                               SBC     #Ø1
F8ØA| FØF2                                BEQ     RET1
F8ØC| E9 Ø1                               SBC     #Ø1
F8ØE| FØEE                                BEQ     RET1
F81Ø| E9 Ø1                               SBC     #Ø1
F812| FØEA                                BEQ     RET1
F814| E9 Ø1                               SBC     #Ø1
F816| FØE6                                BEQ     RET1
F818| E9 Ø1                               SBC     #Ø1
F81A| FØE2                                BEQ     RET1
F81C| E9 Ø1                               SBC     #Ø1
F81E| FØDE                                BEQ     RET1
F82Ø| E9 Ø1                               SBC     #Ø1
F822| FØDA                                BEQ     RET1
F824| E9 Ø1                               SBC     #Ø1
F826| FØD6                                BEQ     RET1
F828| E9 Ø1                               SBC     #Ø1
F82A| FØD2                                BEQ     RET1
F82C| E9 Ø1                               SBC     #Ø1
F82E| FØCE                                BEQ     RET1
F83Ø| E9 Ø1                               SBC     #Ø1
F832| FØCA                                BEQ     RET1
F834| E9 Ø1                               SBC     #Ø1
F836| FØC6                                BEQ     RET1
F838| E9 Ø1                               SBC     #Ø1
F83A| FØC2                                BEQ     RET1
F83C| E9 Ø1                               SBC     #Ø1
F83E| FØBE                                BEQ     RET1
F84Ø| E9 Ø1                               SBC     #Ø1
F842| FØBA                                BEQ     RET1
F844| E9 Ø1                               SBC     #Ø1
F846| FØB6                                BEQ     RET1
F848| E9 Ø1                               SBC     #Ø1
F84A| FØB2                                BEQ     RET1
F84C| E9 Ø1                               SBC     #Ø1
F84E| FØAE                                BEQ     RET1
F85Ø| E9 Ø1                               SBC     #Ø1
F852| FØAA                                BEQ     RET1
F854| E9 Ø1                               SBC     #Ø1
F856| FØA6                                BEQ     RET1
F858| E9 Ø1                               SBC     #Ø1
F85A| FØA2                                BEQ     RET1
F85C| E9 Ø1                               SBC     #Ø1
F85E| FØ9E                                BEQ     RET1
F86Ø| E9 Ø1                               SBC     #Ø1
F862| FØ9A                                BEQ     RET1
F864| E9 Ø1                               SBC     #Ø1
F866| FØ96                                BEQ     RET1
F868| E9 Ø1                               SBC     #Ø1
F86A| FØ92                                BEQ     RET1
F86C| E9 Ø1                               SBC     #Ø1
F86E| FØ8E                                BEQ     RET1
F87Ø| E9 Ø1                               SBC     #Ø1
F872| FØ8A                                BEQ     RET1
F874| E9 Ø1                               SBC     #Ø1
F876| FØ86                                BEQ     RET1
F878| E9 Ø1                               SBC     #Ø1
F87A| FØ82                                BEQ     RET1
F87C| E9 Ø1                               SBC     #Ø1
F87E| FØØ2                                BEQ     RET3
F88Ø| E9 Ø1                               SBC     #Ø1
F882| FØ7C                    RET3        BEQ     RET2
F884| E9 Ø1                               SBC     #Ø1
F886| FØ78                                BEQ     RET2
F888| E9 Ø1                               SBC     #Ø1
F88A| FØ74                                BEQ     RET2
F88C| E9 Ø1                               SBC     #Ø1
F88E| FØ7Ø                                BEQ     RET2
F89Ø| E9 Ø1                               SBC     #Ø1
F892| FØ6C                                BEQ     RET2
F894| E9 Ø1                               SBC     #Ø1
F896| FØ68                                BEQ     RET2
F898| E9 Ø1                               SBC     #Ø1
F89A| FØ64                                BEQ     RET2
```

“APPLE_PAT_4_383_296_C_27” 150 KB 2000-02-28 dpi: 300h x 300v pix: 2324h x 3103v



10/31/89 10:04 HD:Apple ///:ROM - Monitor Page 2

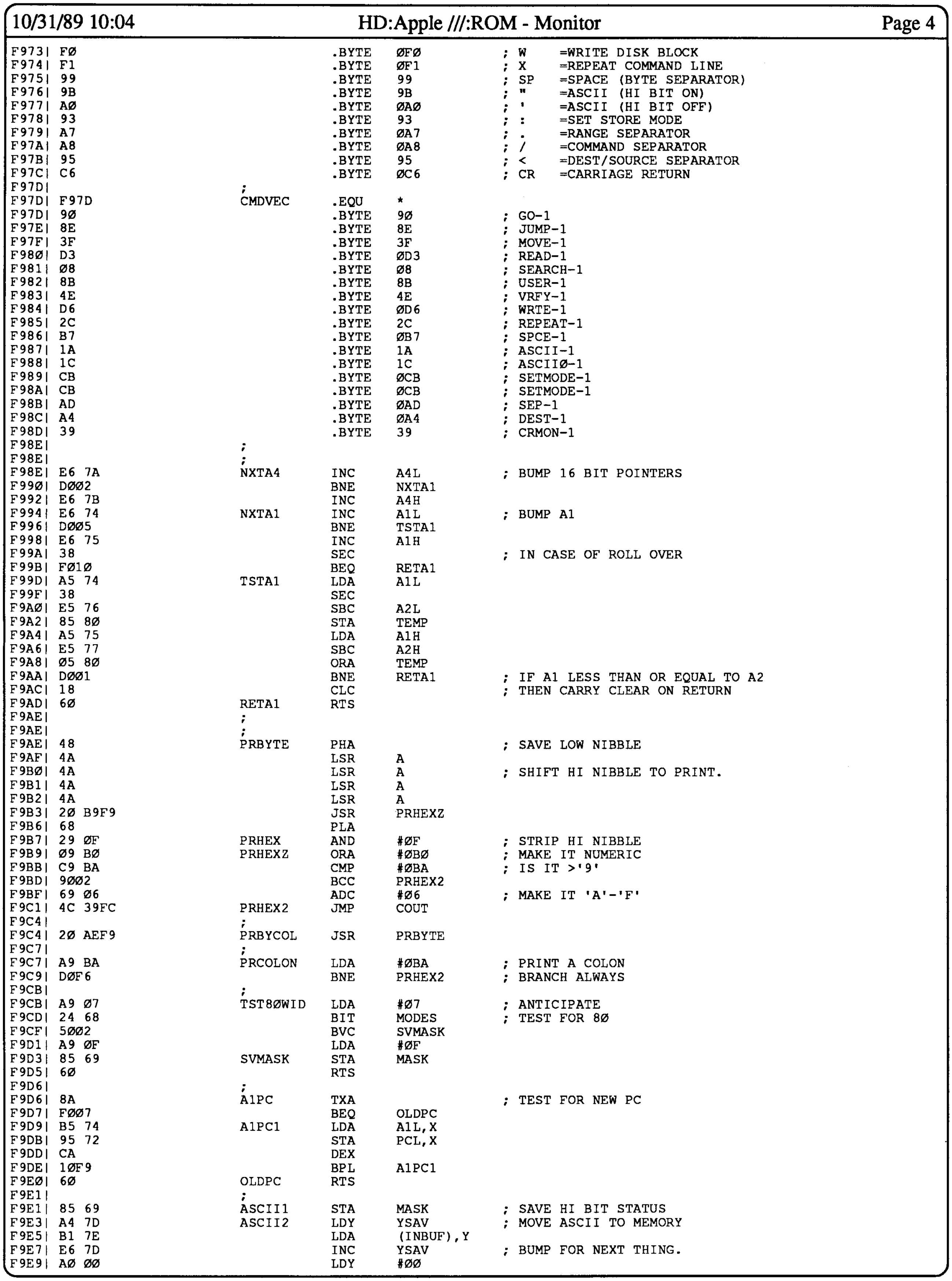
```
F89C| E9 Ø1                           SBC     #Ø1
F89E| FØ6Ø                            BEQ     RET2
F8AØ| E9 Ø1                           SBC     #Ø1
F8A2| FØ5C                            BEQ     RET2
F8A4| E9 Ø1                           SBC     #Ø1
F8A6| FØ58                            BEQ     RET2
F8A8| E9 Ø1                           SBC     #Ø1
F8AA| FØ54                            BEQ     RET2
F8AC| E9 Ø1                           SBC     #Ø1
F8AE| FØ5Ø                            BEQ     RET2
F8BØ| E9 Ø1                           SBC     #Ø1
F8B2| FØ4C                            BEQ     RET2
F8B4| E9 Ø1                           SBC     #Ø1
F8B6| FØ48                            BEQ     RET2
F8B8| E9 Ø1                           SBC     #Ø1
F8BA| FØ44                            BEQ     RET2
F8BC| E9 Ø1                           SBC     #Ø1
F8BE| FØ4Ø                            BEQ     RET2
F8CØ| E9 Ø1                           SBC     #Ø1
F8C2| FØ3C                            BEQ     RET2
F8C4| E9 Ø1                           SBC     #Ø1
F8C6| FØ38                            BEQ     RET2
F8C8| E9 Ø1                           SBC     #Ø1
F8CA| FØ34                            BEQ     RET2
F8CC| E9 Ø1                           SBC     #Ø1
F8CE| FØ3Ø                            BEQ     RET2
F8DØ| E9 Ø1                           SBC     #Ø1
F8D2| FØ2C                            BEQ     RET2
F8D4| E9 Ø1                           SBC     #Ø1
F8D6| FØ28                            BEQ     RET2
F8D8| E9 Ø1                           SBC     #Ø1
F8DA| FØ24                            BEQ     RET2
F8DC| E9 Ø1                           SBC     #Ø1
F8DE| FØ2Ø                            BEQ     RET2
F8EØ| E9 Ø1                           SBC     #Ø1
F8E2| FØ1C                            BEQ     RET2
F8E4| E9 Ø1                           SBC     #Ø1
F8E6| FØ18                            BEQ     RET2
F8E8| E9 Ø1                           SBC     #Ø1
F8EA| FØ14                            BEQ     RET2
F8EC| E9 Ø1                           SBC     #Ø1
F8EE| FØ1Ø                            BEQ     RET2
F8FØ| E9 Ø1                           SBC     #Ø1
F8F2| FØØC                            BEQ     RET2
F8F4| E9 Ø1                           SBC     #Ø1
F8F6| FØØ8                            BEQ     RET2
F8F8| E9 Ø1                           SBC     #Ø1
F8FA| FØØ4                            BEQ     RET2
F8FC| E9 Ø1                           SBC     #Ø1
F8FE| FØØØ                            BEQ     RET2
F9ØØ| 6Ø                   RET2       RTS
F9Ø1|                      ;
F9Ø1|                      ;
F9Ø1| ØØ58                 SCRNLOC    .EQU    58
F9Ø1|                      ;
F9Ø1| ØØ58                 LMARGIN    .EQU    SCRNLOC
F9Ø1| ØØ59                 RMARGIN    .EQU    SCRNLOC+1
F9Ø1| ØØ5A                 WINTOP     .EQU    SCRNLOC+2
F9Ø1| ØØ5B                 WINBTM     .EQU    SCRNLOC+3
F9Ø1| ØØ5C                 CH         .EQU    SCRNLOC+4
F9Ø1| ØØ5D                 CV         .EQU    SCRNLOC+5
F9Ø1| ØØ5E                 BAS4L      .EQU    SCRNLOC+6
F9Ø1| ØØ5F                 BAS4H      .EQU    SCRNLOC+7
F9Ø1| ØØ6Ø                 BAS8L      .EQU    SCRNLOC+8
F9Ø1| ØØ61                 BAS8H      .EQU    SCRNLOC+9
F9Ø1| ØØ58                 TBAS4L     .EQU    SCRNLOC+A
F9Ø1| ØØ63                 TBAS4H     .EQU    SCRNLOC+ØB
F9Ø1| ØØ64                 TBAS8L     .EQU    SCRNLOC+ØC
F9Ø1| ØØ65                 TBAS8H     .EQU    SCRNLOC+ØD
F9Ø1| ØØ66                 FORGND     .EQU    SCRNLOC+ØE
F9Ø1| ØØ67                 BKGND      .EQU    SCRNLOC+ØF
F9Ø1| ØØ68                 MODES      .EQU    SCRNLOC+1Ø
F9Ø1| ØØ69                 CURSOR     .EQU    SCRNLOC+11
F9Ø1| ØØ6A                 STACK      .EQU    SCRNLOC+12
F9Ø1| ØØ6B                 PROMPT     .EQU    SCRNLOC+13
F9Ø1| ØØ6C                 TEMPX      .EQU    SCRNLOC+14
F9Ø1| ØØ6D                 TEMPY      .EQU    SCRNLOC+15
F9Ø1| ØØ6E                 CSWL       .EQU    SCRNLOC+16
F9Ø1| ØØ6F                 CSWH       .EQU    SCRNLOC+17
F9Ø1| ØØ7Ø                 KSWL       .EQU    SCRNLOC+18
F9Ø1| ØØ71                 KSWH       .EQU    SCRNLOC+19
F9Ø1| ØØ72                 PCL        .EQU    SCRNLOC+1A
F9Ø1| ØØ73                 PCH        .EQU    SCRNLOC+1B
F9Ø1| ØØ74                 A1L        .EQU    SCRNLOC+1C
F9Ø1| ØØ75                 A1H        .EQU    A1L+1
F9Ø1| ØØ76                 A2L        .EQU    A1L+2
F9Ø1| ØØ77                 A2H        .EQU    A1L+3
F9Ø1| ØØ78                 A3L        .EQU    A1L+4
F9Ø1| ØØ79                 A3H        .EQU    A1L+5
F9Ø1| ØØ7A                 A4L        .EQU    A1L+6
```



10/31/89 10:04 HD:Apple ///:ROM - Monitor Page 3

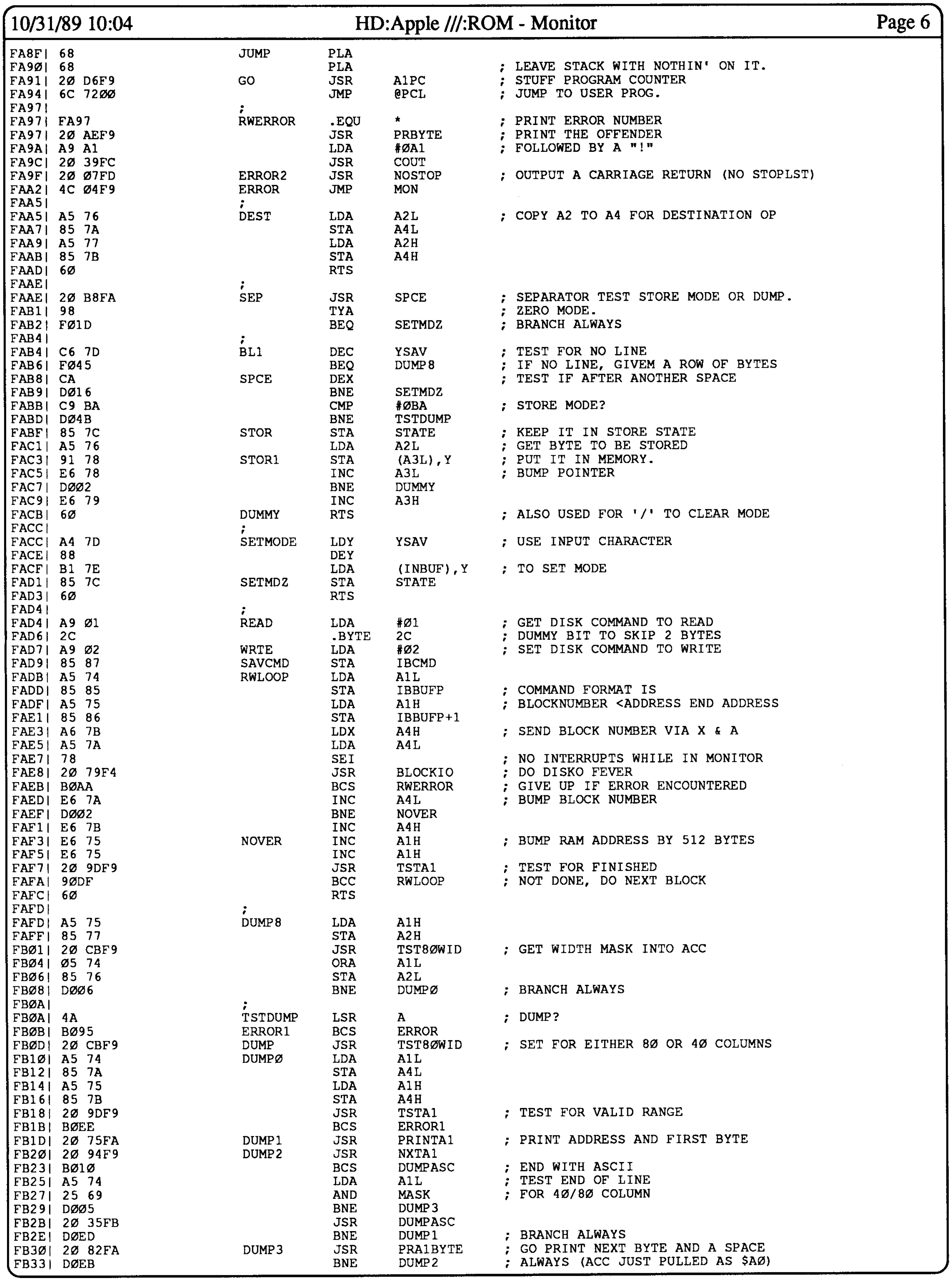
```
F9Ø1| ØØ7B                  A4H        .EQU    A1L+7
F9Ø1| ØØ7C                  STATE      .EQU    A1L+8
F9Ø1| ØØ7D                  YSAV       .EQU    A1L+9
F9Ø1| ØØ7E                  INBUF      .EQU    A1L+ØA
F9Ø1| ØØ8Ø                  TEMP       .EQU    A1L+ØC
F9Ø1| ØØ69                  MASK       .EQU    CURSOR
F9Ø1|                       ;
F9Ø1| CØØØ                  KBD        .EQU    ØCØØØ
F9Ø1| CØ1Ø                  KBDSTRB    .EQU    ØCØ1Ø
F9Ø1|                       ;
F9Ø1| Ø358                  USERADR    .EQU    358
F9Ø1| F479                  BLOCKIO    .EQU    ØF479
F9Ø1| F686                  RECON      .EQU    ØF686        ; AS OF 12/2Ø/1979
F9Ø1| F4EE                  DIAGN      .EQU    ØF4EE
F9Ø1| ØØ5Ø                  INBUFLEN   .EQU    5Ø           ; ONLY 8Ø BYTES ($3AØ-$3EF)
F9Ø1| ØØ81                  IBSLOT     .EQU    81
F9Ø1| ØØ82                  IBDRVN     .EQU    IBSLOT+1
F9Ø1| ØØ85                  IBBUFP     .EQU    IBSLOT+4
F9Ø1| ØØ87                  IBCMD      .EQU    IBSLOT+6
F9Ø1|                       ;
F9Ø1| F9Ø1                  ENTRY      .EQU    *
F9Ø1| BA                               TSX
F9Ø2| 86 6A                            STX     STACK
F9Ø4| D8                    MON        CLD                  ; MUST BE HEX MODE
F9Ø5| 2Ø 4EFC                          JSR     BELL
F9Ø8| A6 6A                 MONZ       LDX     STACK        ; RESTORE STACK TO ORIGINAL LOCATION
F9ØA| 9A                               TXS
F9ØB| A9 DF                            LDA     #ØDF         ; PROMPT (APPLE) FOR SARA MONITOR
F9ØD| 85 6B                            STA     PROMPT
F9ØF| 2Ø D5FC                          JSR     GETLNZ       ; GET A LINE OF INPUT
F912| 2Ø 67F9               SCAN       JSR     ZSTATE       ; SET REGULAR SCAN
F915| 2Ø 2CF9               NXTINP     JSR     GETNUM       ; ATTEMPT TO READ HEX BYTE
F918| 84 7D                            STY     YSAV         ; STORE CURRENT INPUT POINTER
F91A| AØ 12                            LDY     #12          ; 18 COMMANDS
F91C| 88                    CMDSRCH    DEY
F91D| 3ØE5                             BMI     MON          ; GIVE UP IF UNRECOGNIZABLE
F91F| D9 6CF9                          CMP     CMDTAB,Y     ; FOUND?
F922| DØF8                             BNE     CMDSRCH      ; NO KEEP LOOKING
F924| 2Ø 5EF9                          JSR     TOSUB        ; PERFORM FUNCTION
F927| A4 7D                            LDY     YSAV         ; GET NEXT POINTER
F929| 4C 15F9                          JMP     NXTINP       ; DO NEXT COMMAND
F92C|                       ;
F92C| A2 ØØ                 GETNUM     LDX     #ØØ          ; CLEAR A2
F92E| 86 76                            STX     A2L
F93Ø| 86 77                            STX     A2H
F932| B1 7E                 NXTCHR     LDA     (INBUF),Y
F934| C8                               INY                  ; BUMP INDEX FOR NEXT TIME
F935| 49 BØ                            EOR     #ØBØ
F937| C9 ØA                            CMP     #ØA          ; TEST FOR DIGIT
F939| 9ØØ6                             BCC     DIGIT        ; SAVE IT IF 1-9
F93B| 69 88                            ADC     #88          ; TEST FOR HEX A-F
F93D| C9 FA                            CMP     #ØFA
F93F| 9Ø2A                             BCC     DIGRET
F941| A2 Ø3                 DIGIT      LDX     #Ø3
F943| ØA                               ASL     A
F944| ØA                               ASL     A
F945| ØA                               ASL     A
F946| ØA                               ASL     A
F947| ØA                    NXTBIT     ASL     A            ; SHIFT HEX DIGITS INTO A2
F948| 26 76                            ROL     A2L
F94A| 26 77                            ROL     A2H
F94C| CA                               DEX
F94D| 1ØF8                             BPL     NXTBIT       ; SHIFTED ALL YET?
F94F| A5 7C                 NXTBAS     LDA     STATE
F951| DØØ6                             BNE     NXTBS2       ; IF ZERO THEN COPY TO A1,3
F953| B5 77                            LDA     A2H,X
F955| 95 75                            STA     A1H,X
F957| 95 79                            STA     A3H,X
F959| E8                    NXTBS2     INX
F95A| FØF3                             BEQ     NXTBAS
F95C| DØD4                             BNE     NXTCHR
F95E|
F95E|                       ; SWITCH ROUTINE FOR CHARACTER
F95E|
F95E| A9 FA                 TOSUB      LDA     #ØFA         ; PUSH ADDRESS OR FUNCTION
F96Ø| 48                               PHA                  ; AND RETURN IT
F961| B9 7DF9                          LDA     CMDVEC,Y
F964| 48                               PHA
F965| A5 7C                            LDA     STATE        ; PASS MODE VIA ACC.
F967| AØ ØØ                 ZSTATE     LDY     #ØØ
F969| 84 7C                            STY     STATE        ; RESET STATE OF SCAN
F96B| 6Ø                    DIGRET     RTS
F96C| F96C                  CMDTAB     .EQU    *
F96C| ØØ                               .BYTE   ØØ           ; G    =GP (CALL) SUBROUTINE
F96D| Ø3                               .BYTE   Ø3           ; J    =JUMP (CONT) PROGRAM
F96E| Ø6                               .BYTE   Ø6           ; M    =MOVE MEMORY
F96F| EB                               .BYTE   ØEB          ; R    =READ DISK BLOCK
F97Ø| EC                               .BYTE   ØEC          ; S    =MEMORY SEARCH
F971| EE                               .BYTE   ØEE          ; U    =USER FUNCTION
F972| EF                               .BYTE   ØEF          ; V    =VERIFY MEMORY BLOCKS
```

“APPLE_PAT_4_383_296_C_29” 183 KB 2000-02-28 dpi: 300h x 300v pix: 2324h x 3091v



```
F973| FØ                         .BYTE    ØFØ          ; W    =WRITE DISK BLOCK
F974| F1                         .BYTE    ØF1          ; X    =REPEAT COMMAND LINE
F975| 99                         .BYTE    99           ; SP   =SPACE (BYTE SEPARATOR)
F976| 9B                         .BYTE    9B           ; "    =ASCII (HI BIT ON)
F977| AØ                         .BYTE    ØAØ          ; '    =ASCII (HI BIT OFF)
F978| 93                         .BYTE    93           ; :    =SET STORE MODE
F979| A7                         .BYTE    ØA7          ; .    =RANGE SEPARATOR
F97A| A8                         .BYTE    ØA8          ; /    =COMMAND SEPARATOR
F97B| 95                         .BYTE    95           ; <    =DEST/SOURCE SEPARATOR
F97C| C6                         .BYTE    ØC6          ; CR   =CARRIAGE RETURN
F97D|                   ;
F97D| F97D              CMDVEC   .EQU     *
F97D| 9Ø                         .BYTE    9Ø           ; GO-1
F97E| 8E                         .BYTE    8E           ; JUMP-1
F97F| 3F                         .BYTE    3F           ; MOVE-1
F98Ø| D3                         .BYTE    ØD3          ; READ-1
F981| Ø8                         .BYTE    Ø8           ; SEARCH-1
F982| 8B                         .BYTE    8B           ; USER-1
F983| 4E                         .BYTE    4E           ; VRFY-1
F984| D6                         .BYTE    ØD6          ; WRTE-1
F985| 2C                         .BYTE    2C           ; REPEAT-1
F986| B7                         .BYTE    ØB7          ; SPCE-1
F987| 1A                         .BYTE    1A           ; ASCII-1
F988| 1C                         .BYTE    1C           ; ASCIIØ-1
F989| CB                         .BYTE    ØCB          ; SETMODE-1
F98A| CB                         .BYTE    ØCB          ; SETMODE-1
F98B| AD                         .BYTE    ØAD          ; SEP-1
F98C| A4                         .BYTE    ØA4          ; DEST-1
F98D| 39                         .BYTE    39           ; CRMON-1
F98E|                   ;
F98E|                   ;
F98E| E6 7A             NXTA4    INC      A4L          ; BUMP 16 BIT POINTERS
F99Ø| DØØ2                       BNE      NXTA1
F992| E6 7B                      INC      A4H
F994| E6 74             NXTA1    INC      A1L          ; BUMP A1
F996| DØØ5                       BNE      TSTA1
F998| E6 75                      INC      A1H
F99A| 38                         SEC                   ; IN CASE OF ROLL OVER
F99B| FØ1Ø                       BEQ      RETA1
F99D| A5 74             TSTA1    LDA      A1L
F99F| 38                         SEC
F9AØ| E5 76                      SBC      A2L
F9A2| 85 8Ø                      STA      TEMP
F9A4| A5 75                      LDA      A1H
F9A6| E5 77                      SBC      A2H
F9A8| Ø5 8Ø                      ORA      TEMP
F9AA| DØØ1                       BNE      RETA1        ; IF A1 LESS THAN OR EQUAL TO A2
F9AC| 18                         CLC                   ; THEN CARRY CLEAR ON RETURN
F9AD| 6Ø                RETA1    RTS
F9AE|                   ;
F9AE|                   ;
F9AE| 48                PRBYTE   PHA                   ; SAVE LOW NIBBLE
F9AF| 4A                         LSR      A
F9BØ| 4A                         LSR      A            ; SHIFT HI NIBBLE TO PRINT.
F9B1| 4A                         LSR      A
F9B2| 4A                         LSR      A
F9B3| 2Ø B9F9                    JSR      PRHEXZ
F9B6| 68                         PLA
F9B7| 29 ØF             PRHEX    AND      #ØF          ; STRIP HI NIBBLE
F9B9| Ø9 BØ             PRHEXZ   ORA      #ØBØ         ; MAKE IT NUMERIC
F9BB| C9 BA                      CMP      #ØBA         ; IS IT >'9'
F9BD| 9ØØ2                       BCC      PRHEX2
F9BF| 69 Ø6                      ADC      #Ø6          ; MAKE IT 'A'-'F'
F9C1| 4C 39FC           PRHEX2   JMP      COUT
F9C4|                   ;
F9C4| 2Ø AEF9           PRBYCOL  JSR      PRBYTE
F9C7|                   ;
F9C7| A9 BA             PRCOLON  LDA      #ØBA         ; PRINT A COLON
F9C9| DØF6                       BNE      PRHEX2       ; BRANCH ALWAYS
F9CB|                   ;
F9CB| A9 Ø7             TST8ØWID LDA      #Ø7          ; ANTICIPATE
F9CD| 24 68                      BIT      MODES        ; TEST FOR 8Ø
F9CF| 5ØØ2                       BVC      SVMASK
F9D1| A9 ØF                      LDA      #ØF
F9D3| 85 69             SVMASK   STA      MASK
F9D5| 6Ø                         RTS
F9D6|                   ;
F9D6| 8A                A1PC     TXA                   ; TEST FOR NEW PC
F9D7| FØØ7                       BEQ      OLDPC
F9D9| B5 74             A1PC1    LDA      A1L,X
F9DB| 95 72                      STA      PCL,X
F9DD| CA                         DEX
F9DE| 1ØF9                       BPL      A1PC1
F9EØ| 6Ø                OLDPC    RTS
F9E1|                   ;
F9E1| 85 69             ASCII1   STA      MASK         ; SAVE HI BIT STATUS
F9E3| A4 7D             ASCII2   LDY      YSAV         ; MOVE ASCII TO MEMORY
F9E5| B1 7E                      LDA      (INBUF),Y
F9E7| E6 7D                      INC      YSAV         ; BUMP FOR NEXT THING.
F9E9| AØ ØØ                      LDY      #ØØ
```

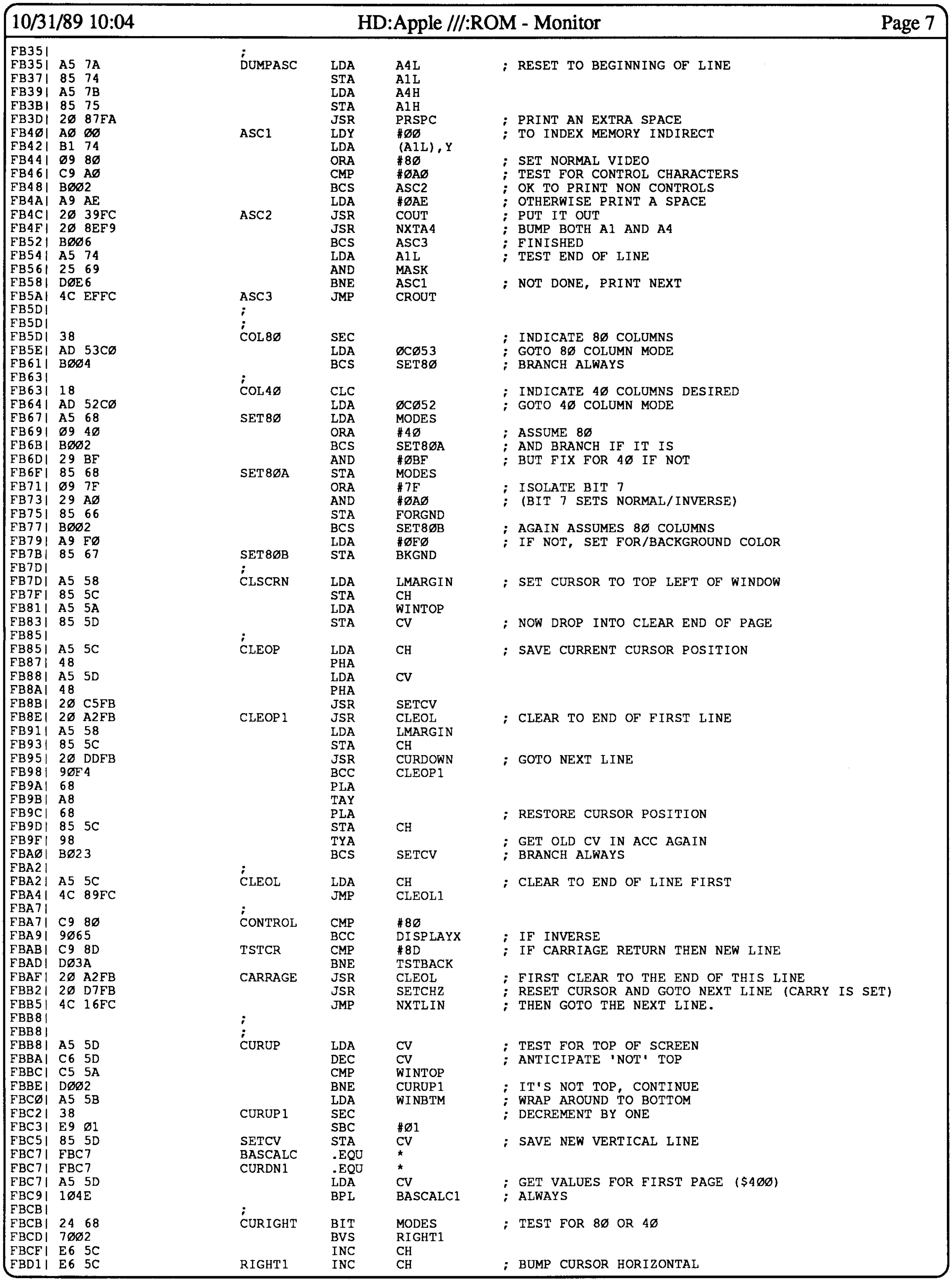
```
F9EB| C9 A2                            CMP     #ØA2         ; ASCII " ?
F9ED| DØØ5                             BNE     ASCII3       ; NOPE, CONTINUE.
F9EF| A5 69                            LDA     MASK
F9F1| 1Ø32                             BPL     BITON        ; HE'S CHANGED MODES.
F9F3| 6Ø                               RTS
F9F4| C9 A7                 ASCII3     CMP     #ØA7         ; ASCII ' ?
F9F6| DØØ5                             BNE     CRCHK        ; NO, TEST FOR EOL.
F9F8| A5 69                            LDA     MASK
F9FA| 3Ø2D                             BMI     BITOFF       ; CHANGE MODES.
F9FC| 6Ø                               RTS
F9FD|
F9FD| C9 8D                 CRCHK      CMP     #8D          ; END OF LINE?
F9FF| FØØ7                             BEQ     ASCDONE      ; YES, FINISHED
FAØ1| 25 69                            AND     MASK
FAØ3| 2Ø C3FA                          JSR     STOR1        ; GO STORE IT!
FAØ6| DØDB                             BNE     ASCII2       ; DO NEXT.
FAØ8| 6Ø                    ASCDONE    RTS
FAØ9|                       ;
FAØ9|                       ;
FAØ9| B1 74                 SEARCH     LDA     (A1L),Y      ; LOAD SEARCH BYTE
FAØB| C5 7A                            CMP     A4L
FAØD| DØØ6                             BNE     SRCH1
FAØF| 2Ø 75FA                          JSR     PRINTA1      ; DUMP MEMORY
FA12| 2Ø EFFC                          JSR     CROUT
FA15| 2Ø 94F9               SRCH1      JSR     NXTA1        ; INCREMENT POINTER
FA18| 9ØEF                             BCC     SEARCH       ; CONTINUE SEARCH
FA1A| 6Ø                               RTS                  ; RETURN
FA1B|                       ;
FA1B|                       ;
FA1B| 38                    ASCII      SEC                  ; INDICATE HI ON.
FA1C| 9Ø                               .BYTE   9Ø           ; (BCC - NEVER TAKEN)
FA1D| 18                    ASCIIØ     CLC                  ; INDICATE HI OFF
FA1E| AA                    CKMDE      TAX                  ; SAVE STATE
FA1F| 86 7C                            STX     STATE        ; RETAIN STATE
FA21| 49 BA                            EOR     #ØBA         ; ARE WE IN STORE MODE?
FA23| DØ7D                             BNE     ERROR
FA25| A9 FF                 BITON      LDA     #ØFF         ; SET HI BIT UNMASKED
FA27| BØB8                             BCS     ASCII1
FA29| A9 7F                 BITOFF     LDA     #7F          ; MASK HI BIT
FA2B| 1ØB4                             BPL     ASCII1       ; ALWAYS BRANCHES
FA2D| 2C ØØCØ               REPEAT     BIT     KBD          ; REPEAT UNTIL KEYPRESS
FA3Ø| 1ØØ3                             BPL     REPEAT1
FA32| 4C ØFFD                          JMP     KEYIN
FA35| 68                    REPEAT1    PLA                  ; CLEAN UP STACK
FA36| 68                    LFA36      PLA
FA37| 4C 12F9                          JMP     SCAN
FA3A|                       ;
FA3A|                       ;
FA3A| 2Ø B4FA               CRMON      JSR     BL1
FA3D| 4C Ø8F9                          JMP     MONZ
FA4Ø|                       ;
FA4Ø|                       ;
FA4Ø| 2Ø 9DF9               MOVE       JSR     TSTA1        ; TEST VALID RANGE
FA43| BØ5D                             BCS     ERROR
FA45| B1 74                 MOVNXT     LDA     (A1L),Y      ; COMPARE BYTE FOR BYTE
FA47| 91 7A                            STA     (A4L),Y
FA49| 2Ø 8EF9                          JSR     NXTA4        ; BUMP BOTH A1 AND A4
FA4C| 9ØF7                             BCC     MOVNXT
FA4E| 6Ø                               RTS                  ; ALL DONE WITH MOVE
FA4F|                       ;
FA4F|                       ;
FA4F| 2Ø 9DF9               VRFY       JSR     TSTA1        ; TEST VALID RANGE
FA52| BØ4E                             BCS     ERROR
FA54| B1 74                 VRFY1      LDA     (A1L),Y      ; COMPARE BYTE FOR BYTE
FA56| D1 7A                            CMP     (A4L),Y      ; MATCH?
FA58| FØØ6                             BEQ     VRFY2        ; YES, DO NEXT.
FA5A| 2Ø 66FA                          JSR     MISMATCH     ; PRINT BOTH BYTES
FA5D| 2Ø EFFC                          JSR     CROUT        ; GOTO NEWLINE
FA6Ø| 2Ø 8EF9               VRFY2      JSR     NXTA4        ; BUMP BOTH A1 AND A4
FA63| 9ØEF                             BCC     VRFY1
FA65| 6Ø                               RTS                  ; VERIFY DONE.
FA66|                       ;
FA66| A5 7B                 MISMATCH   LDA     A4H          ; PRINT ADDRESS OF A4
FA68| 2Ø AEF9                          JSR     PRBYTE
FA6B| A5 7A                            LDA     A4L
FA6D| 2Ø C4F9                          JSR     PRBYCOL      ; OUTPUT A COLON FOR SEPARATOR
FA7Ø| B1 7A                            LDA     (A4L),Y      ; AND THE DATA
FA72| 2Ø 84FA                          JSR     PRBYTSP      ; PRINT THE BYTE AND A SPACE
FA75| 2Ø 87FA               PRINTA1    JSR     PRSPC        ; LEAD WITH A SPACE
FA78| A5 75                            LDA     A1H          ; OUTPUT ADDRESS A1
FA7A| 2Ø AEF9                          JSR     PRBYTE
FA7D| A5 74                            LDA     A1L
FA7F| 2Ø C4F9                          JSR     PRBYCOL      ; SEPARATE WITH A COLON
FA82| B1 74                 PRA1BYTE   LDA     (A1L),Y      ; PRINT BYTE POINTED TO BY A1
FA84| 2Ø AEF9               PRBYTSP    JSR     PRBYTE
FA87| A9 AØ                 PRSPC      LDA     #ØAØ         ; PRINT A SPACE
FA89| 4C 39FC                          JMP     COUT         ; END VIA OUTPUT ROUTINE.
FA8C|                       ;
FA8C| 4C 58Ø3               USER       JMP     USERADR
FA8F|                       ;
```



10/31/89 10:04 HD:Apple ///:ROM - Monitor Page 6

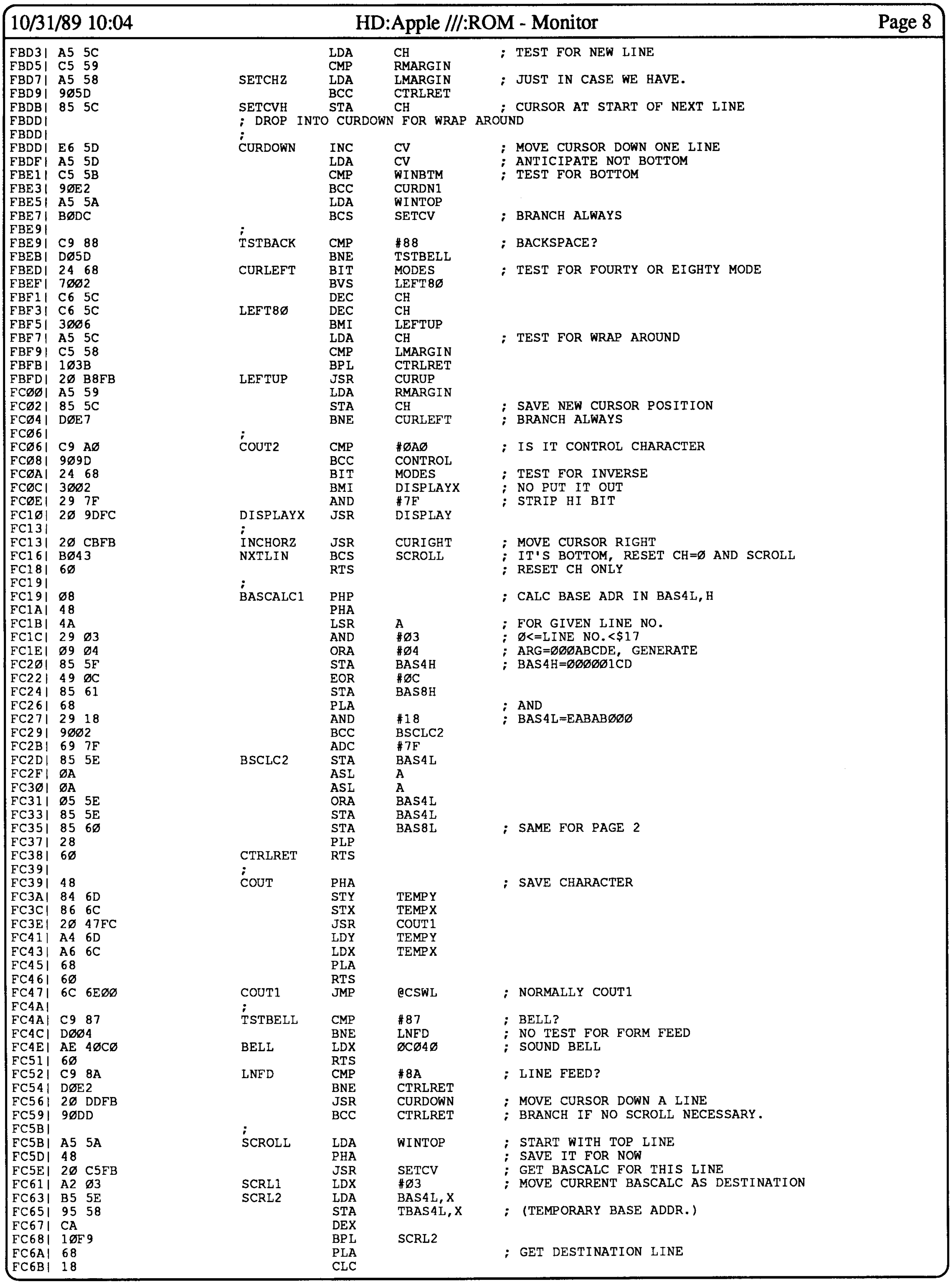
```
FA8F| 68                  JUMP      PLA
FA9Ø| 68                            PLA                      ; LEAVE STACK WITH NOTHIN' ON IT.
FA91| 2Ø D6F9             GO        JSR      A1PC            ; STUFF PROGRAM COUNTER
FA94| 6C 72ØØ                       JMP      @PCL            ; JUMP TO USER PROG.
FA97|                     ;
FA97| FA97                RWERROR   .EQU     *               ; PRINT ERROR NUMBER
FA97| 2Ø AEF9                       JSR      PRBYTE          ; PRINT THE OFFENDER
FA9A| A9 A1                         LDA      #ØA1            ; FOLLOWED BY A "!"
FA9C| 2Ø 39FC                       JSR      COUT
FA9F| 2Ø Ø7FD             ERROR2    JSR      NOSTOP          ; OUTPUT A CARRIAGE RETURN (NO STOPLST)
FAA2| 4C Ø4F9             ERROR     JMP      MON
FAA5|                     ;
FAA5| A5 76               DEST      LDA      A2L             ; COPY A2 TO A4 FOR DESTINATION OP
FAA7| 85 7A                         STA      A4L
FAA9| A5 77                         LDA      A2H
FAAB| 85 7B                         STA      A4H
FAAD| 6Ø                            RTS
FAAE|                     ;
FAAE| 2Ø B8FA             SEP       JSR      SPCE            ; SEPARATOR TEST STORE MODE OR DUMP.
FAB1| 98                            TYA                      ; ZERO MODE.
FAB2| FØ1D                          BEQ      SETMDZ          ; BRANCH ALWAYS
FAB4|                     ;
FAB4| C6 7D               BL1       DEC      YSAV            ; TEST FOR NO LINE
FAB6| FØ45                          BEQ      DUMP8           ; IF NO LINE, GIVEM A ROW OF BYTES
FAB8| CA                  SPCE      DEX                      ; TEST IF AFTER ANOTHER SPACE
FAB9| DØ16                          BNE      SETMDZ
FABB| C9 BA                         CMP      #ØBA            ; STORE MODE?
FABD| DØ4B                          BNE      TSTDUMP
FABF| 85 7C               STOR      STA      STATE           ; KEEP IT IN STORE STATE
FAC1| A5 76                         LDA      A2L             ; GET BYTE TO BE STORED
FAC3| 91 78               STOR1     STA      (A3L),Y         ; PUT IT IN MEMORY.
FAC5| E6 78                         INC      A3L             ; BUMP POINTER
FAC7| DØØ2                          BNE      DUMMY
FAC9| E6 79                         INC      A3H
FACB| 6Ø                  DUMMY     RTS                      ; ALSO USED FOR '/' TO CLEAR MODE
FACC|                     ;
FACC| A4 7D               SETMODE   LDY      YSAV            ; USE INPUT CHARACTER
FACE| 88                            DEY
FACF| B1 7E                         LDA      (INBUF),Y       ; TO SET MODE
FAD1| 85 7C               SETMDZ    STA      STATE
FAD3| 6Ø                            RTS
FAD4|                     ;
FAD4| A9 Ø1               READ      LDA      #Ø1             ; GET DISK COMMAND TO READ
FAD6| 2C                            .BYTE    2C              ; DUMMY BIT TO SKIP 2 BYTES
FAD7| A9 Ø2               WRTE      LDA      #Ø2             ; SET DISK COMMAND TO WRITE
FAD9| 85 87               SAVCMD    STA      IBCMD
FADB| A5 74               RWLOOP    LDA      A1L
FADD| 85 85                         STA      IBBUFP          ; COMMAND FORMAT IS
FADF| A5 75                         LDA      A1H             ; BLOCKNUMBER <ADDRESS END ADDRESS
FAE1| 85 86                         STA      IBBUFP+1
FAE3| A6 7B                         LDX      A4H             ; SEND BLOCK NUMBER VIA X & A
FAE5| A5 7A                         LDA      A4L
FAE7| 78                            SEI                      ; NO INTERRUPTS WHILE IN MONITOR
FAE8| 2Ø 79F4                       JSR      BLOCKIO         ; DO DISKO FEVER
FAEB| BØAA                          BCS      RWERROR         ; GIVE UP IF ERROR ENCOUNTERED
FAED| E6 7A                         INC      A4L             ; BUMP BLOCK NUMBER
FAEF| DØØ2                          BNE      NOVER
FAF1| E6 7B                         INC      A4H
FAF3| E6 75               NOVER     INC      A1H             ; BUMP RAM ADDRESS BY 512 BYTES
FAF5| E6 75                         INC      A1H
FAF7| 2Ø 9DF9                       JSR      TSTA1           ; TEST FOR FINISHED
FAFA| 9ØDF                          BCC      RWLOOP          ; NOT DONE, DO NEXT BLOCK
FAFC| 6Ø                            RTS
FAFD|                     ;
FAFD| A5 75               DUMP8     LDA      A1H
FAFF| 85 77                         STA      A2H
FBØ1| 2Ø CBF9                       JSR      TST8ØWID        ; GET WIDTH MASK INTO ACC
FBØ4| Ø5 74                         ORA      A1L
FBØ6| 85 76                         STA      A2L
FBØ8| DØØ6                          BNE      DUMPØ           ; BRANCH ALWAYS
FBØA|                     ;
FBØA| 4A                  TSTDUMP   LSR      A               ; DUMP?
FBØB| BØ95                ERROR1    BCS      ERROR
FBØD| 2Ø CBF9             DUMP      JSR      TST8ØWID        ; SET FOR EITHER 8Ø OR 4Ø COLUMNS
FB1Ø| A5 74               DUMPØ     LDA      A1L
FB12| 85 7A                         STA      A4L
FB14| A5 75                         LDA      A1H
FB16| 85 7B                         STA      A4H
FB18| 2Ø 9DF9                       JSR      TSTA1           ; TEST FOR VALID RANGE
FB1B| BØEE                          BCS      ERROR1
FB1D| 2Ø 75FA             DUMP1     JSR      PRINTA1         ; PRINT ADDRESS AND FIRST BYTE
FB2Ø| 2Ø 94F9             DUMP2     JSR      NXTA1
FB23| BØ1Ø                          BCS      DUMPASC         ; END WITH ASCII
FB25| A5 74                         LDA      A1L             ; TEST END OF LINE
FB27| 25 69                         AND      MASK            ; FOR 4Ø/8Ø COLUMN
FB29| DØØ5                          BNE      DUMP3
FB2B| 2Ø 35FB                       JSR      DUMPASC
FB2E| DØED                          BNE      DUMP1           ; BRANCH ALWAYS
FB3Ø| 2Ø 82FA             DUMP3     JSR      PRA1BYTE        ; GO PRINT NEXT BYTE AND A SPACE
FB33| DØEB                          BNE      DUMP2           ; ALWAYS (ACC JUST PULLED AS $AØ)
```

“APPLE_PAT_4_383_296_C_32” 200 KB 2000-02-28 dpi: 300h x 300v pix: 2318h x 3091v



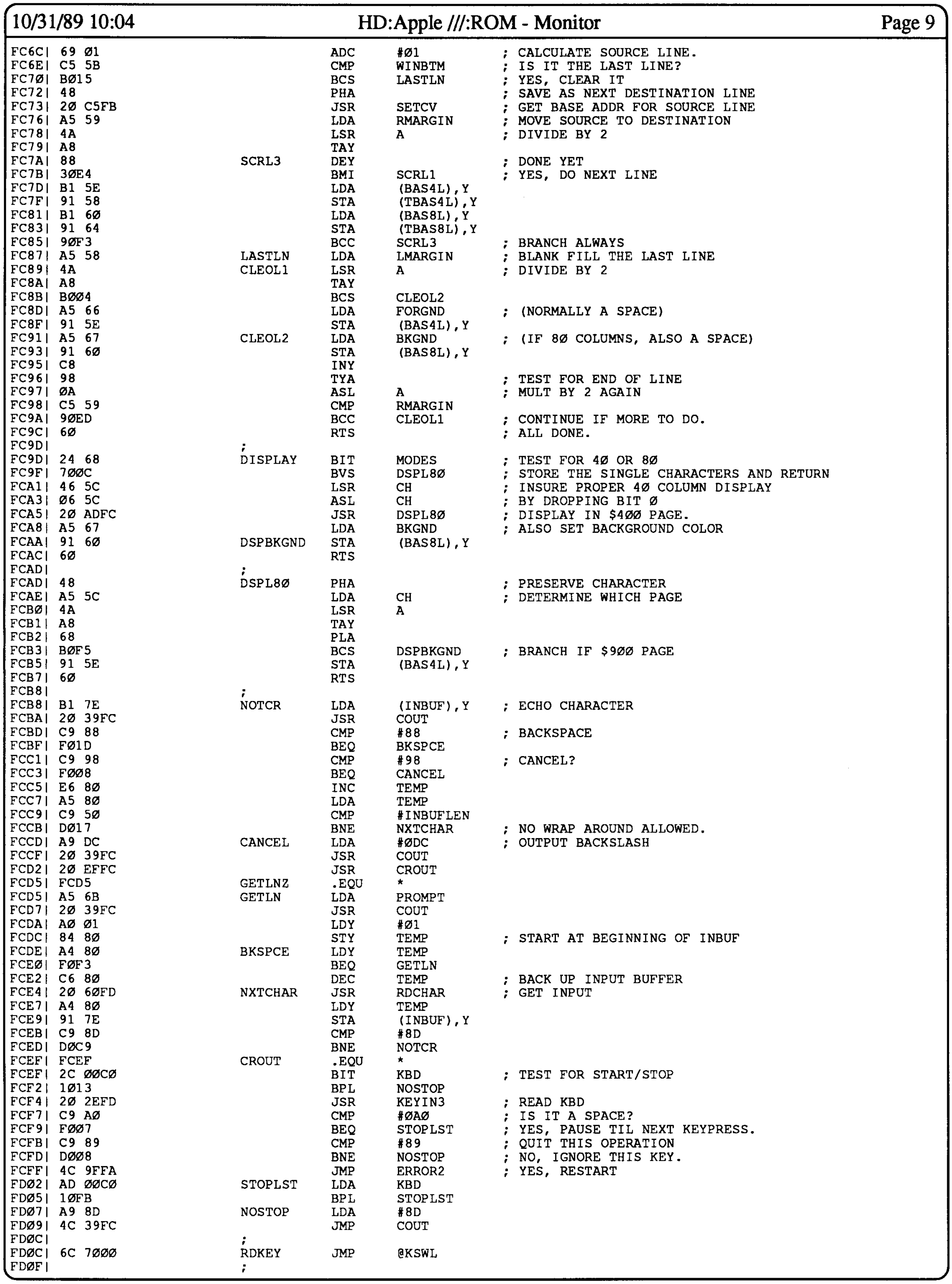
```
FB35|                          ;
FB35| A5 7A                    DUMPASC    LDA     A4L          ; RESET TO BEGINNING OF LINE
FB37| 85 74                               STA     A1L
FB39| A5 7B                               LDA     A4H
FB3B| 85 75                               STA     A1H
FB3D| 2Ø 87FA                             JSR     PRSPC        ; PRINT AN EXTRA SPACE
FB4Ø| AØ ØØ                    ASC1       LDY     #ØØ          ; TO INDEX MEMORY INDIRECT
FB42| B1 74                               LDA     (A1L),Y
FB44| Ø9 8Ø                               ORA     #8Ø          ; SET NORMAL VIDEO
FB46| C9 AØ                               CMP     #ØAØ         ; TEST FOR CONTROL CHARACTERS
FB48| BØØ2                                BCS     ASC2         ; OK TO PRINT NON CONTROLS
FB4A| A9 AE                               LDA     #ØAE         ; OTHERWISE PRINT A SPACE
FB4C| 2Ø 39FC                  ASC2       JSR     COUT         ; PUT IT OUT
FB4F| 2Ø 8EF9                             JSR     NXTA4        ; BUMP BOTH A1 AND A4
FB52| BØØ6                                BCS     ASC3         ; FINISHED
FB54| A5 74                               LDA     A1L          ; TEST END OF LINE
FB56| 25 69                               AND     MASK
FB58| DØE6                                BNE     ASC1         ; NOT DONE, PRINT NEXT
FB5A| 4C EFFC                  ASC3       JMP     CROUT
FB5D|                          ;
FB5D|                          ;
FB5D| 38                       COL8Ø      SEC                  ; INDICATE 8Ø COLUMNS
FB5E| AD 53CØ                             LDA     ØCØ53        ; GOTO 8Ø COLUMN MODE
FB61| BØØ4                                BCS     SET8Ø        ; BRANCH ALWAYS
FB63|                          ;
FB63| 18                       COL4Ø      CLC                  ; INDICATE 4Ø COLUMNS DESIRED
FB64| AD 52CØ                             LDA     ØCØ52        ; GOTO 4Ø COLUMN MODE
FB67| A5 68                    SET8Ø      LDA     MODES
FB69| Ø9 4Ø                               ORA     #4Ø          ; ASSUME 8Ø
FB6B| BØØ2                                BCS     SET8ØA       ; AND BRANCH IF IT IS
FB6D| 29 BF                               AND     #ØBF         ; BUT FIX FOR 4Ø IF NOT
FB6F| 85 68                    SET8ØA     STA     MODES
FB71| Ø9 7F                               ORA     #7F          ; ISOLATE BIT 7
FB73| 29 AØ                               AND     #ØAØ         ; (BIT 7 SETS NORMAL/INVERSE)
FB75| 85 66                               STA     FORGND
FB77| BØØ2                                BCS     SET8ØB       ; AGAIN ASSUMES 8Ø COLUMNS
FB79| A9 FØ                               LDA     #ØFØ         ; IF NOT, SET FOR/BACKGROUND COLOR
FB7B| 85 67                    SET8ØB     STA     BKGND
FB7D|                          ;
FB7D| A5 58                    CLSCRN     LDA     LMARGIN      ; SET CURSOR TO TOP LEFT OF WINDOW
FB7F| 85 5C                               STA     CH
FB81| A5 5A                               LDA     WINTOP
FB83| 85 5D                               STA     CV           ; NOW DROP INTO CLEAR END OF PAGE
FB85|                          ;
FB85| A5 5C                    CLEOP      LDA     CH           ; SAVE CURRENT CURSOR POSITION
FB87| 48                                  PHA
FB88| A5 5D                               LDA     CV
FB8A| 48                                  PHA
FB8B| 2Ø C5FB                             JSR     SETCV
FB8E| 2Ø A2FB                  CLEOP1     JSR     CLEOL        ; CLEAR TO END OF FIRST LINE
FB91| A5 58                               LDA     LMARGIN
FB93| 85 5C                               STA     CH
FB95| 2Ø DDFB                             JSR     CURDOWN      ; GOTO NEXT LINE
FB98| 9ØF4                                BCC     CLEOP1
FB9A| 68                                  PLA
FB9B| A8                                  TAY
FB9C| 68                                  PLA                  ; RESTORE CURSOR POSITION
FB9D| 85 5C                               STA     CH
FB9F| 98                                  TYA                  ; GET OLD CV IN ACC AGAIN
FBAØ| BØ23                                BCS     SETCV        ; BRANCH ALWAYS
FBA2|                          ;
FBA2| A5 5C                    CLEOL      LDA     CH           ; CLEAR TO END OF LINE FIRST
FBA4| 4C 89FC                             JMP     CLEOL1
FBA7|                          ;
FBA7| C9 8Ø                    CONTROL    CMP     #8Ø
FBA9| 9Ø65                                BCC     DISPLAYX     ; IF INVERSE
FBAB| C9 8D                    TSTCR      CMP     #8D          ; IF CARRIAGE RETURN THEN NEW LINE
FBAD| DØ3A                                BNE     TSTBACK
FBAF| 2Ø A2FB                  CARRAGE    JSR     CLEOL        ; FIRST CLEAR TO THE END OF THIS LINE
FBB2| 2Ø D7FB                             JSR     SETCHZ       ; RESET CURSOR AND GOTO NEXT LINE (CARRY IS SET)
FBB5| 4C 16FC                             JMP     NXTLIN       ; THEN GOTO THE NEXT LINE.
FBB8|                          ;
FBB8|                          ;
FBB8| A5 5D                    CURUP      LDA     CV           ; TEST FOR TOP OF SCREEN
FBBA| C6 5D                               DEC     CV           ; ANTICIPATE 'NOT' TOP
FBBC| C5 5A                               CMP     WINTOP
FBBE| DØØ2                                BNE     CURUP1       ; IT'S NOT TOP, CONTINUE
FBCØ| A5 5B                               LDA     WINBTM       ; WRAP AROUND TO BOTTOM
FBC2| 38                       CURUP1     SEC                  ; DECREMENT BY ONE
FBC3| E9 Ø1                               SBC     #Ø1
FBC5| 85 5D                    SETCV      STA     CV           ; SAVE NEW VERTICAL LINE
FBC7| FBC7                     BASCALC    .EQU    *
FBC7| FBC7                     CURDN1     .EQU    *
FBC7| A5 5D                               LDA     CV           ; GET VALUES FOR FIRST PAGE ($4ØØ)
FBC9| 1Ø4E                                BPL     BASCALC1     ; ALWAYS
FBCB|                          ;
FBCB| 24 68                    CURIGHT    BIT     MODES        ; TEST FOR 8Ø OR 4Ø
FBCD| 7ØØ2                                BVS     RIGHT1
FBCF| E6 5C                               INC     CH
FBD1| E6 5C                    RIGHT1     INC     CH           ; BUMP CURSOR HORIZONTAL
```

“APPLE_PAT_4_383_296_C_33” 197 KB 2000-02-28 dpi: 300h x 300v pix: 2324h x 3091v

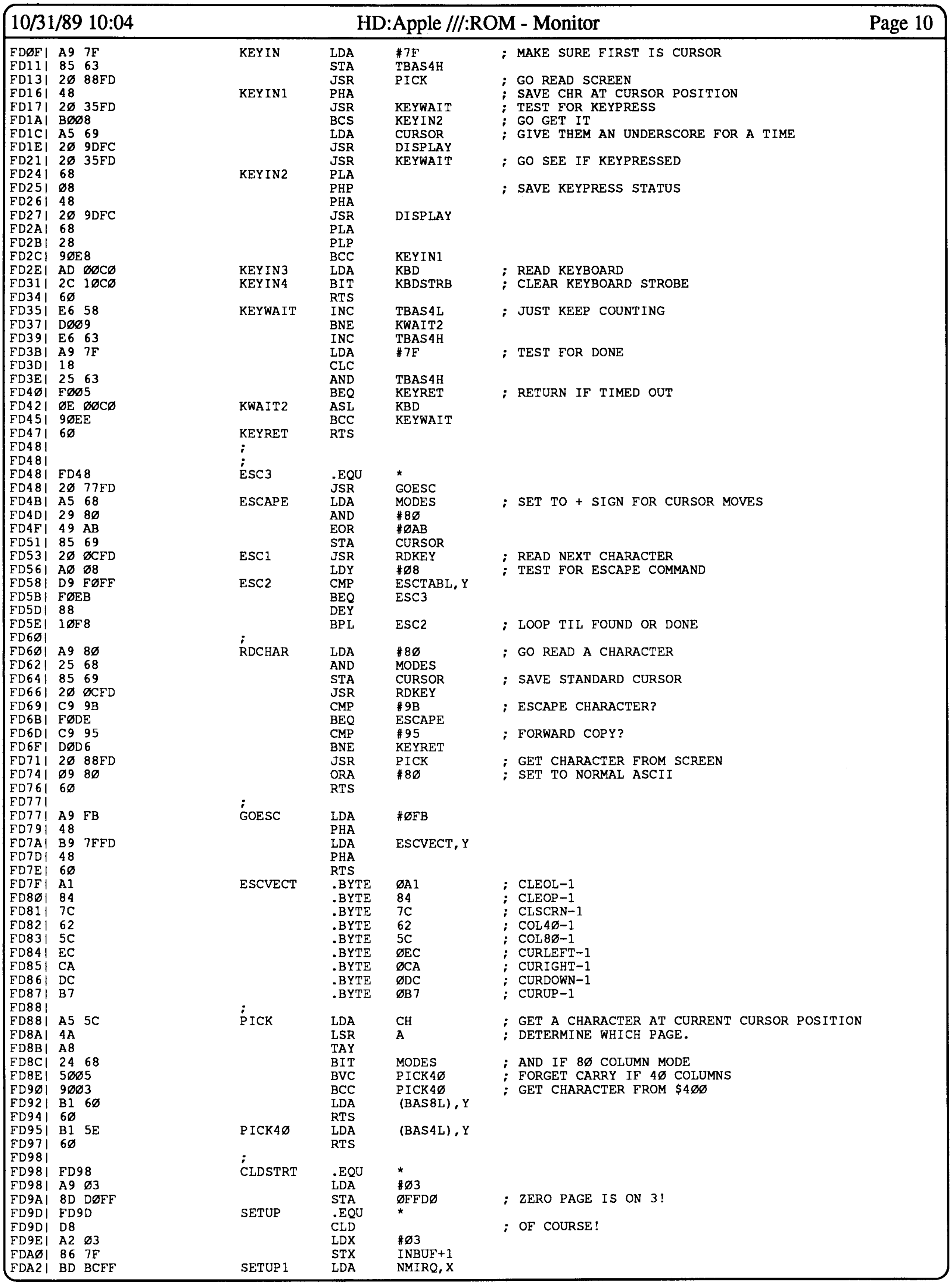


```
FBD3| A5 5C                               LDA      CH             ; TEST FOR NEW LINE
FBD5| C5 59                               CMP      RMARGIN
FBD7| A5 58                    SETCHZ     LDA      LMARGIN        ; JUST IN CASE WE HAVE.
FBD9| 9Ø5D                                BCC      CTRLRET
FBDB| 85 5C                    SETCVH     STA      CH             ; CURSOR AT START OF NEXT LINE
FBDD|                          ; DROP INTO CURDOWN FOR WRAP AROUND
FBDD|                          ;
FBDD| E6 5D                    CURDOWN    INC      CV             ; MOVE CURSOR DOWN ONE LINE
FBDF| A5 5D                               LDA      CV             ; ANTICIPATE NOT BOTTOM
FBE1| C5 5B                               CMP      WINBTM         ; TEST FOR BOTTOM
FBE3| 9ØE2                                BCC      CURDN1
FBE5| A5 5A                               LDA      WINTOP
FBE7| BØDC                                BCS      SETCV          ; BRANCH ALWAYS
FBE9|                          ;
FBE9| C9 88                    TSTBACK    CMP      #88            ; BACKSPACE?
FBEB| DØ5D                                BNE      TSTBELL
FBED| 24 68                    CURLEFT    BIT      MODES          ; TEST FOR FOURTY OR EIGHTY MODE
FBEF| 7ØØ2                                BVS      LEFT8Ø
FBF1| C6 5C                               DEC      CH
FBF3| C6 5C                    LEFT8Ø     DEC      CH
FBF5| 3ØØ6                                BMI      LEFTUP
FBF7| A5 5C                               LDA      CH             ; TEST FOR WRAP AROUND
FBF9| C5 58                               CMP      LMARGIN
FBFB| 1Ø3B                                BPL      CTRLRET
FBFD| 2Ø B8FB                  LEFTUP     JSR      CURUP
FCØØ| A5 59                               LDA      RMARGIN
FCØ2| 85 5C                               STA      CH             ; SAVE NEW CURSOR POSITION
FCØ4| DØE7                                BNE      CURLEFT        ; BRANCH ALWAYS
FCØ6|                          ;
FCØ6| C9 AØ                    COUT2      CMP      #ØAØ           ; IS IT CONTROL CHARACTER
FCØ8| 9Ø9D                                BCC      CONTROL
FCØA| 24 68                               BIT      MODES          ; TEST FOR INVERSE
FCØC| 3ØØ2                                BMI      DISPLAYX       ; NO PUT IT OUT
FCØE| 29 7F                               AND      #7F            ; STRIP HI BIT
FC1Ø| 2Ø 9DFC                  DISPLAYX   JSR      DISPLAY
FC13|                          ;
FC13| 2Ø CBFB                  INCHORZ    JSR      CURIGHT        ; MOVE CURSOR RIGHT
FC16| BØ43                     NXTLIN     BCS      SCROLL         ; IT'S BOTTOM, RESET CH=Ø AND SCROLL
FC18| 6Ø                                  RTS                     ; RESET CH ONLY
FC19|                          ;
FC19| Ø8                       BASCALC1   PHP                     ; CALC BASE ADR IN BAS4L,H
FC1A| 48                                  PHA
FC1B| 4A                                  LSR      A              ; FOR GIVEN LINE NO.
FC1C| 29 Ø3                               AND      #Ø3            ; Ø<=LINE NO.<$17
FC1E| Ø9 Ø4                               ORA      #Ø4            ; ARG=ØØØABCDE, GENERATE
FC2Ø| 85 5F                               STA      BAS4H          ; BAS4H=ØØØØØ1CD
FC22| 49 ØC                               EOR      #ØC
FC24| 85 61                               STA      BAS8H
FC26| 68                                  PLA                     ; AND
FC27| 29 18                               AND      #18            ; BAS4L=EABABØØØ
FC29| 9ØØ2                                BCC      BSCLC2
FC2B| 69 7F                               ADC      #7F
FC2D| 85 5E                    BSCLC2     STA      BAS4L
FC2F| ØA                                  ASL      A
FC3Ø| ØA                                  ASL      A
FC31| Ø5 5E                               ORA      BAS4L
FC33| 85 5E                               STA      BAS4L
FC35| 85 6Ø                               STA      BAS8L          ; SAME FOR PAGE 2
FC37| 28                                  PLP
FC38| 6Ø                       CTRLRET    RTS
FC39|                          ;
FC39| 48                       COUT       PHA                     ; SAVE CHARACTER
FC3A| 84 6D                               STY      TEMPY
FC3C| 86 6C                               STX      TEMPX
FC3E| 2Ø 47FC                             JSR      COUT1
FC41| A4 6D                               LDY      TEMPY
FC43| A6 6C                               LDX      TEMPX
FC45| 68                                  PLA
FC46| 6Ø                                  RTS
FC47| 6C 6EØØ                  COUT1      JMP      @CSWL          ; NORMALLY COUT1
FC4A|                          ;
FC4A| C9 87                    TSTBELL    CMP      #87            ; BELL?
FC4C| DØØ4                                BNE      LNFD           ; NO TEST FOR FORM FEED
FC4E| AE 4ØCØ                  BELL       LDX      ØCØ4Ø          ; SOUND BELL
FC51| 6Ø                                  RTS
FC52| C9 8A                    LNFD       CMP      #8A            ; LINE FEED?
FC54| DØE2                                BNE      CTRLRET
FC56| 2Ø DDFB                             JSR      CURDOWN        ; MOVE CURSOR DOWN A LINE
FC59| 9ØDD                                BCC      CTRLRET        ; BRANCH IF NO SCROLL NECESSARY.
FC5B|                          ;
FC5B| A5 5A                    SCROLL     LDA      WINTOP         ; START WITH TOP LINE
FC5D| 48                                  PHA                     ; SAVE IT FOR NOW
FC5E| 2Ø C5FB                             JSR      SETCV          ; GET BASCALC FOR THIS LINE
FC61| A2 Ø3                    SCRL1      LDX      #Ø3            ; MOVE CURRENT BASCALC AS DESTINATION
FC63| B5 5E                    SCRL2      LDA      BAS4L,X
FC65| 95 58                               STA      TBAS4L,X       ; (TEMPORARY BASE ADDR.)
FC67| CA                                  DEX
FC68| 1ØF9                                BPL      SCRL2
FC6A| 68                                  PLA                     ; GET DESTINATION LINE
FC6B| 18                                  CLC
```

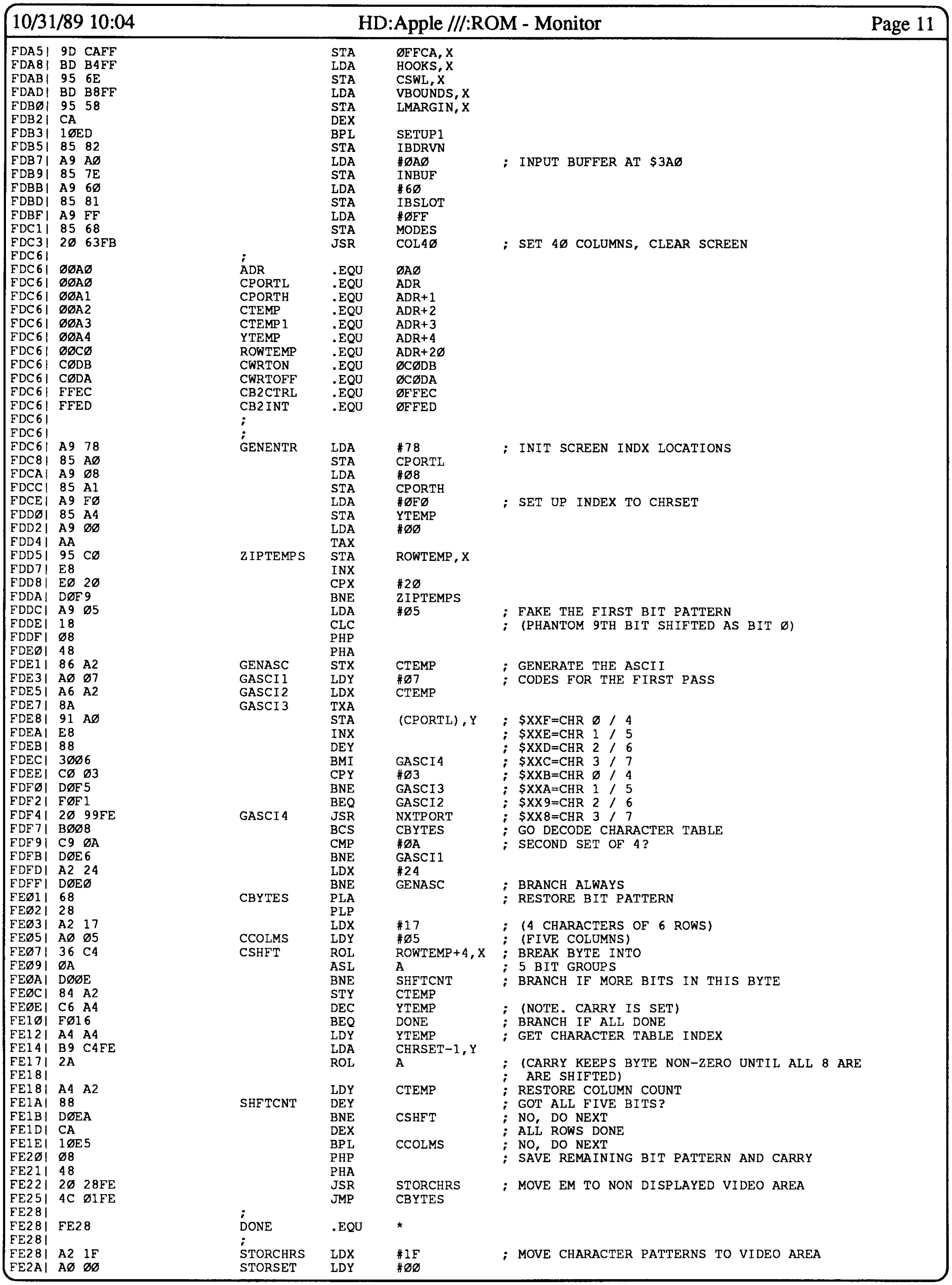
“APPLE_PAT_4_383_296_C_34” 186 KB 2000-02-28 dpi: 300h x 300v pix: 2330h x 3091v



```
FC6C| 69 Ø1                      ADC     #Ø1          ; CALCULATE SOURCE LINE.
FC6E| C5 5B                      CMP     WINBTM       ; IS IT THE LAST LINE?
FC7Ø| BØ15                       BCS     LASTLN       ; YES, CLEAR IT
FC72| 48                         PHA                  ; SAVE AS NEXT DESTINATION LINE
FC73| 2Ø C5FB                    JSR     SETCV        ; GET BASE ADDR FOR SOURCE LINE
FC76| A5 59                      LDA     RMARGIN      ; MOVE SOURCE TO DESTINATION
FC78| 4A                         LSR     A            ; DIVIDE BY 2
FC79| A8                         TAY
FC7A| 88              SCRL3      DEY                  ; DONE YET
FC7B| 3ØE4                       BMI     SCRL1        ; YES, DO NEXT LINE
FC7D| B1 5E                      LDA     (BAS4L),Y
FC7F| 91 58                      STA     (TBAS4L),Y
FC81| B1 6Ø                      LDA     (BAS8L),Y
FC83| 91 64                      STA     (TBAS8L),Y
FC85| 9ØF3                       BCC     SCRL3        ; BRANCH ALWAYS
FC87| A5 58           LASTLN     LDA     LMARGIN      ; BLANK FILL THE LAST LINE
FC89| 4A              CLEOL1     LSR     A            ; DIVIDE BY 2
FC8A| A8                         TAY
FC8B| BØØ4                       BCS     CLEOL2
FC8D| A5 66                      LDA     FORGND       ; (NORMALLY A SPACE)
FC8F| 91 5E                      STA     (BAS4L),Y
FC91| A5 67           CLEOL2     LDA     BKGND        ; (IF 8Ø COLUMNS, ALSO A SPACE)
FC93| 91 6Ø                      STA     (BAS8L),Y
FC95| C8                         INY
FC96| 98                         TYA                  ; TEST FOR END OF LINE
FC97| ØA                         ASL     A            ; MULT BY 2 AGAIN
FC98| C5 59                      CMP     RMARGIN
FC9A| 9ØED                       BCC     CLEOL1       ; CONTINUE IF MORE TO DO.
FC9C| 6Ø                         RTS                  ; ALL DONE.
FC9D|                 ;
FC9D| 24 68           DISPLAY    BIT     MODES        ; TEST FOR 4Ø OR 8Ø
FC9F| 7ØØC                       BVS     DSPL8Ø       ; STORE THE SINGLE CHARACTERS AND RETURN
FCA1| 46 5C                      LSR     CH           ; INSURE PROPER 4Ø COLUMN DISPLAY
FCA3| Ø6 5C                      ASL     CH           ; BY DROPPING BIT Ø
FCA5| 2Ø ADFC                    JSR     DSPL8Ø       ; DISPLAY IN $4ØØ PAGE.
FCA8| A5 67                      LDA     BKGND        ; ALSO SET BACKGROUND COLOR
FCAA| 91 6Ø           DSPBKGND   STA     (BAS8L),Y
FCAC| 6Ø                         RTS
FCAD|                 ;
FCAD| 48              DSPL8Ø     PHA                  ; PRESERVE CHARACTER
FCAE| A5 5C                      LDA     CH           ; DETERMINE WHICH PAGE
FCBØ| 4A                         LSR     A
FCB1| A8                         TAY
FCB2| 68                         PLA
FCB3| BØF5                       BCS     DSPBKGND     ; BRANCH IF $9ØØ PAGE
FCB5| 91 5E                      STA     (BAS4L),Y
FCB7| 6Ø                         RTS
FCB8|                 ;
FCB8| B1 7E           NOTCR      LDA     (INBUF),Y    ; ECHO CHARACTER
FCBA| 2Ø 39FC                    JSR     COUT
FCBD| C9 88                      CMP     #88          ; BACKSPACE
FCBF| FØ1D                       BEQ     BKSPCE
FCC1| C9 98                      CMP     #98          ; CANCEL?
FCC3| FØØ8                       BEQ     CANCEL
FCC5| E6 8Ø                      INC     TEMP
FCC7| A5 8Ø                      LDA     TEMP
FCC9| C9 5Ø                      CMP     #INBUFLEN
FCCB| DØ17                       BNE     NXTCHAR      ; NO WRAP AROUND ALLOWED.
FCCD| A9 DC           CANCEL     LDA     #ØDC         ; OUTPUT BACKSLASH
FCCF| 2Ø 39FC                    JSR     COUT
FCD2| 2Ø EFFC                    JSR     CROUT
FCD5| FCD5            GETLNZ     .EQU    *
FCD5| A5 6B           GETLN      LDA     PROMPT
FCD7| 2Ø 39FC                    JSR     COUT
FCDA| AØ Ø1                      LDY     #Ø1
FCDC| 84 8Ø                      STY     TEMP         ; START AT BEGINNING OF INBUF
FCDE| A4 8Ø           BKSPCE     LDY     TEMP
FCEØ| FØF3                       BEQ     GETLN
FCE2| C6 8Ø                      DEC     TEMP         ; BACK UP INPUT BUFFER
FCE4| 2Ø 6ØFD         NXTCHAR    JSR     RDCHAR       ; GET INPUT
FCE7| A4 8Ø                      LDY     TEMP
FCE9| 91 7E                      STA     (INBUF),Y
FCEB| C9 8D                      CMP     #8D
FCED| DØC9                       BNE     NOTCR
FCEF| FCEF            CROUT      .EQU    *
FCEF| 2C ØØCØ                    BIT     KBD          ; TEST FOR START/STOP
FCF2| 1Ø13                       BPL     NOSTOP
FCF4| 2Ø 2EFD                    JSR     KEYIN3       ; READ KBD
FCF7| C9 AØ                      CMP     #ØAØ         ; IS IT A SPACE?
FCF9| FØØ7                       BEQ     STOPLST      ; YES, PAUSE TIL NEXT KEYPRESS.
FCFB| C9 89                      CMP     #89          ; QUIT THIS OPERATION
FCFD| DØØ8                       BNE     NOSTOP       ; NO, IGNORE THIS KEY.
FCFF| 4C 9FFA                    JMP     ERROR2       ; YES, RESTART
FDØ2| AD ØØCØ         STOPLST    LDA     KBD
FDØ5| 1ØFB                       BPL     STOPLST
FDØ7| A9 8D           NOSTOP     LDA     #8D
FDØ9| 4C 39FC                    JMP     COUT
FDØC|                 ;
FDØC| 6C 7ØØØ         RDKEY      JMP     @KSWL
FDØF|                 ;
```

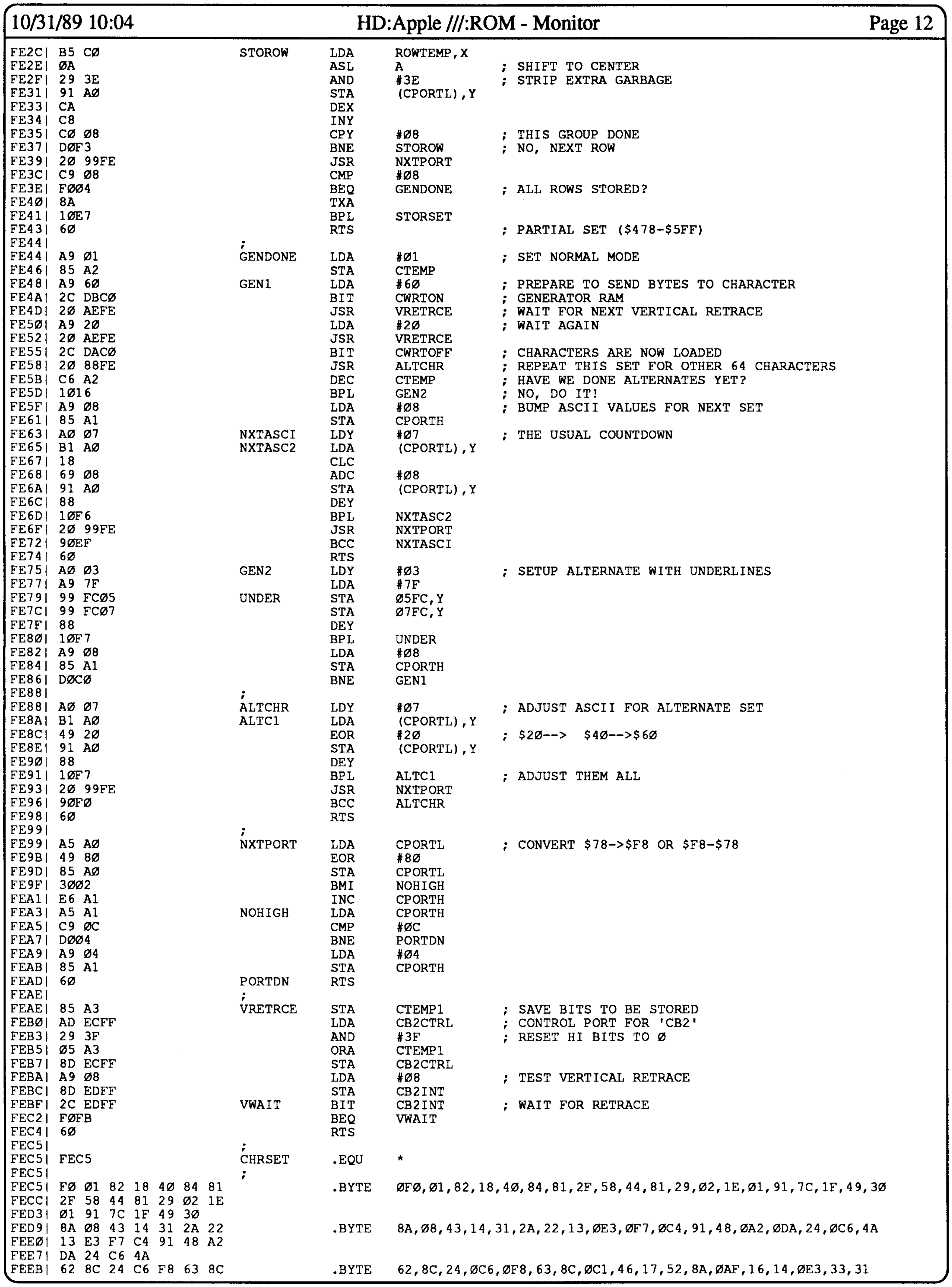


```
FDØF| A9 7F             KEYIN     LDA     #7F           ; MAKE SURE FIRST IS CURSOR
FD11| 85 63                       STA     TBAS4H
FD13| 2Ø 88FD                     JSR     PICK          ; GO READ SCREEN
FD16| 48                KEYIN1    PHA                   ; SAVE CHR AT CURSOR POSITION
FD17| 2Ø 35FD                     JSR     KEYWAIT       ; TEST FOR KEYPRESS
FD1A| BØØ8                        BCS     KEYIN2        ; GO GET IT
FD1C| A5 69                       LDA     CURSOR        ; GIVE THEM AN UNDERSCORE FOR A TIME
FD1E| 2Ø 9DFC                     JSR     DISPLAY
FD21| 2Ø 35FD                     JSR     KEYWAIT       ; GO SEE IF KEYPRESSED
FD24| 68                KEYIN2    PLA
FD25| Ø8                          PHP                   ; SAVE KEYPRESS STATUS
FD26| 48                          PHA
FD27| 2Ø 9DFC                     JSR     DISPLAY
FD2A| 68                          PLA
FD2B| 28                          PLP
FD2C| 9ØE8                        BCC     KEYIN1
FD2E| AD ØØCØ           KEYIN3    LDA     KBD           ; READ KEYBOARD
FD31| 2C 1ØCØ           KEYIN4    BIT     KBDSTRB       ; CLEAR KEYBOARD STROBE
FD34| 6Ø                          RTS
FD35| E6 58             KEYWAIT   INC     TBAS4L        ; JUST KEEP COUNTING
FD37| DØØ9                        BNE     KWAIT2
FD39| E6 63                       INC     TBAS4H
FD3B| A9 7F                       LDA     #7F           ; TEST FOR DONE
FD3D| 18                          CLC
FD3E| 25 63                       AND     TBAS4H
FD4Ø| FØØ5                        BEQ     KEYRET        ; RETURN IF TIMED OUT
FD42| ØE ØØCØ           KWAIT2    ASL     KBD
FD45| 9ØEE                        BCC     KEYWAIT
FD47| 6Ø                KEYRET    RTS
FD48|                   ;
FD48|                   ;
FD48| FD48              ESC3      .EQU    *
FD48| 2Ø 77FD                     JSR     GOESC
FD4B| A5 68             ESCAPE    LDA     MODES         ; SET TO + SIGN FOR CURSOR MOVES
FD4D| 29 8Ø                       AND     #8Ø
FD4F| 49 AB                       EOR     #ØAB
FD51| 85 69                       STA     CURSOR
FD53| 2Ø ØCFD           ESC1      JSR     RDKEY         ; READ NEXT CHARACTER
FD56| AØ Ø8                       LDY     #Ø8           ; TEST FOR ESCAPE COMMAND
FD58| D9 FØFF           ESC2      CMP     ESCTABL,Y
FD5B| FØEB                        BEQ     ESC3
FD5D| 88                          DEY
FD5E| 1ØF8                        BPL     ESC2          ; LOOP TIL FOUND OR DONE
FD6Ø|                   ;
FD6Ø| A9 8Ø             RDCHAR    LDA     #8Ø           ; GO READ A CHARACTER
FD62| 25 68                       AND     MODES
FD64| 85 69                       STA     CURSOR        ; SAVE STANDARD CURSOR
FD66| 2Ø ØCFD                     JSR     RDKEY
FD69| C9 9B                       CMP     #9B           ; ESCAPE CHARACTER?
FD6B| FØDE                        BEQ     ESCAPE
FD6D| C9 95                       CMP     #95           ; FORWARD COPY?
FD6F| DØD6                        BNE     KEYRET
FD71| 2Ø 88FD                     JSR     PICK          ; GET CHARACTER FROM SCREEN
FD74| Ø9 8Ø                       ORA     #8Ø           ; SET TO NORMAL ASCII
FD76| 6Ø                          RTS
FD77|                   ;
FD77| A9 FB             GOESC     LDA     #ØFB
FD79| 48                          PHA
FD7A| B9 7FFD                     LDA     ESCVECT,Y
FD7D| 48                          PHA
FD7E| 6Ø                          RTS
FD7F| A1                ESCVECT   .BYTE   ØA1           ; CLEOL-1
FD8Ø| 84                          .BYTE   84            ; CLEOP-1
FD81| 7C                          .BYTE   7C            ; CLSCRN-1
FD82| 62                          .BYTE   62            ; COL4Ø-1
FD83| 5C                          .BYTE   5C            ; COL8Ø-1
FD84| EC                          .BYTE   ØEC           ; CURLEFT-1
FD85| CA                          .BYTE   ØCA           ; CURIGHT-1
FD86| DC                          .BYTE   ØDC           ; CURDOWN-1
FD87| B7                          .BYTE   ØB7           ; CURUP-1
FD88|                   ;
FD88| A5 5C             PICK      LDA     CH            ; GET A CHARACTER AT CURRENT CURSOR POSITION
FD8A| 4A                          LSR     A             ; DETERMINE WHICH PAGE.
FD8B| A8                          TAY
FD8C| 24 68                       BIT     MODES         ; AND IF 8Ø COLUMN MODE
FD8E| 5ØØ5                        BVC     PICK4Ø        ; FORGET CARRY IF 4Ø COLUMNS
FD9Ø| 9ØØ3                        BCC     PICK4Ø        ; GET CHARACTER FROM $4ØØ
FD92| B1 6Ø                       LDA     (BAS8L),Y
FD94| 6Ø                          RTS
FD95| B1 5E             PICK4Ø    LDA     (BAS4L),Y
FD97| 6Ø                          RTS
FD98|                   ;
FD98| FD98              CLDSTRT   .EQU    *
FD98| A9 Ø3                       LDA     #Ø3
FD9A| 8D DØFF                     STA     ØFFDØ         ; ZERO PAGE IS ON 3!
FD9D| FD9D              SETUP     .EQU    *
FD9D| D8                          CLD                   ; OF COURSE!
FD9E| A2 Ø3                       LDX     #Ø3
FDAØ| 86 7F                       STX     INBUF+1
FDA2| BD BCFF           SETUP1    LDA     NMIRQ,X
```

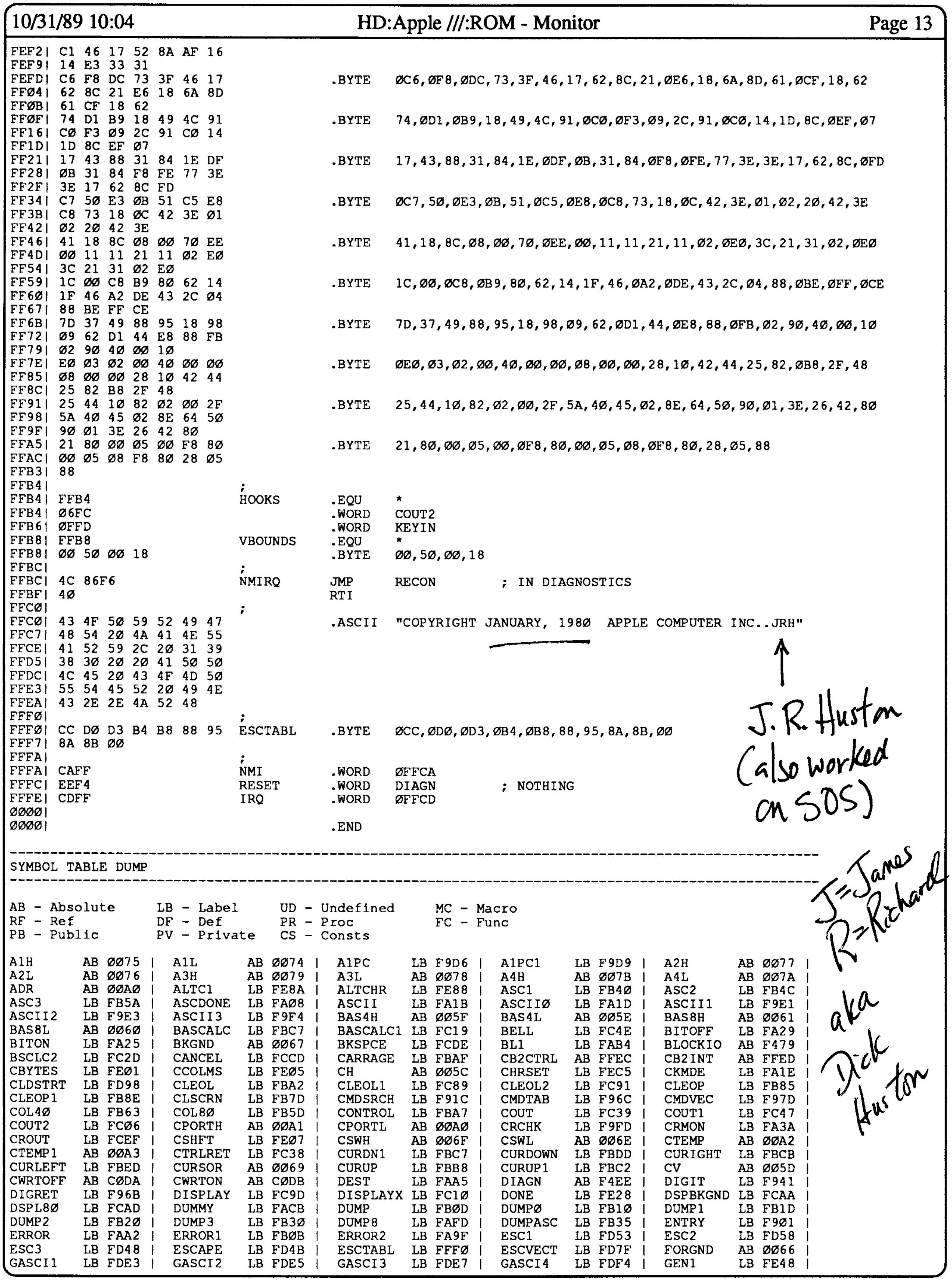


```
FDA5| 9D CAFF                          STA     ØFFCA,X
FDA8| BD B4FF                          LDA     HOOKS,X
FDAB| 95 6E                            STA     CSWL,X
FDAD| BD B8FF                          LDA     VBOUNDS,X
FDBØ| 95 58                            STA     LMARGIN,X
FDB2| CA                               DEX
FDB3| 1ØED                             BPL     SETUP1
FDB5| 85 82                            STA     IBDRVN
FDB7| A9 AØ                            LDA     #ØAØ         ; INPUT BUFFER AT $3AØ
FDB9| 85 7E                            STA     INBUF
FDBB| A9 6Ø                            LDA     #6Ø
FDBD| 85 81                            STA     IBSLOT
FDBF| A9 FF                            LDA     #ØFF
FDC1| 85 68                            STA     MODES
FDC3| 2Ø 63FB                          JSR     COL4Ø        ; SET 4Ø COLUMNS, CLEAR SCREEN
FDC6|                       ;
FDC6| ØØAØ                  ADR        .EQU    ØAØ
FDC6| ØØAØ                  CPORTL     .EQU    ADR
FDC6| ØØA1                  CPORTH     .EQU    ADR+1
FDC6| ØØA2                  CTEMP      .EQU    ADR+2
FDC6| ØØA3                  CTEMP1     .EQU    ADR+3
FDC6| ØØA4                  YTEMP      .EQU    ADR+4
FDC6| ØØCØ                  ROWTEMP    .EQU    ADR+2Ø
FDC6| CØDB                  CWRTON     .EQU    ØCØDB
FDC6| CØDA                  CWRTOFF    .EQU    ØCØDA
FDC6| FFEC                  CB2CTRL    .EQU    ØFFEC
FDC6| FFED                  CB2INT     .EQU    ØFFED
FDC6|                       ;
FDC6|                       ;
FDC6| A9 78                 GENENTR    LDA     #78          ; INIT SCREEN INDX LOCATIONS
FDC8| 85 AØ                            STA     CPORTL
FDCA| A9 Ø8                            LDA     #Ø8
FDCC| 85 A1                            STA     CPORTH
FDCE| A9 FØ                            LDA     #ØFØ         ; SET UP INDEX TO CHRSET
FDDØ| 85 A4                            STA     YTEMP
FDD2| A9 ØØ                            LDA     #ØØ
FDD4| AA                               TAX
FDD5| 95 CØ                 ZIPTEMPS   STA     ROWTEMP,X
FDD7| E8                               INX
FDD8| EØ 2Ø                            CPX     #2Ø
FDDA| DØF9                             BNE     ZIPTEMPS
FDDC| A9 Ø5                            LDA     #Ø5          ; FAKE THE FIRST BIT PATTERN
FDDE| 18                               CLC                  ; (PHANTOM 9TH BIT SHIFTED AS BIT Ø)
FDDF| Ø8                               PHP
FDEØ| 48                               PHA
FDE1| 86 A2                 GENASC     STX     CTEMP        ; GENERATE THE ASCII
FDE3| AØ Ø7                 GASCI1     LDY     #Ø7          ; CODES FOR THE FIRST PASS
FDE5| A6 A2                 GASCI2     LDX     CTEMP
FDE7| 8A                    GASCI3     TXA
FDE8| 91 AØ                            STA     (CPORTL),Y   ; $XXF=CHR Ø / 4
FDEA| E8                               INX                  ; $XXE=CHR 1 / 5
FDEB| 88                               DEY                  ; $XXD=CHR 2 / 6
FDEC| 3ØØ6                             BMI     GASCI4       ; $XXC=CHR 3 / 7
FDEE| CØ Ø3                            CPY     #Ø3          ; $XXB=CHR Ø / 4
FDFØ| DØF5                             BNE     GASCI3       ; $XXA=CHR 1 / 5
FDF2| FØF1                             BEQ     GASCI2       ; $XX9=CHR 2 / 6
FDF4| 2Ø 99FE               GASCI4     JSR     NXTPORT      ; $XX8=CHR 3 / 7
FDF7| BØØ8                             BCS     CBYTES       ; GO DECODE CHARACTER TABLE
FDF9| C9 ØA                            CMP     #ØA          ; SECOND SET OF 4?
FDFB| DØE6                             BNE     GASCI1
FDFD| A2 24                            LDX     #24
FDFF| DØEØ                             BNE     GENASC       ; BRANCH ALWAYS
FEØ1| 68                    CBYTES     PLA                  ; RESTORE BIT PATTERN
FEØ2| 28                               PLP
FEØ3| A2 17                            LDX     #17          ; (4 CHARACTERS OF 6 ROWS)
FEØ5| AØ Ø5                 CCOLMS     LDY     #Ø5          ; (FIVE COLUMNS)
FEØ7| 36 C4                 CSHFT      ROL     ROWTEMP+4,X  ; BREAK BYTE INTO
FEØ9| ØA                               ASL     A            ; 5 BIT GROUPS
FEØA| DØØE                             BNE     SHFTCNT      ; BRANCH IF MORE BITS IN THIS BYTE
FEØC| 84 A2                            STY     CTEMP
FEØE| C6 A4                            DEC     YTEMP        ; (NOTE. CARRY IS SET)
FE1Ø| FØ16                             BEQ     DONE         ; BRANCH IF ALL DONE
FE12| A4 A4                            LDY     YTEMP        ; GET CHARACTER TABLE INDEX
FE14| B9 C4FE                          LDA     CHRSET-1,Y
FE17| 2A                               ROL     A            ; (CARRY KEEPS BYTE NON-ZERO UNTIL ALL 8 ARE
FE18|                                                       ;  ARE SHIFTED)
FE18| A4 A2                            LDY     CTEMP        ; RESTORE COLUMN COUNT
FE1A| 88                    SHFTCNT    DEY                  ; GOT ALL FIVE BITS?
FE1B| DØEA                             BNE     CSHFT        ; NO, DO NEXT
FE1D| CA                               DEX                  ; ALL ROWS DONE
FE1E| 1ØE5                             BPL     CCOLMS       ; NO, DO NEXT
FE2Ø| Ø8                               PHP                  ; SAVE REMAINING BIT PATTERN AND CARRY
FE21| 48                               PHA
FE22| 2Ø 28FE                          JSR     STORCHRS     ; MOVE EM TO NON DISPLAYED VIDEO AREA
FE25| 4C Ø1FE                          JMP     CBYTES
FE28|                       ;
FE28| FE28                  DONE       .EQU    *
FE28|                       ;
FE28| A2 1F                 STORCHRS   LDX     #1F          ; MOVE CHARACTER PATTERNS TO VIDEO AREA
FE2A| AØ ØØ                 STORSET    LDY     #ØØ
```

“APPLE_PAT_4_383_296_C_37” 189 KB 2000-02-28 dpi: 300h x 300v pix: 2312h x 3097v



```
FE2C| B5 CØ                 STOROW     LDA     ROWTEMP,X
FE2E| ØA                               ASL     A            ; SHIFT TO CENTER
FE2F| 29 3E                            AND     #3E          ; STRIP EXTRA GARBAGE
FE31| 91 AØ                            STA     (CPORTL),Y
FE33| CA                               DEX
FE34| C8                               INY
FE35| CØ Ø8                            CPY     #Ø8          ; THIS GROUP DONE
FE37| DØF3                             BNE     STOROW       ; NO, NEXT ROW
FE39| 2Ø 99FE                          JSR     NXTPORT
FE3C| C9 Ø8                            CMP     #Ø8
FE3E| FØØ4                             BEQ     GENDONE      ; ALL ROWS STORED?
FE4Ø| 8A                               TXA
FE41| 1ØE7                             BPL     STORSET
FE43| 6Ø                               RTS                  ; PARTIAL SET ($478-$5FF)
FE44|                       ;
FE44| A9 Ø1                 GENDONE    LDA     #Ø1          ; SET NORMAL MODE
FE46| 85 A2                            STA     CTEMP
FE48| A9 6Ø                 GEN1       LDA     #6Ø          ; PREPARE TO SEND BYTES TO CHARACTER
FE4A| 2C DBCØ                          BIT     CWRTON       ; GENERATOR RAM
FE4D| 2Ø AEFE                          JSR     VRETRCE      ; WAIT FOR NEXT VERTICAL RETRACE
FE5Ø| A9 2Ø                            LDA     #2Ø          ; WAIT AGAIN
FE52| 2Ø AEFE                          JSR     VRETRCE
FE55| 2C DACØ                          BIT     CWRTOFF      ; CHARACTERS ARE NOW LOADED
FE58| 2Ø 88FE                          JSR     ALTCHR       ; REPEAT THIS SET FOR OTHER 64 CHARACTERS
FE5B| C6 A2                            DEC     CTEMP        ; HAVE WE DONE ALTERNATES YET?
FE5D| 1Ø16                             BPL     GEN2         ; NO, DO IT!
FE5F| A9 Ø8                            LDA     #Ø8          ; BUMP ASCII VALUES FOR NEXT SET
FE61| 85 A1                            STA     CPORTH
FE63| AØ Ø7                 NXTASCI    LDY     #Ø7          ; THE USUAL COUNTDOWN
FE65| B1 AØ                 NXTASC2    LDA     (CPORTL),Y
FE67| 18                               CLC
FE68| 69 Ø8                            ADC     #Ø8
FE6A| 91 AØ                            STA     (CPORTL),Y
FE6C| 88                               DEY
FE6D| 1ØF6                             BPL     NXTASC2
FE6F| 2Ø 99FE                          JSR     NXTPORT
FE72| 9ØEF                             BCC     NXTASCI
FE74| 6Ø                               RTS
FE75| AØ Ø3                 GEN2       LDY     #Ø3          ; SETUP ALTERNATE WITH UNDERLINES
FE77| A9 7F                            LDA     #7F
FE79| 99 FCØ5               UNDER      STA     Ø5FC,Y
FE7C| 99 FCØ7                          STA     Ø7FC,Y
FE7F| 88                               DEY
FE8Ø| 1ØF7                             BPL     UNDER
FE82| A9 Ø8                            LDA     #Ø8
FE84| 85 A1                            STA     CPORTH
FE86| DØCØ                             BNE     GEN1
FE88|                       ;
FE88| AØ Ø7                 ALTCHR     LDY     #Ø7          ; ADJUST ASCII FOR ALTERNATE SET
FE8A| B1 AØ                 ALTC1      LDA     (CPORTL),Y
FE8C| 49 2Ø                            EOR     #2Ø          ; $2Ø-->  $4Ø-->$6Ø
FE8E| 91 AØ                            STA     (CPORTL),Y
FE9Ø| 88                               DEY
FE91| 1ØF7                             BPL     ALTC1        ; ADJUST THEM ALL
FE93| 2Ø 99FE                          JSR     NXTPORT
FE96| 9ØFØ                             BCC     ALTCHR
FE98| 6Ø                               RTS
FE99|                       ;
FE99| A5 AØ                 NXTPORT    LDA     CPORTL       ; CONVERT $78->$F8 OR $F8-$78
FE9B| 49 8Ø                            EOR     #8Ø
FE9D| 85 AØ                            STA     CPORTL
FE9F| 3ØØ2                             BMI     NOHIGH
FEA1| E6 A1                            INC     CPORTH
FEA3| A5 A1                 NOHIGH     LDA     CPORTH
FEA5| C9 ØC                            CMP     #ØC
FEA7| DØØ4                             BNE     PORTDN
FEA9| A9 Ø4                            LDA     #Ø4
FEAB| 85 A1                            STA     CPORTH
FEAD| 6Ø                    PORTDN     RTS
FEAE|                       ;
FEAE| 85 A3                 VRETRCE    STA     CTEMP1       ; SAVE BITS TO BE STORED
FEBØ| AD ECFF                          LDA     CB2CTRL      ; CONTROL PORT FOR 'CB2'
FEB3| 29 3F                            AND     #3F          ; RESET HI BITS TO Ø
FEB5| Ø5 A3                            ORA     CTEMP1
FEB7| 8D ECFF                          STA     CB2CTRL
FEBA| A9 Ø8                            LDA     #Ø8          ; TEST VERTICAL RETRACE
FEBC| 8D EDFF                          STA     CB2INT
FEBF| 2C EDFF               VWAIT      BIT     CB2INT       ; WAIT FOR RETRACE
FEC2| FØFB                             BEQ     VWAIT
FEC4| 6Ø                               RTS
FEC5|                       ;
FEC5| FEC5                  CHRSET     .EQU    *
FEC5|                       ;
FEC5| FØ Ø1 82 18 4Ø 84 81             .BYTE   ØFØ,Ø1,82,18,4Ø,84,81,2F,58,44,81,29,Ø2,1E,Ø1,91,7C,1F,49,3Ø
FECC| 2F 58 44 81 29 Ø2 1E
FED3| Ø1 91 7C 1F 49 3Ø
FED9| 8A Ø8 43 14 31 2A 22             .BYTE   8A,Ø8,43,14,31,2A,22,13,ØE3,ØF7,ØC4,91,48,ØA2,ØDA,24,ØC6,4A
FEEØ| 13 E3 F7 C4 91 48 A2
FEE7| DA 24 C6 4A
FEEB| 62 8C 24 C6 F8 63 8C             .BYTE   62,8C,24,ØC6,ØF8,63,8C,ØC1,46,17,52,8A,ØAF,16,14,ØE3,33,31
```



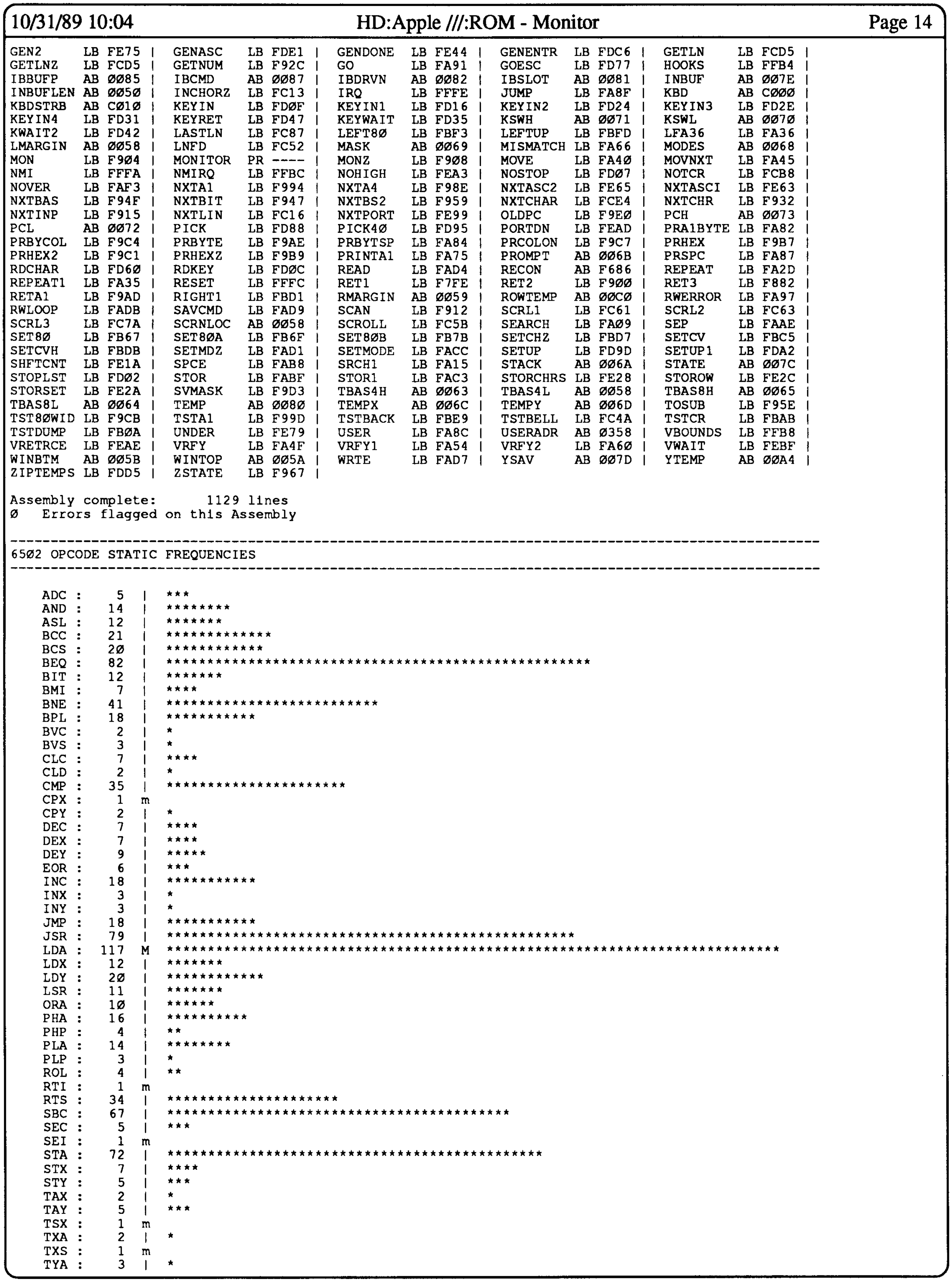
10/31/89 10:04 HD:Apple ///:ROM - Monitor Page 13

```
FEF2| C1 46 17 52 8A AF 16
FEF9| 14 E3 33 31
FEFD| C6 F8 DC 73 3F 46 17                 .BYTE   ØC6,ØF8,ØDC,73,3F,46,17,62,8C,21,ØE6,18,6A,8D,61,ØCF,18,62
FFØ4| 62 8C 21 E6 18 6A 8D
FFØB| 61 CF 18 62
FFØF| 74 D1 B9 18 49 4C 91                 .BYTE   74,ØD1,ØB9,18,49,4C,91,ØCØ,ØF3,Ø9,2C,91,ØCØ,14,1D,8C,ØEF,Ø7
FF16| CØ F3 Ø9 2C 91 CØ 14
FF1D| 1D 8C EF Ø7
FF21| 17 43 88 31 84 1E DF                 .BYTE   17,43,88,31,84,1E,ØDF,ØB,31,84,ØF8,ØFE,77,3E,3E,17,62,8C,ØFD
FF28| ØB 31 84 F8 FE 77 3E
FF2F| 3E 17 62 8C FD
FF34| C7 5Ø E3 ØB 51 C5 E8                 .BYTE   ØC7,5Ø,ØE3,ØB,51,ØC5,ØE8,ØC8,73,18,ØC,42,3E,Ø1,Ø2,2Ø,42,3E
FF3B| C8 73 18 ØC 42 3E Ø1
FF42| Ø2 2Ø 42 3E
FF46| 41 18 8C Ø8 ØØ 7Ø EE                 .BYTE   41,18,8C,Ø8,ØØ,7Ø,ØEE,ØØ,11,11,21,11,Ø2,ØEØ,3C,21,31,Ø2,ØEØ
FF4D| ØØ 11 11 21 11 Ø2 EØ
FF54| 3C 21 31 Ø2 EØ
FF59| 1C ØØ C8 B9 8Ø 62 14                 .BYTE   1C,ØØ,ØC8,ØB9,8Ø,62,14,1F,46,ØA2,ØDE,43,2C,Ø4,88,ØBE,ØFF,ØCE
FF6Ø| 1F 46 A2 DE 43 2C Ø4
FF67| 88 BE FF CE
FF6B| 7D 37 49 88 95 18 98                 .BYTE   7D,37,49,88,95,18,98,Ø9,62,ØD1,44,ØE8,88,ØFB,Ø2,9Ø,4Ø,ØØ,1Ø
FF72| Ø9 62 D1 44 E8 88 FB
FF79| Ø2 9Ø 4Ø ØØ 1Ø
FF7E| EØ Ø3 Ø2 ØØ 4Ø ØØ ØØ                 .BYTE   ØEØ,Ø3,Ø2,ØØ,4Ø,ØØ,ØØ,Ø8,ØØ,ØØ,28,1Ø,42,44,25,82,ØB8,2F,48
FF85| Ø8 ØØ ØØ 28 1Ø 42 44
FF8C| 25 82 B8 2F 48
FF91| 25 44 1Ø 82 Ø2 ØØ 2F                 .BYTE   25,44,1Ø,82,Ø2,ØØ,2F,5A,4Ø,45,Ø2,8E,64,5Ø,9Ø,Ø1,3E,26,42,8Ø
FF98| 5A 4Ø 45 Ø2 8E 64 5Ø
FF9F| 9Ø Ø1 3E 26 42 8Ø
FFA5| 21 8Ø ØØ Ø5 ØØ F8 8Ø                 .BYTE   21,8Ø,ØØ,Ø5,ØØ,ØF8,8Ø,ØØ,Ø5,Ø8,ØF8,8Ø,28,Ø5,88
FFAC| ØØ Ø5 Ø8 F8 8Ø 28 Ø5
FFB3| 88
FFB4|                          ;
FFB4| FFB4                     HOOKS       .EQU    *
FFB4| Ø6FC                                 .WORD   COUT2
FFB6| ØFFD                                 .WORD   KEYIN
FFB8| FFB8                     VBOUNDS     .EQU    *
FFB8| ØØ 5Ø ØØ 18                          .BYTE   ØØ,5Ø,ØØ,18
FFBC|                          ;
FFBC| 4C 86F6                  NMIRQ       JMP     RECON          ; IN DIAGNOSTICS
FFBF| 4Ø                                   RTI
FFCØ|                          ;
FFCØ| 43 4F 5Ø 59 52 49 47                 .ASCII  "COPYRIGHT JANUARY, 198Ø  APPLE COMPUTER INC..JRH"
FFC7| 48 54 2Ø 4A 41 4E 55
FFCE| 41 52 59 2C 2Ø 31 39
FFD5| 38 3Ø 2Ø 2Ø 41 5Ø 5Ø
FFDC| 4C 45 2Ø 43 4F 4D 5Ø
FFE3| 55 54 45 52 2Ø 49 4E
FFEA| 43 2E 2E 4A 52 48
FFFØ|                          ;
FFFØ| CC DØ D3 B4 B8 88 95     ESCTABL     .BYTE   ØCC,ØDØ,ØD3,ØB4,ØB8,88,95,8A,8B,ØØ
FFF7| 8A 8B ØØ
FFFA|                          ;
FFFA| CAFF                     NMI         .WORD   ØFFCA
FFFC| EEF4                     RESET       .WORD   DIAGN          ; NOTHING
FFFE| CDFF                     IRQ         .WORD   ØFFCD
ØØØØ|
ØØØØ|                                      .END

-------------------------------------------------------------------------------------------------
SYMBOL TABLE DUMP
-------------------------------------------------------------------------------------------------

AB - Absolute     LB - Label      UD - Undefined     MC - Macro
RF - Ref          DF - Def        PR - Proc          FC - Func
PB - Public       PV - Private    CS - Consts

A1H      AB ØØ75 |  A1L      AB ØØ74 |  A1PC      LB F9D6 |  A1PC1    LB F9D9 |  A2H      AB ØØ77 |
A2L      AB ØØ76 |  A3H      AB ØØ79 |  A3L       AB ØØ78 |  A4H      AB ØØ7B |  A4L      AB ØØ7A |
ADR      AB ØØAØ |  ALTC1    LB FE8A |  ALTCHR    LB FE88 |  ASC1     LB FB4Ø |  ASC2     LB FB4C |
ASC3     LB FB5A |  ASCDONE  LB FAØ8 |  ASCII     LB FA1B |  ASCIIØ   LB FA1D |  ASCII1   LB F9E1 |
ASCII2   LB F9E3 |  ASCII3   LB F9F4 |  BAS4H     AB ØØ5F |  BAS4L    AB ØØ5E |  BAS8H    AB ØØ61 |
BAS8L    AB ØØ6Ø |  BASCALC  LB FBC7 |  BASCALC1  LB FC19 |  BELL     LB FC4E |  BITOFF   LB FA29 |
BITON    LB FA25 |  BKGND    AB ØØ67 |  BKSPCE    LB FCDE |  BL1      LB FAB4 |  BLOCKIO  AB F479 |
BSCLC2   LB FC2D |  CANCEL   LB FCCD |  CARRAGE   LB FBAF |  CB2CTRL  AB FFEC |  CB2INT   AB FFED |
CBYTES   LB FEØ1 |  CCOLMS   LB FEØ5 |  CH        AB ØØ5C |  CHRSET   LB FEC5 |  CKMDE    LB FA1E |
CLDSTRT  LB FD98 |  CLEOL    LB FBA2 |  CLEOL1    LB FC89 |  CLEOL2   LB FC91 |  CLEOP    LB FB85 |
CLEOP1   LB FB8E |  CLSCRN   LB FB7D |  CMDSRCH   LB F91C |  CMDTAB   LB F96C |  CMDVEC   LB F97D |
COL4Ø    LB FB63 |  COL8Ø    LB FB5D |  CONTROL   LB FBA7 |  COUT     LB FC39 |  COUT1    LB FC47 |
COUT2    LB FCØ6 |  CPORTH   AB ØØA1 |  CPORTL    AB ØØAØ |  CRCHK    LB F9FD |  CRMON    LB FA3A |
CROUT    LB FCEF |  CSHFT    LB FEØ7 |  CSWH      AB ØØ6F |  CSWL     AB ØØ6E |  CTEMP    AB ØØA2 |
CTEMP1   AB ØØA3 |  CTRLRET  LB FC38 |  CURDN1    LB FBC7 |  CURDOWN  LB FBDD |  CURIGHT  LB FBCB |
CURLEFT  LB FBED |  CURSOR   AB ØØ69 |  CURUP     LB FBB8 |  CURUP1   LB FBC2 |  CV       AB ØØ5D |
CWRTOFF  AB CØDA |  CWRTON   AB CØDB |  DEST      LB FAA5 |  DIAGN    AB F4EE |  DIGIT    LB F941 |
DIGRET   LB F96B |  DISPLAY  LB FC9D |  DISPLAYX  LB FC1Ø |  DONE     LB FE28 |  DSPBKGND LB FCAA |
DSPL8Ø   LB FCAD |  DUMMY    LB FACB |  DUMP      LB FBØD |  DUMPØ    LB FB1Ø |  DUMP1    LB FB1D |
DUMP2    LB FB2Ø |  DUMP3    LB FB3Ø |  DUMP8     LB FAFD |  DUMPASC  LB FB35 |  ENTRY    LB F9Ø1 |
ERROR    LB FAA2 |  ERROR1   LB FBØB |  ERROR2    LB FA9F |  ESC1     LB FD53 |  ESC2     LB FD58 |
ESC3     LB FD48 |  ESCAPE   LB FD4B |  ESCTABL   LB FFFØ |  ESCVECT  LB FD7F |  FORGND   AB ØØ66 |
GASCI1   LB FDE3 |  GASCI2   LB FDE5 |  GASCI3    LB FDE7 |  GASCI4   LB FDF4 |  GEN1     LB FE48 |
```

J.R. Huston
(also worked
on SOS)

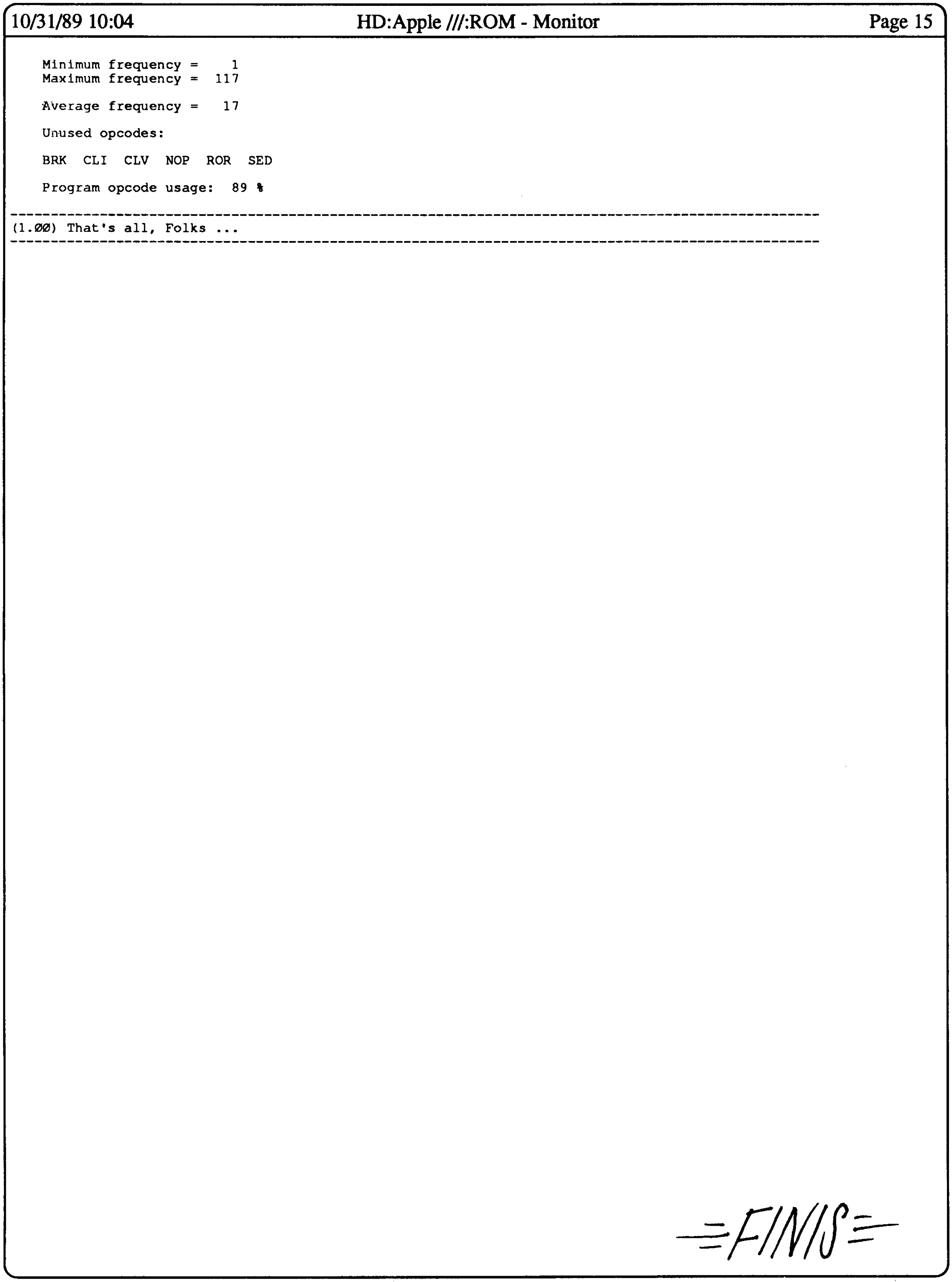
J=James
R=Richard

aka
Dick
Huston



10/31/89 10:04 HD:Apple ///:ROM - Monitor Page 14

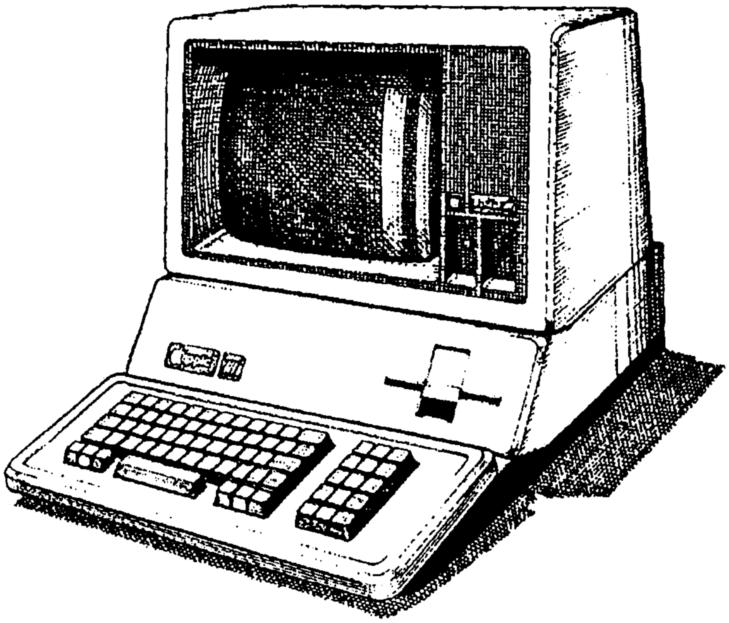
```
GEN2     LB FE75 | GENASC   LB FDE1 | GENDONE  LB FE44 | GENENTR  LB FDC6 | GETLN    LB FCD5 |
GETLNZ   LB FCD5 | GETNUM   LB F92C | GO       LB FA91 | GOESC    LB FD77 | HOOKS    LB FFB4 |
IBBUFP   AB ØØ85 | IBCMD    AB ØØ87 | IBDRVN   AB ØØ82 | IBSLOT   AB ØØ81 | INBUF    AB ØØ7E |
INBUFLEN AB ØØ5Ø | INCHORZ  LB FC13 | IRQ      LB FFFE | JUMP     LB FA8F | KBD      AB CØØØ |
KBDSTRB  AB CØ1Ø | KEYIN    LB FDØF | KEYIN1   LB FD16 | KEYIN2   LB FD24 | KEYIN3   LB FD2E |
KEYIN4   LB FD31 | KEYRET   LB FD47 | KEYWAIT  LB FD35 | KSWH     AB ØØ71 | KSWL     AB ØØ7Ø |
KWAIT2   LB FD42 | LASTLN   LB FC87 | LEFT8Ø   LB FBF3 | LEFTUP   LB FBFD | LFA36    LB FA36 |
LMARGIN  AB ØØ58 | LNFD     LB FC52 | MASK     AB ØØ69 | MISMATCH LB FA66 | MODES    AB ØØ68 |
MON      LB F9Ø4 | MONITOR  PR ---- | MONZ     LB F9Ø8 | MOVE     LB FA4Ø | MOVNXT   LB FA45 |
NMI      LB FFFA | NMIRQ    LB FFBC | NOHIGH   LB FEA3 | NOSTOP   LB FDØ7 | NOTCR    LB FCB8 |
NOVER    LB FAF3 | NXTA1    LB F994 | NXTA4    LB F98E | NXTASC2  LB FE65 | NXTASCI  LB FE63 |
NXTBAS   LB F94F | NXTBIT   LB F947 | NXTBS2   LB F959 | NXTCHAR  LB FCE4 | NXTCHR   LB F932 |
NXTINP   LB F915 | NXTLIN   LB FC16 | NXTPORT  LB FE99 | OLDPC    LB F9EØ | PCH      AB ØØ73 |
PCL      AB ØØ72 | PICK     LB FD88 | PICK4Ø   LB FD95 | PORTDN   LB FEAD | PRA1BYTE LB FA82 |
PRBYCOL  LB F9C4 | PRBYTE   LB F9AE | PRBYTSP  LB FA84 | PRCOLON  LB F9C7 | PRHEX    LB F9B7 |
PRHEX2   LB F9C1 | PRHEXZ   LB F9B9 | PRINTA1  LB FA75 | PROMPT   AB ØØ6B | PRSPC    LB FA87 |
RDCHAR   LB FD6Ø | RDKEY    LB FDØC | READ     LB FAD4 | RECON    AB F686 | REPEAT   LB FA2D |
REPEAT1  LB FA35 | RESET    LB FFFC | RET1     LB F7FE | RET2     LB F9ØØ | RET3     LB F882 |
RETA1    LB F9AD | RIGHT1   LB FBD1 | RMARGIN  AB ØØ59 | ROWTEMP  AB ØØCØ | RWERROR  LB FA97 |
RWLOOP   LB FADB | SAVCMD   LB FAD9 | SCAN     LB F912 | SCRL1    LB FC61 | SCRL2    LB FC63 |
SCRL3    LB FC7A | SCRNLOC  AB ØØ58 | SCROLL   LB FC5B | SEARCH   LB FAØ9 | SEP      LB FAAE |
SET8Ø    LB FB67 | SET8ØA   LB FB6F | SET8ØB   LB FB7B | SETCHZ   LB FBD7 | SETCV    LB FBC5 |
SETCVH   LB FBDB | SETMDZ   LB FAD1 | SETMODE  LB FACC | SETUP    LB FD9D | SETUP1   LB FDA2 |
SHFTCNT  LB FE1A | SPCE     LB FAB8 | SRCH1    LB FA15 | STACK    AB ØØ6A | STATE    AB ØØ7C |
STOPLST  LB FDØ2 | STOR     LB FABF | STOR1    LB FAC3 | STORCHRS LB FE28 | STOROW   LB FE2C |
STORSET  LB FE2A | SVMASK   LB F9D3 | TBAS4H   AB ØØ63 | TBAS4L   AB ØØ58 | TBAS8H   AB ØØ65 |
TBAS8L   AB ØØ64 | TEMP     AB ØØ8Ø | TEMPX    AB ØØ6C | TEMPY    AB ØØ6D | TOSUB    LB F95E |
TST8ØWID LB F9CB | TSTA1    LB F99D | TSTBACK  LB FBE9 | TSTBELL  LB FC4A | TSTCR    LB FBAB |
TSTDUMP  LB FBØA | UNDER    LB FE79 | USER     LB FA8C | USERADR  AB Ø358 | VBOUNDS  LB FFB8 |
VRETRCE  LB FEAE | VRFY     LB FA4F | VRFY1    LB FA54 | VRFY2    LB FA6Ø | VWAIT    LB FEBF |
WINBTM   AB ØØ5B | WINTOP   AB ØØ5A | WRTE     LB FAD7 | YSAV     AB ØØ7D | YTEMP    AB ØØA4 |
ZIPTEMPS LB FDD5 | ZSTATE   LB F967 |

Assembly complete:      1129 lines
Ø   Errors flagged on this Assembly

----------------------------------------------------------------------------------------------------
65Ø2 OPCODE STATIC FREQUENCIES
----------------------------------------------------------------------------------------------------

    ADC :    5  |  ***
    AND :   14  |  ********
    ASL :   12  |  *******
    BCC :   21  |  *************
    BCS :   2Ø  |  ************
    BEQ :   82  |  ****************************************************
    BIT :   12  |  *******
    BMI :    7  |  ****
    BNE :   41  |  **************************
    BPL :   18  |  ***********
    BVC :    2  |  *
    BVS :    3  |  *
    CLC :    7  |  ****
    CLD :    2  |  *
    CMP :   35  |  **********************
    CPX :    1  m
    CPY :    2  |  *
    DEC :    7  |  ****
    DEX :    7  |  ****
    DEY :    9  |  *****
    EOR :    6  |  ***
    INC :   18  |  ***********
    INX :    3  |  *
    INY :    3  |  *
    JMP :   18  |  ***********
    JSR :   79  |  **************************************************
    LDA :  117  M  ***************************************************************************
    LDX :   12  |  *******
    LDY :   2Ø  |  ************
    LSR :   11  |  *******
    ORA :   1Ø  |  ******
    PHA :   16  |  **********
    PHP :    4  |  **
    PLA :   14  |  ********
    PLP :    3  |  *
    ROL :    4  |  **
    RTI :    1  m
    RTS :   34  |  *********************
    SBC :   67  |  ******************************************
    SEC :    5  |  ***
    SEI :    1  m
    STA :   72  |  **********************************************
    STX :    7  |  ****
    STY :    5  |  ***
    TAX :    2  |  *
    TAY :    5  |  ***
    TSX :    1  m
    TXA :    2  |  *
    TXS :    1  m
    TYA :    3  |  *
```



```
    Minimum frequency =    1
    Maximum frequency =  117

    Average frequency =   17

    Unused opcodes:

    BRK  CLI  CLV  NOP  ROR  SED

    Program opcode usage:  89 %

----------------------------------------------------------------------------------------------------
(1.ØØ) That's all, Folks ...
----------------------------------------------------------------------------------------------------
```

=FINIS=

Apple III Computer Information



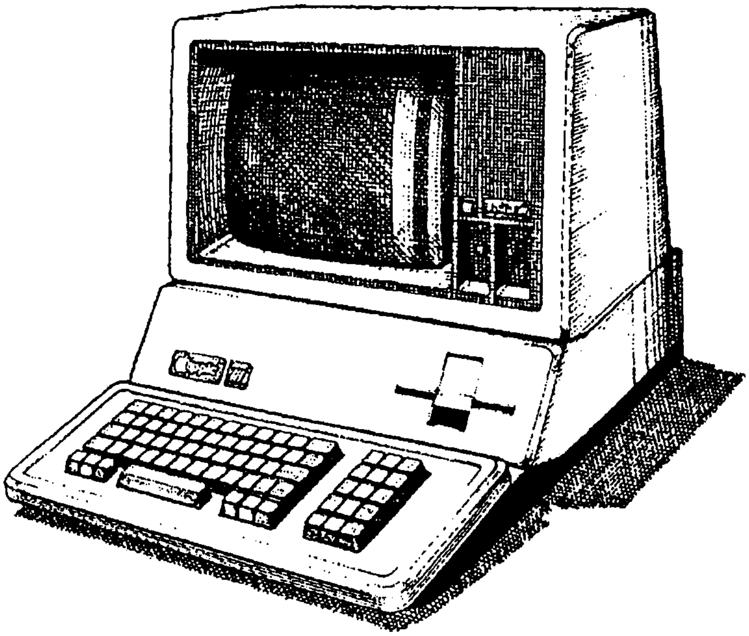
Inside the Apple III ROM

Document Table of Contents

Revision 2 • 04 Dec 1997
Revision 1 • 30 Nov 1997

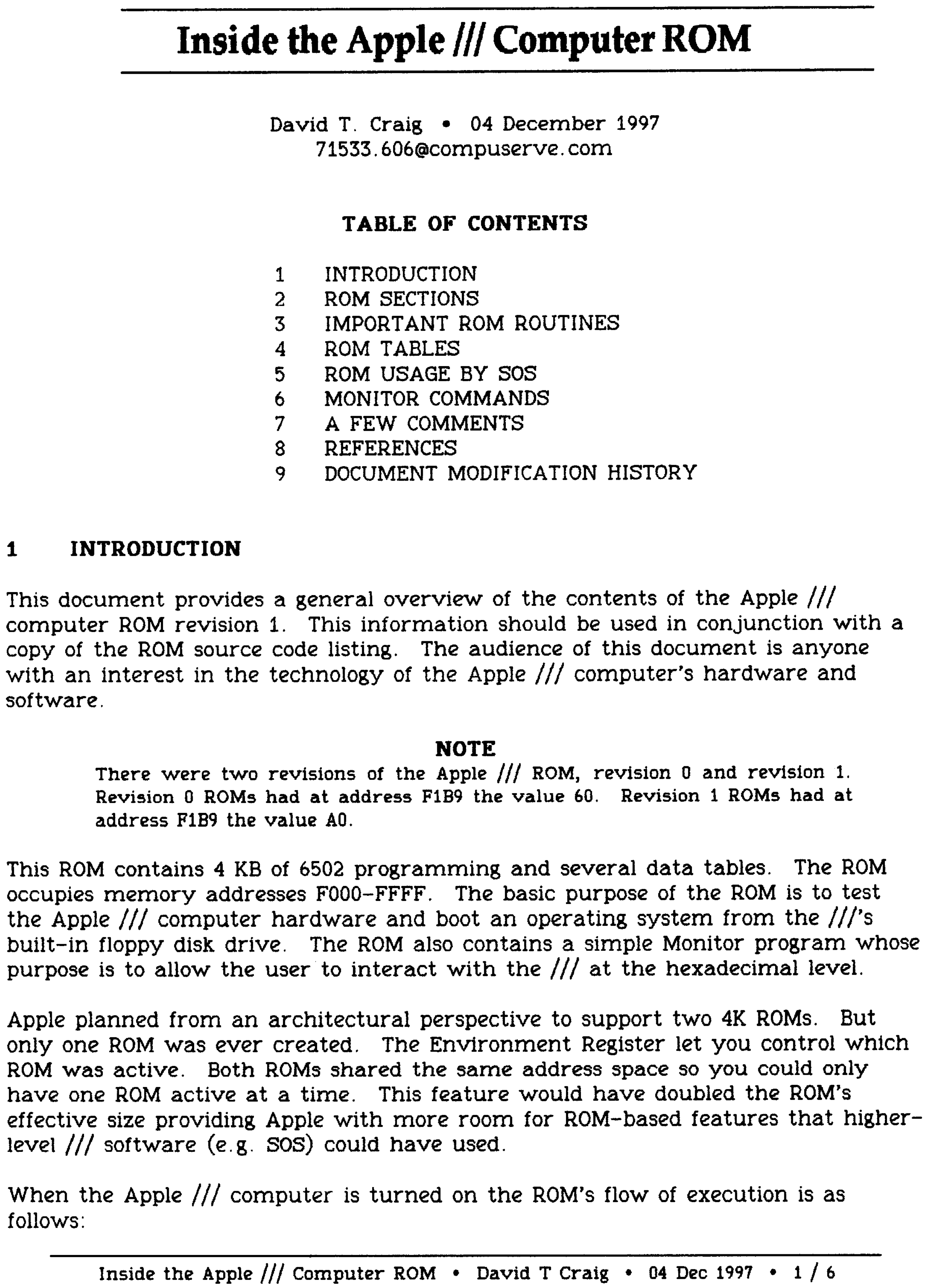
“APPLE_PAT_4_383_296_D_01” 125 KB 2000-02-28 dpi: 300h x 300v pix: 1941h x 2776v

Apple III Computer Information



Inside the Apple III ROM

Revision 2 • 04 Dec 1997



Inside the Apple /// Computer ROM

David T. Craig • 04 December 1997
71533.606@compuserve.com

TABLE OF CONTENTS

1 INTRODUCTION

This document provides a general overview of the contents of the Apple /// computer ROM revision 1. This information should be used in conjunction with a copy of the ROM source code listing. The audience of this document is anyone with an interest in the technology of the Apple /// computer's hardware and software.

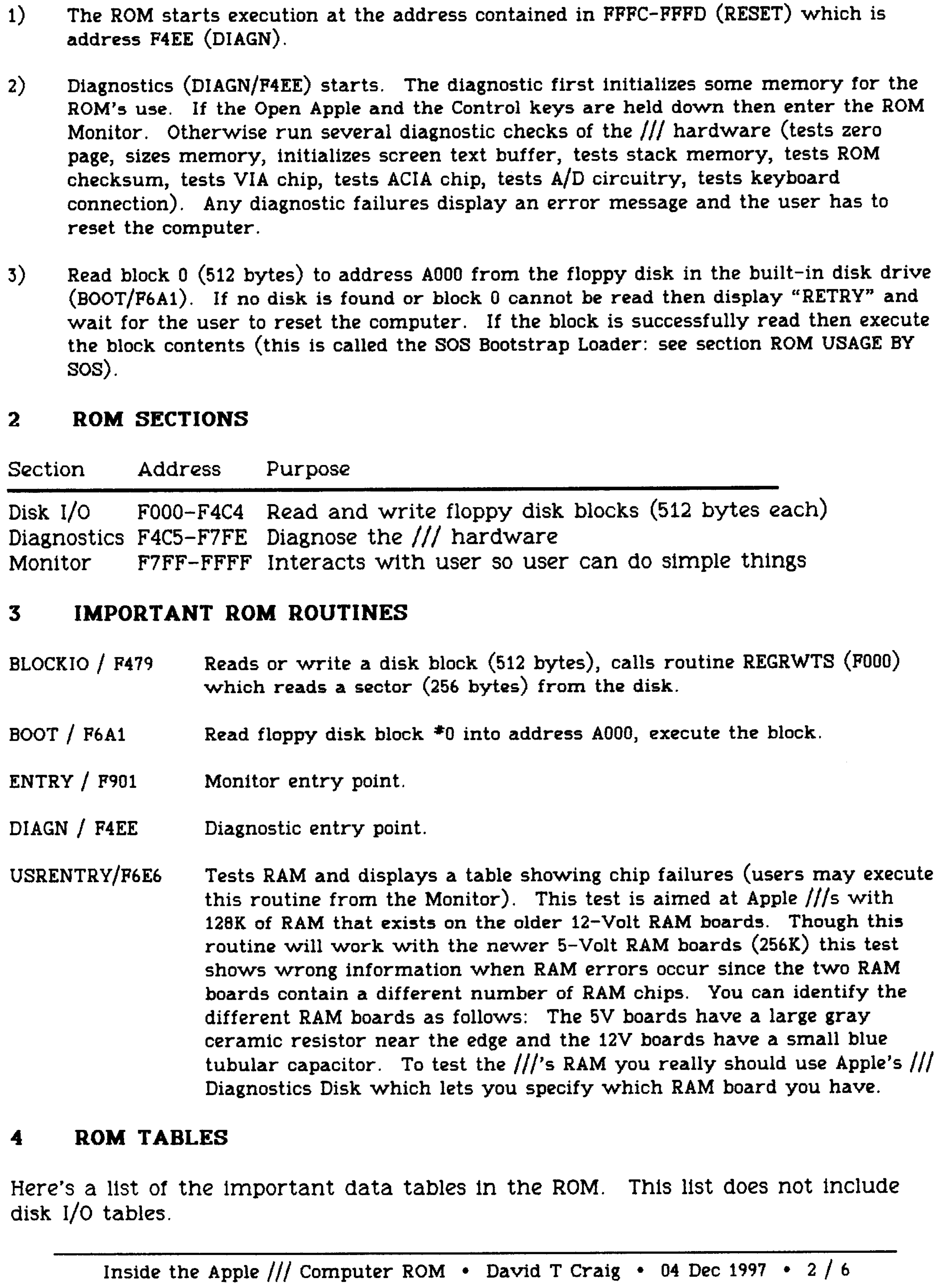
NOTE

There were two revisions of the Apple /// ROM, revision 0 and revision 1. Revision 0 ROMs had at address F1B9 the value 60. Revision 1 ROMs had at address F1B9 the value A0.

This ROM contains 4 KB of 6502 programming and several data tables. The ROM occupies memory addresses F000-FFFF. The basic purpose of the ROM is to test the Apple /// computer hardware and boot an operating system from the ///'s built-in floppy disk drive. The ROM also contains a simple Monitor program whose purpose is to allow the user to interact with the /// at the hexadecimal level.

Apple planned from an architectural perspective to support two 4K ROMs. But only one ROM was ever created. The Environment Register let you control which ROM was active. Both ROMs shared the same address space so you could only have one ROM active at a time. This feature would have doubled the ROM's effective size providing Apple with more room for ROM-based features that higher-level /// software (e.g. SOS) could have used.

When the Apple /// computer is turned on the ROM's flow of execution is as follows:



1) The ROM starts execution at the address contained in FFFC-FFFD (RESET) which is address F4EE (DIAGN).

2) Diagnostics (DIAGN/F4EE) starts. The diagnostic first initializes some memory for the ROM's use. If the Open Apple and the Control keys are held down then enter the ROM Monitor. Otherwise run several diagnostic checks of the /// hardware (tests zero page, sizes memory, initializes screen text buffer, tests stack memory, tests ROM checksum, tests VIA chip, tests ACIA chip, tests A/D circuitry, tests keyboard connection). Any diagnostic failures display an error message and the user has to reset the computer.

3) Read block 0 (512 bytes) to address A000 from the floppy disk in the built-in disk drive (BOOT/F6A1). If no disk is found or block 0 cannot be read then display "RETRY" and wait for the user to reset the computer. If the block is successfully read then execute the block contents (this is called the SOS Bootstrap Loader: see section ROM USAGE BY SOS).

2 ROM SECTIONS

Section	Address	Purpose
Disk I/O	F000-F4C4	Read and write floppy disk blocks (512 bytes each)
Diagnostics	F4C5-F7FE	Diagnose the /// hardware
Monitor	F7FF-FFFF	Interacts with user so user can do simple things

3 IMPORTANT ROM ROUTINES

BLOCKIO / F479 — Reads or write a disk block (512 bytes), calls routine REGRWTS (F000) which reads a sector (256 bytes) from the disk.

BOOT / F6A1 — Read floppy disk block *0 into address A000, execute the block.

ENTRY / F901 — Monitor entry point.

DIAGN / F4EE — Diagnostic entry point.

USRENTRY/F6E6 — Tests RAM and displays a table showing chip failures (users may execute this routine from the Monitor). This test is aimed at Apple ///s with 128K of RAM that exists on the older 12-Volt RAM boards. Though this routine will work with the newer 5-Volt RAM boards (256K) this test shows wrong information when RAM errors occur since the two RAM boards contain a different number of RAM chips. You can identify the different RAM boards as follows: The 5V boards have a large gray ceramic resistor near the edge and the 12V boards have a small blue tubular capacitor. To test the ///'s RAM you really should use Apple's /// Diagnostics Disk which lets you specify which RAM board you have.

4 ROM TABLES

Here's a list of the important data tables in the ROM. This list does not include disk I/O tables.

"APPLE_PAT_4_383_296_D_04" 219 KB 2000-02-28 dpi: 300h x 300v pix: 2213h x 3004v

Table Name / Address	Contents
CHRSET / FEC5-FFB3	Default character set (overridden when SOS loads the character set from SOS.DRIVER)
Copyright / FFC0-FFEF	Copyright message (contains the initials "JRH" for J. R. "Dick" Huston who was a key player behind the /// and SOS)
NMI / FFFA-FFFB	Jump address for the Non-Maskable Interrupt signal
RESET / FFFC-FFFD	Jump address when the /// is powered on
IRQ / FFFE-FFFF	Jump address for the Interrupt Request signal

5 ROM USAGE BY SOS

The Apple /// operating system (SOS = Sophisticated Operating System or Sara's OS) uses several ROM routines. These routines seem to all be related to disk block I/O. The following discussion is based on SOS version 1.3.

When the ROM loads block 0 from a SOS disk the ROM is loading the SOS Bootstrap Loader program. This program, which is at most 512 bytes in length, uses the ROM routine REGRWTS (F000) to read the SOS Loader into memory. This program does not test the ROM revision. It is interesting to note that ROM routine BLOCKIO is not used, instead a lower-level routine (REGRWTS) is used.

The SOS Loader determines if the ROM is revision 1 by comparing address F1B9's contents against A0 (reference: SOS source file SOSLDR.D.SRC). If this comparison fails then SOS displays on the screen the error "ROM ERROR: PLEASE NOTIFY YOUR DEALER." If the ROM revision is correct then the SOS loader uses the ROM's disk I/O routines to read more of SOS into memory.

The disk /// driver that is built into SOS also uses the ROM to perform disk block I/O (reference: DISK3.SRC). It is interesting to note that when the disk driver is initialized the driver checks if the ROM revision is 0 or 1. A revision of 0 is detected if address F1B9 contains 60. If neither revision is found then the disk driver returns an error to SOS (I don't think this will ever happen since the SOS loader has already determined that the ROM is revision 1). For a valid ROM revision the disk driver sets up several jump vectors which point to the appropriate addresses in the ROM for the various ROM routines needed by the disk driver. Therefore, the disk driver seems compatible with either ROM revision whereas the SOS loader likes only revision 1.

The .CONSOLE driver source listing appears to not use any ROM routines even though the ROM contains 40 and 80 column text routines and keyboard input routines. I assume the console driver was much more sophisticated than the ROM's text features and so using the ROM routines would not have worked well for this driver. I also assume that if the console driver used the ROM that when ROM revision 1 was built the console driver would have had to be changed and Apple (smartly) did not want to do this.

6 MONITOR COMMANDS

Holding down the Open Apple and Control keys when the /// starts or when you press the Reset key activates the /// ROM Monitor. The screen will display in the upper left corner a small right-facing arrow with a blinking underscore character as the cursor. The Monitor's commands are based on the Apple][‘s Monitor commands but some commands have changed slightly and others are new for the (newer) ///.

The Monitor supports the following commands:

addr1.addr2	Dump memory data to screen from address 1 to address 2 and display ASCII character at the right of the screen.
CARRIAGE RETURN	Dump next line of addresses to the screen.
SPACE	Pause current memory dump. Press again to continue.
addr:byte_list	Store starting at the address the list of bytes.
addr:'text'	Store text starting at address with high bit clear.
addr:"text"	Store text starting at address with high bit set.
addr3<addr1.addr2M	Move data in addresses 1-2 to address 3.
addr3<addr1.addr2V	Verify data in addresses 1-2 is the same as data starting at address 3.
byte<addr1.addr2S	Search memory in address range 1-2 for the byte.
block<addr1.addr2W	Write address range to disk starting at the disk block.
block<addr1.addr2R	Read disk starting at block to the address range.
addrG	Call subroutine at the address.
addrJ	Jump to the address.
U	Call user routine starting at address $03F8.
X	Repeat last command line until you press the SPACE BAR.
ESC-8	Display 80 columns of text.
ESC-4	Display 40 columns of text.
/	Seperate multiple commands on the same line.
CTRL-I	Interrupt current operation, return to Monitor command line.

Note: See Wells' *Apple /// Entry Points* article for a great overview of the ROM Monitor, its commands (with some syntax errors), and the memory locations that need setting up for the key ROM routines to work. Apple's */// Service Reference Manual* (p. 13.57) has a list of Monitor commands. Anderson's *The Apple Nobody Knows* also has good Monitor command info.

To obtain a binary dump of the /// ROM you can do the following:

1. Initialize a disk on either the /// or an Apple][computer.
2. Insert the new disk in the ///.
3. Start the /// and hold down the Open Apple and Control keys.
4. You should be in the /// Monitor.
5. Type 0<F000.FFFW to write the ROM to disk blocks 0 to 7
6. Use a disk block reader on the /// or the][to read the ROM blocks and save them to a real file.

This disk writing is needed since the ROM does not provide a command for redirecting screen output to the ///'s serial port. But, I've read that you can output the ROM contents to the ///'s serial port but this involves using the Monitor to write a small program. If anyone has such a program please send a copy my way.

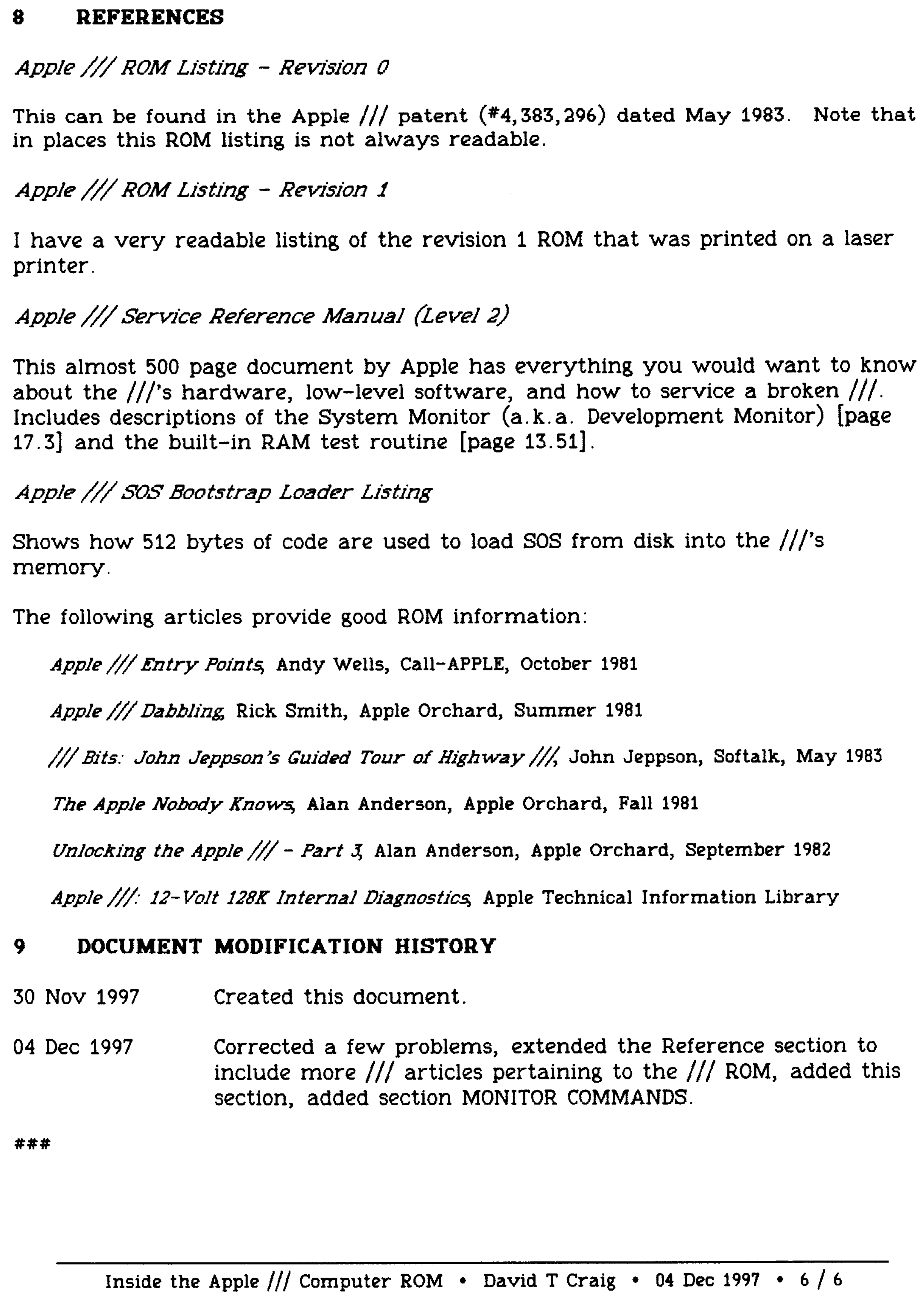
7 A FEW COMMENTS

I find it interesting, at least from a software engineering perspective, to note that in my opinion the /// ROM is missing several key features which I thought any system ROM would need. The ROM is missing two features which I think would have been useful to Apple and outside /// programmers:

1) The ROM does not have an explicit version number which exists at a specific ROM address. This version number could be used to validate the ROM in case there were several different ROMs (as there were). Apple uses a pseudo ROM version number (called the revision number) during the loading of SOS but this is somewhat lame in my opinion.

2) The ROM does not have a dispatch routine for use by the OS or applications that want to use ROM routines. This dispatch routine would reside at a specific address (e.g., F000) and it would take as input a command number and a set of parameters. These parameters could be passed via registers or on the stack. This routine would allow Apple to change the ROM and ROM "users" would not need to change their programming as long as they used the selector routine. The Apple][ROM did not have such a routine which caused Apple many headaches when it wanted to change the Apple][ROM and had to keep lots of routines in their same place.

3) The ROM source code is rather sparse concerning comments. It would be nice if the ROM contained detailed information about what each routine did and how to call the routines. Obviously, Apple did not expect anyone but Apple's own programmers to ever see the ROM source or use the ROM routines. (I've seen the Lisa computer's ROM listing which is much better documented than the ///'s and both are comparable in terms of age).



8 REFERENCES

Apple /// ROM Listing - Revision 0

This can be found in the Apple /// patent (#4,383,296) dated May 1983. Note that in places this ROM listing is not always readable.

Apple /// ROM Listing - Revision 1

I have a very readable listing of the revision 1 ROM that was printed on a laser printer.

Apple /// Service Reference Manual (Level 2)

This almost 500 page document by Apple has everything you would want to know about the ///'s hardware, low-level software, and how to service a broken ///. Includes descriptions of the System Monitor (a.k.a. Development Monitor) [page 17.3] and the built-in RAM test routine [page 13.51].

Apple /// SOS Bootstrap Loader Listing

Shows how 512 bytes of code are used to load SOS from disk into the ///'s memory.

The following articles provide good ROM information:

Apple /// Entry Points, Andy Wells, Call-APPLE, October 1981

Apple /// Dabbling, Rick Smith, Apple Orchard, Summer 1981

/// Bits: John Jeppson's Guided Tour of Highway ///, John Jeppson, Softalk, May 1983

The Apple Nobody Knows, Alan Anderson, Apple Orchard, Fall 1981

Unlocking the Apple /// - Part 3, Alan Anderson, Apple Orchard, September 1982

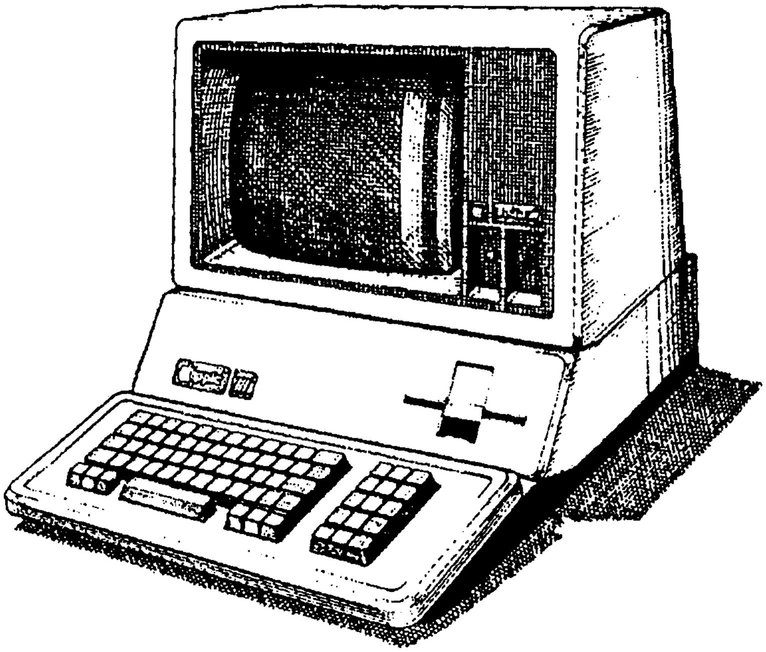
Apple ///: 12-Volt 128K Internal Diagnostics, Apple Technical Information Library

9 DOCUMENT MODIFICATION HISTORY

30 Nov 1997	Created this document.
04 Dec 1997	Corrected a few problems, extended the Reference section to include more /// articles pertaining to the /// ROM, added this section, added section MONITOR COMMANDS.

###

Apple III Computer Information



Inside the Apple III ROM

Revision 1 • 30 Nov 1997

"APPLE_PAT_4_383_296_D_09" 112 KB 2000-02-28 dpi: 300h x 300v pix: 2127h x 2546v

Inside the Apple /// Computer ROM

David T. Craig • 30 November 1997
71533.606@compuserve.com

TABLE OF CONTENTS

1 INTRODUCTION

This document provides a general overview of the contents of the Apple /// computer ROM revision 1. This information should be used in conjunction with a copy of the ROM source code listing. The audience of this document is anyone with an interest in the technology of the Apple /// computer's hardware and software.

> NOTE
>
> There were two revisions of the Apple /// ROM, revision 0 and revision 1. Revision 0 ROMs had at address F1B9 the value 60. Revision 1 ROMs had at address F1B9 the value A0.

This ROM contains 4 KB of 6502 programming and several data tables. The ROM occupies memory addresses F000-FFFF. The basic purpose of the ROM is to test the Apple /// computer hardware and boot an operating system from the ///'s built-in floppy disk drive. The ROM also contains a simple Monitor program whose purpose is to allow the user to interact with the /// at the hexadecimal level.

When the Apple /// computer is turned on the ROM's flow of execution is as follows:

1) The ROM starts execution at the address contained in FFFC-FFFD (RESET) which is address F4EE (DIAGN).

2) Diagnostics (DIAGN/F4EE) starts. The diagnostic first initializes some memory for the ROM's use. If the Open Apple key is held down then enter the ROM Monitor. Otherwise run several diagnostic checks of the /// hardware (tests zero page, sizes memory, initializes screen text buffer,

tests stack memory, tests ROM checksum, tests VIA chip, tests ACIA chip, tests A/D circuitry, tests keyboard connection). Any diagnostic failures display an error message and the user has to reset the computer.

3) Read block 0 (512 bytes) to address A000 from the floppy disk in the built-in disk drive (BOOT/F6A1). If no disk is found or block 0 cannot be read then display "RETRY" and wait for the user to reset the computer. If the block is successfully read then execute the block contents (this is called the SOS Bootstrap Loader: see section ROM USAGE BY SOS).

2 ROM SECTIONS

Section	Address	Purpose
Disk I/O	F000-F4C4	Read and write floppy disk blocks (512 bytes each)
Diagnostics	F4C5-F7FE	Diagnose the /// hardware
Monitor	F7FF-FFFF	Interacts with user so user can do simple things

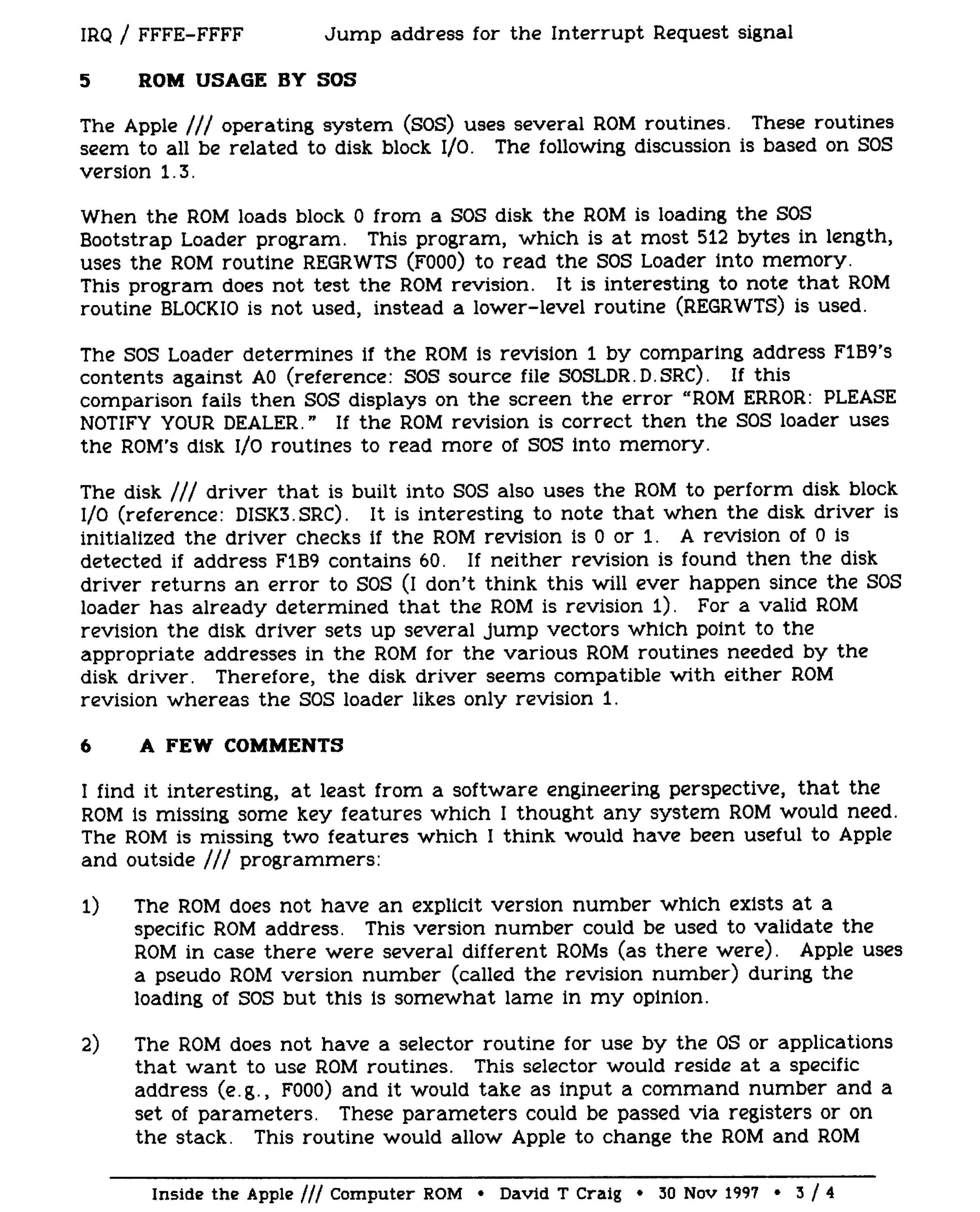
3 IMPORTANT ROM ROUTINES

BLOCKIO / F479	Reads or write a disk block (512 bytes), calls routine REGRWTS (F000) which reads a sector (256 bytes) from the disk
BOOT / F6A1	Read floppy disk block #0 into address A000, execute the block
ENTRY / F901	Monitor entry point
DIAGN / F4EE	Diagnostic entry point
USRENTRY/F6E6	Tests RAM and displays a table showing chip failures (users may execute this routine from the Monitor)

4 ROM TABLES

Here's a list of the important data tables in the ROM. This list does not include disk I/O tables.

Table Name / Address	Contents
CHRSET / FEC5-FFB3	Default character set (overridden when SOS loads the character set from SOS.DRIVER)
Copyright / FFC0-FFEF	Copyright message (contains the initials "JRH" for J. R. Huston who was a key player behind the /// and SOS)
NMI / FFFA-FFFB	Jump address for the Non-Maskable Interrupt signal
RESET / FFFC-FFFD	Jump address when the /// is powered on



IRQ / FFFE-FFFF	Jump address for the Interrupt Request signal

5 ROM USAGE BY SOS

The Apple /// operating system (SOS) uses several ROM routines. These routines seem to all be related to disk block I/O. The following discussion is based on SOS version 1.3.

When the ROM loads block 0 from a SOS disk the ROM is loading the SOS Bootstrap Loader program. This program, which is at most 512 bytes in length, uses the ROM routine REGRWTS (F000) to read the SOS Loader into memory. This program does not test the ROM revision. It is interesting to note that ROM routine BLOCKIO is not used, instead a lower-level routine (REGRWTS) is used.

The SOS Loader determines if the ROM is revision 1 by comparing address F1B9's contents against A0 (reference: SOS source file SOSLDR.D.SRC). If this comparison fails then SOS displays on the screen the error "ROM ERROR: PLEASE NOTIFY YOUR DEALER." If the ROM revision is correct then the SOS loader uses the ROM's disk I/O routines to read more of SOS into memory.

The disk /// driver that is built into SOS also uses the ROM to perform disk block I/O (reference: DISK3.SRC). It is interesting to note that when the disk driver is initialized the driver checks if the ROM revision is 0 or 1. A revision of 0 is detected if address F1B9 contains 60. If neither revision is found then the disk driver returns an error to SOS (I don't think this will ever happen since the SOS loader has already determined that the ROM is revision 1). For a valid ROM revision the disk driver sets up several jump vectors which point to the appropriate addresses in the ROM for the various ROM routines needed by the disk driver. Therefore, the disk driver seems compatible with either ROM revision whereas the SOS loader likes only revision 1.

6 A FEW COMMENTS

I find it interesting, at least from a software engineering perspective, that the ROM is missing some key features which I thought any system ROM would need. The ROM is missing two features which I think would have been useful to Apple and outside /// programmers:

1) The ROM does not have an explicit version number which exists at a specific ROM address. This version number could be used to validate the ROM in case there were several different ROMs (as there were). Apple uses a pseudo ROM version number (called the revision number) during the loading of SOS but this is somewhat lame in my opinion.

2) The ROM does not have a selector routine for use by the OS or applications that want to use ROM routines. This selector would reside at a specific address (e.g., F000) and it would take as input a command number and a set of parameters. These parameters could be passed via registers or on the stack. This routine would allow Apple to change the ROM and ROM

"APPLE_PAT_4_383_296_D_12" 294 KB 2000-02-28 dpi: 300h x 300v pix: 2182h x 3048v

"users" would not need to change their programming as long as they used the selector routine. The Apple][ROM did not have such a routine which caused Apple many headaches when it wanted to change the Apple][ROM and had to keep lots of routines in their same place.

7 REFERENCES

Apple /// ROM Listing

I have a very nice listing of revision 1 ROM. A listing (that is somewhat readable) for the earlier revision 0 ROM may be found in the Apple /// patent.

Apple /// Service Reference Manual (Level 2)

This almost 500 page book by Apple has everything you would want to know about the ///'s hardware, low-level software, and how to service a broken ///. Includes descriptions of the System Monitor (a.k.a. Development Monitor) [page 17.3] and the built-in RAM test routine [page 13.51].

Apple /// SOS Bootstrap Loader Listing

Shows how 512 bytes of code is used to load SOS from disk into the ///'s memory.

###

Apple /// Computer Technical Information

SOME COMMENTS ABOUT THE APPLE /// COMPUTER BOOT ROM

David T Craig -- 27 February 2004

BACKGROUND

The Apple /// computer was introduced by Apple Computer in 1980 and was discontinued in 1985.

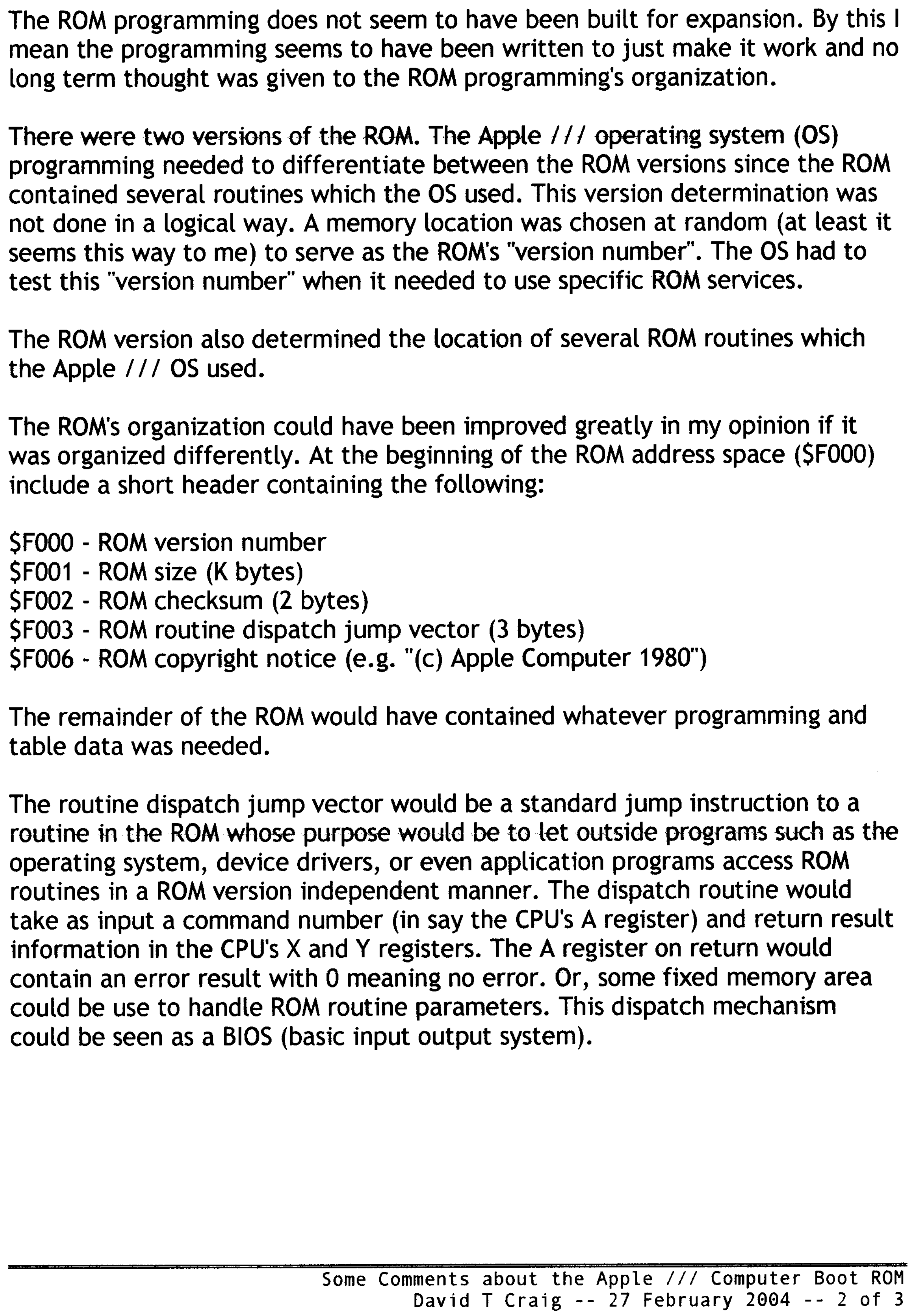
This computer was a microcomputer with orginally 128 KB of RAM memory expandable to 256 KB of RAM. It featured a 4 KB ROM (addressed from $F000 to $FFFF hexadecimal) which housed the initial programming that executed when the user turned on the computer. This ROM contained programming for the following functions:

+ diagnose hardware circuitry and memory
+ load and run a disk operating system (i.e. "boot")
+ provide an interface to a simple monitor program

The author wrote these comments after looking at the Apple /// ROM listing as found in Apple Computer's patent number 4,383,296 dated 10 May 1983. This analysis occured during a scanning of the Apple /// patent.

ROM COMMENTS

The Apple /// patent's ROM program listing is terrible in terms of printed quality. Many parts are very faint and impossible to read. I assume this was done on purpose by Apple's legal department so that Apple's competitors would not be able to duplicate this ROM programming easily.



The ROM programming does not seem to have been built for expansion. By this I mean the programming seems to have been written to just make it work and no long term thought was given to the ROM programming's organization.

There were two versions of the ROM. The Apple /// operating system (OS) programming needed to differentiate between the ROM versions since the ROM contained several routines which the OS used. This version determination was not done in a logical way. A memory location was chosen at random (at least it seems this way to me) to serve as the ROM's "version number". The OS had to test this "version number" when it needed to use specific ROM services.

The ROM version also determined the location of several ROM routines which the Apple /// OS used.

The ROM's organization could have been improved greatly in my opinion if it was organized differently. At the beginning of the ROM address space ($F000) include a short header containing the following:

$F000 - ROM version number
$F001 - ROM size (K bytes)
$F002 - ROM checksum (2 bytes)
$F003 - ROM routine dispatch jump vector (3 bytes)
$F006 - ROM copyright notice (e.g. "(c) Apple Computer 1980")

The remainder of the ROM would have contained whatever programming and table data was needed.

The routine dispatch jump vector would be a standard jump instruction to a routine in the ROM whose purpose would be to let outside programs such as the operating system, device drivers, or even application programs access ROM routines in a ROM version independent manner. The dispatch routine would take as input a command number (in say the CPU's A register) and return result information in the CPU's X and Y registers. The A register on return would contain an error result with 0 meaning no error. Or, some fixed memory area could be use to handle ROM routine parameters. This dispatch mechanism could be seen as a BIOS (basic input output system).

Possible dispatch routines could be:

- + Restart or Cold start or Warm start the computer
- + Read a block from a disk drive
- + Write a block to a disk drive
- + Return size in blocks of a disk drive
- + Checksum the ROM for diagnostic purposes
- + Test computer's RAM memory for diagnostic purposes
- + Enter the Apple /// Monitor program

This dispatch mechanism would have simplified the Apple /// OS use of the ROM services since the ROM would always be accessed from just one address ($F003). If the OS requested a ROM service which was unavailable (e.g. an old ROM was installed) then the ROM would tell the OS that the service did not exist via a dispatch error result.

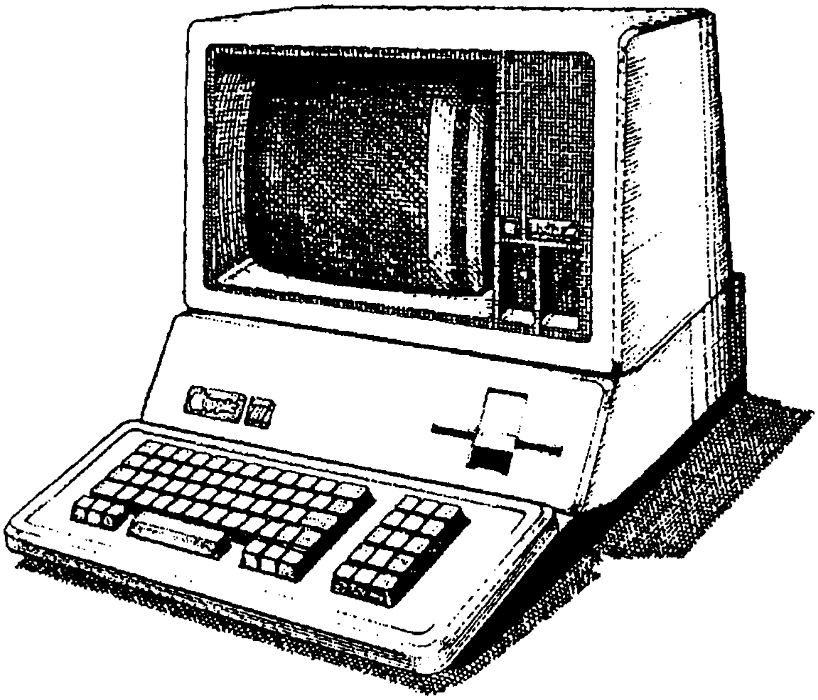
CONCLUSION

Hopefully this little commentary provides some useful information to its reader. If you are interested in the Apple /// computer you should see its patents (one is for the Apple ///, the other is for the Apple /// Plus). The first patent contains the full ROM listing, but the author has a real digital copy which is much more readable.

Enjoy.

###

Apple III Computer Information



Apple III Emulator Ideas

Version 4 • 12 Dec 1997

“APPLE_PAT_4_383_296_F_01” 106 KB 2000-02-28 dpi: 300h x 300v pix: 1923h x 2473v

SOME IDEAS ABOUT AN APPLE /// COMPUTER EMULATOR

David T. Craig -- 12 December 1997 -- Version 4

941 Calle Mejia #1006, Santa Fe, NM 87501 USA
e-mail: 71533.606@compuserve.com

TABLE OF CONTENTS

new: disk: READFILES
read all
files from
disk to
host computer
disk

Corrections — by page #

1 - Mod. History: add 2 spaces
9 - "C" char set bank → "K"
10 — _ : add extra space between
all bold command
line words.
13 - HGPP cmd name
10 - change RD cmd to show
(15) P and E bit names (uppercase
=1, lower=0)
12 - ZPAGE - show offset byte line,
same for adr1 - adr2
13 - S cmd

20 - SCROLL
14 - DISKBUFFER
15 - SS
16 - BPE more general
16-17 del some BP cmds
21 - BPNE
FONT FONTROM

MODIFICATIONHISTORY 2 sps

28 Nov 1997 -- Version 1
Created by David T. Craig.

04 Dec 1997 -- Version 2
New sections: MONITOR SUPPORT, EMULATOR DEBUGGING FACILITIES.
Updated sections: DISK IMAGES, MEMORY BANK SWITCHING EMULATOR, SOS SYSTEM CALL EMULATION, REFERENCES.
Added several good comments by Chris Smolinski (he's writing a /// emulator called SARA).

09 Dec 1997 -- Version 3
DISK IMAGES: Updated info about DTCMake3///DiskImage Mac application, made disk image file an all-text file.
SOS SYSTEM CALL EMULATION: typo Silentypr --> Silentype.
WHAT TARGET MACHINES SHOULD BE SUPPORTED: More pre-68040 Mac comments.
EMULATOR DEBUGGING FACILITIES: typo affects --> affect, added info about enabling/disabling SOS BRK disassembly, same for ProDOS, added list of emulator debugging commands.
EMULATOR MEMORY STRUCTURE: New section.

12 Dec 1997 -- Version 4
EMULATOR DEBUGGING FACILITIES: Added examples to every debugging command. Added commands SNAPSHOTW, SNAPSHOTR, ZPAGE, SPAGE, EPAGE, DRIVERS, macro commands.

1.0 PURPOSE

This document describes some ideas about implementing a software emulator for the Apple /// computer. These ideas are based on my experiences with the Apple /// computer and its software programming. No specific target machine is mentioned in this document since these ideas should be non-target machine specific. These ideas are submitted to stimulate thought about such an emulator and hopefully inspire someone to produce a working Apple /// emulator.

The technical details behind the Apple /// computer, its operating system (SOS), and /// programs (e.g. AppleWriter ///) are based on my extensive collection of /// technical manuals, specification sheets, and many /// technical articles (Dr. John Jeppson's articles are very exhaustive and full of lots of neat /// techoid stuff). I have around 15 Apple manuals, the majority of which were published by Apple, which include user manuals and the technical programming manuals.

For those people seriously interested in implementing an Apple /// emulator program I highly recommend that they have at least the Apple /// Service Reference Manual. This manual, which is almost 500 pages long, is the definitive reference for how the Apple /// computer works. Most of its contents describe theory of operation even though its title suggests service-type information only. The important features of this manual for a /// emulator writer are the /// memory map and the /// memory mapped I/O locations.

I also own an Apple /// computer which still today works very well. I programmed the /// many moons ago and have worked professionally as an Apple Macintosh computer programmer since 1984.

Note: All comments are welcome. If you have anything to add or correct please let me know and I will update the master copy of this document.

2.0 EMULATOR GOALS

The /// emulator should provide a complete emulation environment for the faithful execution of Apple /// and /// Plus programs. As far as the emulator user is concerned when they run the emulator program their computer should work just like an Apple /// computer and all /// visual fidelity should be maintained. Emulation of the Apple /// Plus computer may also be supported (this means the /// Plus' interlaced screen). If the /// Plus is supported by the emulator you may want to let the user specify if they want to run a /// or a /// Plus.

I think it would be beyond neat if the emulator could run Apple's running horses demo and the other /// demos.

The /// emulator should support an Apple /// computer with at least 256K of memory and four floppy 140K disks (.D1, .D2, .D3, .D4). Support for 512K of memory may also exist since the ///'s operating system (SOS) supports up to 512K of memory. Memory size, if variable, should always be a multiple of 32K. I believe the lowest memory size supported by the /// (ROM?) is 96K. Support for a ProFile disk may also exist (for this disk there would need to be a disk image with a size of 5M). The first floppy disk (.D1) would correspond to the floppy disk drive that is built into the Apple ///. The other disks correspond to external disks and should exist as image files with specific file names (e.g. "Apple 3 D1", "Apple 3 D2", etc). The ProFile disk image file should also have a specific file name (e.g. "Apple 3 ProFile").

Image file names should have an extension (e.g. ".D3I") since this is needed by PCs.

3.0 EMULATOR USER INTERFACE

When the user runs the Apple /// emulator program the user should see on their computer screen a screen (or a window representing the screen on GUI systems) corresponding to the ///'s screen which the user would see if they were in front of a real Apple /// computer. All /// text and graphic modes should be supported by the

/// emulator (this includes the special modes supported by the /// Plus and its interlaced screen architecture).

I recommend that the emulator also support a screen dump facility that writes the current /// screen to either a text file (for text modes) or to a graphic file (for graphic modes) or always just creates a graphic file. The screen dump graphic file should be a standard graphic file for whatever target machine your support (e.g. on the IBM PC running Windows produce .BMP files, on the Apple Macintosh produce PICT files). Since the /// supports custom character sets dumping the screen to a PICT file (or to the target computer's clipboard) may be the best solution.

The emulator screen if implemented in a GUI window may also display a status area at the bottom of the window. This status area would display at least two lines of text and would keep the user informed of what the emulator was doing internally.

4.0 DISK IMAGES

The /// emulator should read disk image files which correspond directly to real /// 140K disks. When the /// emulator starts it should look in its folder and if there exists a /// disk image file the emulator should boot this image. If there are multiple disk image files then the emulator may want to display a list of these images and have the user select an image to boot.

The disk images should be exact copies of real /// disks. To make copies of these disks there should exist an utility program that runs on the /// computer and which outputs disk block data to the /// serial port (I plan to make this utility and call it DTCDumpIt). This utility's output should be a hex/ascii dump that specifies block numbers and has a checksum for each line of data. This utility should ask the user if it should dump a file or a disk.

On the target machine there should exist a similar utility that inputs the disk block data and creates a disk image file. I recommend that the transmitted disk block data consist of a hex dump with block number and checksum information in a human readable fashion. The receiving program (on the target computer) would read this human readable information, verify that the data was sent correctly, and produce binary disk image file images (I plan to create this utility for the Apple Macintosh and call it DTCMake///DiskImage).

There should also exist a disk image file for the ///'s Boot ROM (recommended file name: "Apple 3 Boot ROM"). This image should contain the 4K ROM image. This ROM should be the Revision 1 ROM (not Revision 0) since this was the last ROM produced and SOS 1.3 (the last SOS) requires this ROM.

Users should also be able to format a disk image by specifying the disk drive device name (e.g. .D2). Users should then be able to name the disk image so that they can use it later. Users should be able to assign specific disk images to specific disk drives.

I recommend that all disk image files have a very specific internal format. This format should support the verification of disk image files so that if a disk image file becomes corrupted in some fashion the /// emulator can detect this corruption, not use the image, and alert the user.

Note: Support for existing Apple][disk image files may be feasible but I recommend against this since the format of these images could change.

The proposed image format:

The disk image file contains two parts, a header part and a data part. The header part appears first followed by the data part. The header part contains identification and verification information. The data part contains the actual disk blocks for the /// disk. This file contains only text, no binary data appears here in any fashion. The only non-text information that can appear in these files is the Carriage Return (CR) and the Line Feed (LF) characters. The emulator should ignore

LFs if appropriate. All information appears in lines with a maximum length of 255 characters. Character case is immaterial. Blank lines are ignored. The reason for this format is so these image files can be transferred over the internet without the need for any binary-to-text conversion. Also, text-only files can easily be viewed by people using a word processor.

The header part contains:

Line	Comments
Signature	"APPLE /// DISK IMAGE"
Version	"VERSION" version number (e.g. "1")
Image Name	"IMAGE NAME" name of image, anything the user wants, most likely the name of the interpreter on the disk, e.g. "Apple Writer ///"
Creation Date	"CREATED" date image file created, "YYYY-MM-DD"
Created by Name	"CREATED BY" name of person or company who created this image
Comment	"COMMENT" comment for anything user wants
Data Size	"DATA SIZE" size of data part (decimal, e.g. "143360")
Data Checksum	"DATA CHECKSUM" hexadecimal checksum (e.g. "FA7C3188")
Reserved 1	"RESERVED"
Reserved 2	"RESERVED"
Reserved 3	"RESERVED"
Reserved 4	"RESERVED"
Tech Comment	"TECH COMMENT" name of program that this is for
Header Checksum	"HEADER CHECKSUM" hexadecimal checksum (e.g. "B97C31D5")

Notes:

The checksum should be calculated as the exclusive-OR of each byte followed by a left rotation of 1 bit. Checksum starts with zero. Checksums should always be 4 bytes in size and be stored in the header as an 8 character string.

The Tech Comment's purpose is to allow people who obtain an image file to be able to contact someone about the file's purpose.

The data part contains lines representing 16 bytes from the original disk. Each line has a specific format which begins with the starting disk address for the line, 16 bytes, the ASCII equivalent of the 16 bytes, and a checksum for the bytes of the line with the format:

```
[00000000] 0123 4567 89ab cdef 0123 4567 89ab cdef [1234567890123456] 12345678
```

The last line of the file must be the word "FINIS".

Sample disk image file:

```
  APPLE /// DISK IMAGE
  VERSION 1
  IMAGE NAME Apple Writer ///
  CREATED 1997-10-11
  CREATED BY David T. Craig
  COMMENT Thanks to Paul Lutus
  DATA SIZE 16
  DATA CHECKSUM FA7C3188
  RESERVED
  RESERVED
  RESERVED
  RESERVED
  TECH COMMENT For David Craig's /// Emulator - 71533.606@compuserve.com
  HEADER CHECKSUM B97C31D5

  [00000000] 0123 4567 89ab cdef 0123 4567 89ab cdef [Apple.///.Emul..] FA7C3188
```

FINIS

5.0 6502 CPU EMULATION

The heart of the /// emulator should be the emulation of the 6502 CPU. The heart may be referred to as the "6502 engine." The emulator should support all of the 6502 instructions, the 6502 registers, and the special Apple /// registers (e.g. the bank switch register, the environment register, and the zero-page register). Special register descriptions and usage can be found in the Apple /// SOS Reference Manual.

The 6502 engine must be smart about accessing memory and use the bank switch and environment registers correctly.

If this level of the /// emulation is complete and robust the rest of the /// emulator should work much more easily.

Support for special /// features may also exist at this level of the /// emulator. For example, the /// emulator may not want to emulate all of the ///'s memory-mapped I/O features, but instead intercept access to special areas or routines and call the target machine's operating system to handle these features. See sections ROM EMULATION and MEMORY-MAPPED I/O EMULATION for more details.

6.0 ROM EMULATION

The /// emulator should also support as much as possible the ///'s Boot ROM. This means the Boot ROM's routines should work for the most part as-is.

Note: I have a listing of the Boot ROM which could be useful for this emulation discussion.

For the Boot ROM's floppy disk I/O support I recommend that all the gory details here not be supported directly at the memory-mapped I/O level but instead the /// emulator should emulate this I/O. Specifically, the /// emulator should intercept any access to the Boot ROM routines which read or write disk blocks and use the appropriate target machine operating system routines to accomplish this feature.

The /// emulator should also initialize the ROM's character set which the ROM normally loads into a special RAM chip that is not accessible to the ///'s 6502 processor. See section MEMORY BANK SWITCHING EMULATION for more details.

7.0 MEMORY-MAPPED I/O EMULATION

All memory-mapped I/O locations that in some way deal with the physical world need to be handled by the /// emulator. These areas include such addresses as the speaker addresses. The Apple /// Service Reference Manual provides detailed information about these addresses.

All accesses to memory by the /// emulator must respect the bank switch and environment register settings so that the emulator does not try to access a memory-mapped address when that address is not mapped into the 6502 address space.

Programs which access low-level I/O locations such as the disk I/O addresses should not be supported. I assume most /// programs will access hardware components using SOS or device drivers.

Note: Chris Smolinski says that emulating the low-level stuff on a Power PC-based Macintosh is not very difficult and works rather fast (he's implemented in his SARA emulator the ///'s floppy disk I/O).

8.0 MEMORY BANK SWITCHING EMULATION

The /// emulator must also fully support the ///'s bank switched and enhanced indirect addressing memory architecture. Detailed descriptions and usage of /// memory handling can be found in the Apple /// SOS Reference Manual.

The /// emulator should also support the ///'s character set RAM chip. This holds the bitmap descriptions of each of the 128 characters in the /// character. This RAM area, which is not accessible to the ///'s 6502 CPU, holds 1024 bytes. See the Apple /// Standard Device Drivers Manual (Console Character Sets section) for more information.

Note: I believe the storage of the Boot ROM character set is different than the storage of the character set in the SOS.DRIVER file. I believe the ROM character set has bits that are reversed compared to the SOS.DRIVER character set.

The storage of text and graphics in memory should be supported also. This should happen automatically when a /// program writes to the text/graphic memory buffers. The emulator needs to detect such writes and update its screen as appropriate.

9.0 SOS SYSTEM CALL EMULATION

The majority of system calls to SOS and its drivers should most likely not be intercepted by the /// emulator. But certain calls may need to be intercepted unless a lower level of the /// emulator intercepts these feature already. System calls to SOS or drivers that may need intercepting by the /// emulator could be:

o Disk I/O	(.D[1-4] and .PROFILE drivers)
o Keyboard I/O	(.CONSOLE driver)
o Screen I/O	(.CONSOLE and .GRAPHIC drivers)
o Sound generation	(.AUDIO driver)
o Serial port I/O	(.RS232 driver)
o Silentype Printer	(.SILENTYPE) [I'm not sure about support for this]
o Clock I/O	(Y2K dates may be a problem)

I recommend that the /// emulator intercept all activity dealing with the above and have the target machine perform the equivalent features. For example, to read or write a disk block the /// emulator should have a routine that accesses the appropriate location in the disk image file.

The /// emulator may also provide the user with some type of setup options so that the user can specify specific properties of some of the above drivers. For example, if the target machine supports several output ports the emulator may let the user specify which port to use (e.g. for the .PRINTER driver the user could assign it to a specific serial or parallel port on the target machine).

Note: The ///'s clock does not support the year 2000 or greater. I think the emulator should support Y2K dates but I'm not sure if SOS's file system date stamps will support this easily.

10.0 DEVICE DRIVER EMULATION

This section is for the most part handled by my comments in section SOS SYSTEM CALL EMULATION. I suspect the programming within the /// emulator for this area could be the most work since there are lots of device drivers that make up a simple Apple /// configuration.

One area of device drivers that the /// emulator may not want to emulate is interrupt handling. Since the emulator does not have physical devices connected to it in any direct fashion I don't think interrupts exist as far as the emulator is concerned. Interrupts dealing with disks or the keyboard can be handled at a lower level by having the /// emulator call the appropriate system call in the target machine. These low-level I/O handlers should set up the appropriate driver data areas so that the rest of the ///'s software (SOS and the interpreter) will work correctly. For example, keyboard I/O should be setup in the /// emulator so that when the keyboard input memory-mapped I/O location is accessed the target machine OS really reads the keyboard and sets up the memory-mapped location as appropriate.

11.0 KEYBOARD SUPPORT

11.1 User interface support

The /// computer's keyboard layout is basically compatible with modern keyboards. The /// keyboard does have two extra keys, Open Apple and Closed Apple which are positioned to the left of the Apple /// keyboard. Also present on the keyboard are four arrow keys. The emulator should support these keys either directly (i.e., the target machine has similar keys) or associate other keys with the ///'s special keys (e.g., the Macintosh computer's two Option keys could be used to simulate the special Open and Closed Apple keys). The emulator's associated keys need not physically be in the same location as the ///'s special keys but having them in the general area will be beneficial.

Note: The /// Plus keyboard contains an extra key, Delete, compared to the /// keyboard.

11.2 Low-level access

The /// emulator should handle low-level access to the keyboard memory-mapped I/O locations as detailed in section DEVICE DRIVER EMULATION.

12.0 MONITOR SUPPORT

The emulator should support the Apple's built-in ROM Monitor. Entry to the Monitor should be similar to how this is done on a real /// (at startup if Open Apple and Control keys are pressed). The code in the ROM which tests for Monitor entry should work.

13.0 APPLE] [EMULATION DISK SUPPORT

It would be nice if the /// emulator supported the Apple] [Emulation Disk. I'm not sure of what would be involved here but suspect that if the ///'s 6502 CPU and the memory-mapped I/O locations are robustly supported that the] [emulation should work also without any special additional /// emulation features.

Special consideration may need to be given to Apple /// keyboard keys which do not exist in the Apple] [world.] [emulation details can be found in the Apple /// Owner's Guide and the Apple /// Service Reference Manual.

Note: I have a disassembled listing of the Apple] [Emulation Disk ROM source listing which could prove useful in this area.

Further analysis of the] [emulation disk's boot sequence needs to be done since I'm unknowledgable about this area. Also, I've heard that the] [emulation accesses an I/O location which disables some /// features.

14.0 WHAT LANGUAGE SHOULD THE /// EMULATOR BE WRITTEN IN?

I highly recommend that the /// emulator be written in a high level language such as Pascal or C. This should make the emulator more compatible with different target computers and make development and maintenance of the emulator much easier. I recommend avoiding low-level languages such as assembly.

15.0 WHAT TARGET MACHINES SHOULD BE SUPPORTED?

I recommend that the target machine (or machines) for the emulator be machines that are commonly used today by most computer users. This means either the IBM PC or the Apple Macintosh machine family. For the PC world I recommend the /// emulator run under Windows 95 and Windows NT. For the Macintosh world I recommend the emulator run on most Macintosh models which means support the Macintosh 512 and above. Color display should also be supported by the /// emulator (for the Macintosh this means use Color QuickDraw if the machine supports CQD and if CQD is not supported by a Macintosh model use the Classic B/W QD and maybe use patterns as "colors").

Any of these machines should be fast enough to emulate the /// and most likely will be too fast in many areas. I recommend some type of speed control be built into the emulator so that users can control how fast the emulator works. For many /// programs (e.g. AppleWriter /// and VisiCalc ///) emulation speed will be immaterial since these programs typically wait for the user to enter data and then do their thing. But for programs such as games the user will want to control the emulator speed otherwise the game's actions will be super fast and unplayable.

Some people say that the older machines such as pre-68040 Macintoshs will be too slow for a reasonable /// emulator. I would like to see this /// emulator run on a Mac 512 machine an onwards. Running on a Mac 128 machine seems a problem due to this machine's small memory size and should not be supported (if a virtual memory scheme was used by the emulator the Mac 128 could be supported but I think having this extra level of support in the emulator would not be worth it). I disagree and am willing to wager a small sum that I'm right.

16.0 EMULATOR DEBUGGING FACILITIES

The emulator should support a comprehensive built-in debugger. This debugger's purpose should be to let the sophisticated emulator user access any part of the emulator's /// address space. This should include all of the memory that is allocated to the /// as its memory. This memory would encompass the 256K (or 512K) of /// RAM, the /// ROM (4K), the character set RAM (1K), the 6502 registers, and the special /// registers (e.g. bank register).

This debugger will prove invaluable in diagnosing emulator bugs. Not only will the user be able to type commands for the debugger but the emulator will be able to send messages to the debugger.

Logging of all debugger sessions should be stored to a text file for possible analysis. This text file would be created when the emulator starts. The log file should be appended to by the emulator. Only the user can delete the file.

The debugger should exist as a separate window that does not in any way affect the emulator's main window. This window should display only commands that the user enters or replies returned by the debugger. There should not exist a separate window area showing things such as the 6502 registers since all such information should appear in the debugger log file. The window should support at least 80 columns of text and 24 rows.

The emulator user interface should be based on a simple command line control scheme. All commands and command outputs should be text-based. This scheme could be based on the ///'s Monitor's commands or on a little more readable command scheme such as in Apple's MacsBug debugger. There should be full on-line help that discusses the debugger commands in general and each command should also have on-line help available. The debugger should show at the beginning of each line a prompt character to indicate when it is waiting for a command. I recommend the prompt be the ">" character. The debugger should also show a cursor which I recommend to be a black square.

The debugger should support the standard debugging commands such as displaying/setting memory, displaying/setting registers, and disassembling 6502 instructions. This disassembly should support the special SOS BRK call by listing the word "BRK/SOS" instead of just "BRK" and following this with the SOS command number/name and the parameter list address:

```
  SOS  C0/CREATE  345A
```

The user should be able to enable or disable this feature.

Note: It may be good to also support the Apple][ProDOS command calling scheme in case this emulator ever becomes an Apple][emulator.

The debugger should support break points, single stepping, and timing buckets. The timing buckets would be used in conjunction with break points to record how long a sequence of 6502 instructions took to execute. This can be very useful in locating emulator bottlenecks. The debugger supports many break point commands since I have a feeling that this facility will be very powerful and useful during the emulator's development.

The debugger should support the collection of statistics about the emulator. I recommend tracking how many times specific 6502 opcodes are executed (obviously, the debugger would need commands to display and clear this information). I would also track memory accesses on at least a page (256 bytes) basis.

The debugger should be accessible at any time that the emulator is running. I recommend some type of key press combination that the emulator would detect and display the debugger window. Once the debugger window is active it should remain on the screen until the user closes the window.

The emulator should also support a special key press combination at emulator startup time that activates the debugger just before the /// ROM is run. This can give the emulator developer a good way of tracing ROM execution.

The emulator should activate the debugger if any fatal emulation errors are detected and the debugger should show a message detailing the reason for the activation. All of these errors display a dump of the 6502 and SOS control registers. Reasons for debugger activation from the emulator are:

1. A program writes to write-protected memory (e.g. SOS's address space). The displayed message is "EMULATOR EXCEPTION: WRITING TO WRITE-PROTECTED MEMORY".

2. A program executes an undefined 6502 instruction (e.g. 6502 opcode $02). The displayed message is "EMULATOR EXCEPTION: UNDEFINED 6502 OPCODE".

When the debugger is initialized (which should be when the emulator starts) the debugger should check if a text file named "DDT.TXT" exists. If so, the debugger should read each line from this file and execute it. Obviously, this file should contain debugger instructions. This can be very useful for setting up commonly used break points which if you use many would be tedious to type everytime you wanted to use the emulator.

A memory snapshot facility should also exist. When activated by a debugger command this facility would write to the host computer's disk a binary file containing a copy of all the /// memory areas. This snapshot should also be readable by the debugger so that the user could restart a specific emulation session from the snapshot.

I recommend the following emulator debugger commands which are based on the /// Monitor commands so that these debugger commands will be familiar to Monitor users. These commands for the most part have the general syntax of address-command. See my document "Inside the Apple /// Computer ROM" for a list of the /// Monitor commands. For information about the Apple][Monitor commands, which the /// Monitor commands are based upon, see "Apple][Reference Manual" (Chapter 3: The System Monitor, dated 1981).

Addresses appearing in debugger commands may be prefaced by "N/" where N is a bank number. For example, to reference address 2000 of bank 4 use 4/2000. If no bank number precedes an address the current bank is used. To reference a ROM address use a bank "number" of "R", for example "R/F000". To reference a character set address use a bank "number" of "C", for example "C/0000". To reference the SOS system bank use "S", e.g. "S/1400". (K)K

Commands should be case-insensitive (none of the UNIX case-sensitivity gobbly-gook).

Commands that display more than a screen full of information should either automatically pause when the screen is full, or the user can use the SPACE key.

Note: Commands using ":" may also use ";" which is easier to type since this character does not need the user of the shift key. Same for "<" and "/".

Most debugger command numeric arguments must be specified in hexadecimal. The exception is the X command which supports hexadecimal, decimal, and binary.

The debugger command parser should be very liberal. This means that users should be able to include extra spaces (or no spaces) and the command should be parsable. For example, if a command needs a list of bytes the user should be able to enter any of the following: "AABBCC", "AA BB CC", " A ABBC C " and the debugger will see these as "AABBCC".

The debugger should also support a command macro facility. This facility allows you to define a macro consisting of other debugger commands. Typing the name of the macro will then type the commands as if you entered them manually.

```
============================================================
```

HELP (or ?) cmd name
2 sps

Display debugger on-line help for all commands. Help info should be stored in an external text file for easier modification. I recommend that this section of this document be the help file.

cmd name = 1st part or all cmd name

Example: HELP -- shows all help info

Note: ? also same

HELP S -- show all cmds starting w/ S

HELP SS -- shows SS cmd

```
------------------------------------------------------------
```

BYE

Return to the emulator.

Example: BYE

```
------------------------------------------------------------
```

CARRIAGE RETURN keypress
2 sps

Repeat last command.

Example: If the last command was HELP and you press the CARRIAGE RETURN key then HELP will be displayed and executed again.

```
------------------------------------------------------------
```

SPACE keypress
2 sps

Pause current command's output. Press again to continue.

Example: If a command is executing and you press the SPACE key the comand's output will be paused, pressing SPACE again resumes the command's output. Pausing/Resuming are done on an output line basis only.

```
------------------------------------------------------------
```

DELETE keypress
2 sp

Stop current command's output.

Example: If a command is executing and you press this key then the command will stop executing and you will be returned to the debugger's prompt.

```
------------------------------------------------------------
```

RD

Display 6502 registers and /// system control registers.

Show table explaining bits in P and E

Example: RD

```
                    bit names                     bit names
A=04  X=01  Y=D8  P=30/00000011  S=F8  PC=034A : E=77/01110111  Z=1A  B=03
 1 sp                NV_BDIZC                      SIVRWSRR
                     unused - shows 0 or 1
```

N negative
V overflow
B break command
D decimal mode
I interrupt disable
Z zero
C carry

S - System clock rate
I - I/O space
C - Screen (~~C~~ video) state

R - Reset enable
W - Write protect
S - Stack in use

R - ROM
R - ROM

---

byte:SA

Set 6502 A register to byte.

Example: 45:SA

---

byte:SX

Set 6502 X register to byte.

Example: 7B:SX

---

byte:SY

Set 6502 Y register to byte.

Example: FF:SY

---

byte:SP

Set 6502 P register to byte.

Example: 56:SP

---

byte:SS

Set 6502 S register to byte.

Example: AA:SS

---

word:SPC

Set 6502 PC register to word.

Example: 2000:SPC

---

byte:SE

Set /// E system control register to byte.

Example: 34:SE

---

byte:SZ

Set /// Z system control register to byte.

Example: 19:SZ

---

byte:SB

Set /// B system control register to byte.

Example: 06:SB

```
------------------------------------------------------------
```

addr1.addr2

Dump memory data to screen from address 1 to address 2 and display ASCII character at the right of the screen.

```
Example (assumes current bank is bank 4): 300.30F      -- (assumes bank 4 is
                                                           current)
           0  1  2 3 ...
4/0300- B900 080A 0A0A 9900  08C8 D0F4 A62B A909 [F..d.uy%^&90@..G]
```

```
------------------------------------------------------------
```

ZPAGE

Dump the contents of the current interpreter's Zero Page (256 bytes). Also supported are commands for the Stack Page and the Extend Page:

```
   SPAGE    -   stack page
   EPAGE    -   extend page
```

To dump the pages for SOS (and drivers) use the following commands:

```
   SZPAGE   -   zero page
   SSPAGE   -   stack page
   SEPAGE   -   extend page
```

```
Example:  ZPAGE

Zero Page (interpreter)
        00 1 2 3 4 5 6 7 ...
1400- 0123456789ABCDEF 0123456789ABCDEF 0123456789ABCDEF 0123456789ABCDEF
1420- 0123456789ABCDEF 0123456789ABCDEF 0123456789ABCDEF 0123456789ABCDEF
...
14E0- 0123456789ABCDEF 0123456789ABCDEF 0123456789ABCDEF 0123456789ABCDEF
```

```
------------------------------------------------------------
```

addr:bytes

Store starting at the address the bytes.

```
Example: 2000:AA BB CC DD EE FF      --
         2000:AABBCCDDEEFF           --
```

```
------------------------------------------------------------
```

addr:'text'

Store text starting at address (high bit clear).

```
Example: 2000:'Hello World'         --
         2000:'David''s Dog'        -- this stores David's Dog
```

```
------------------------------------------------------------
```

addr:"text"

Store text starting at address (high bit set).

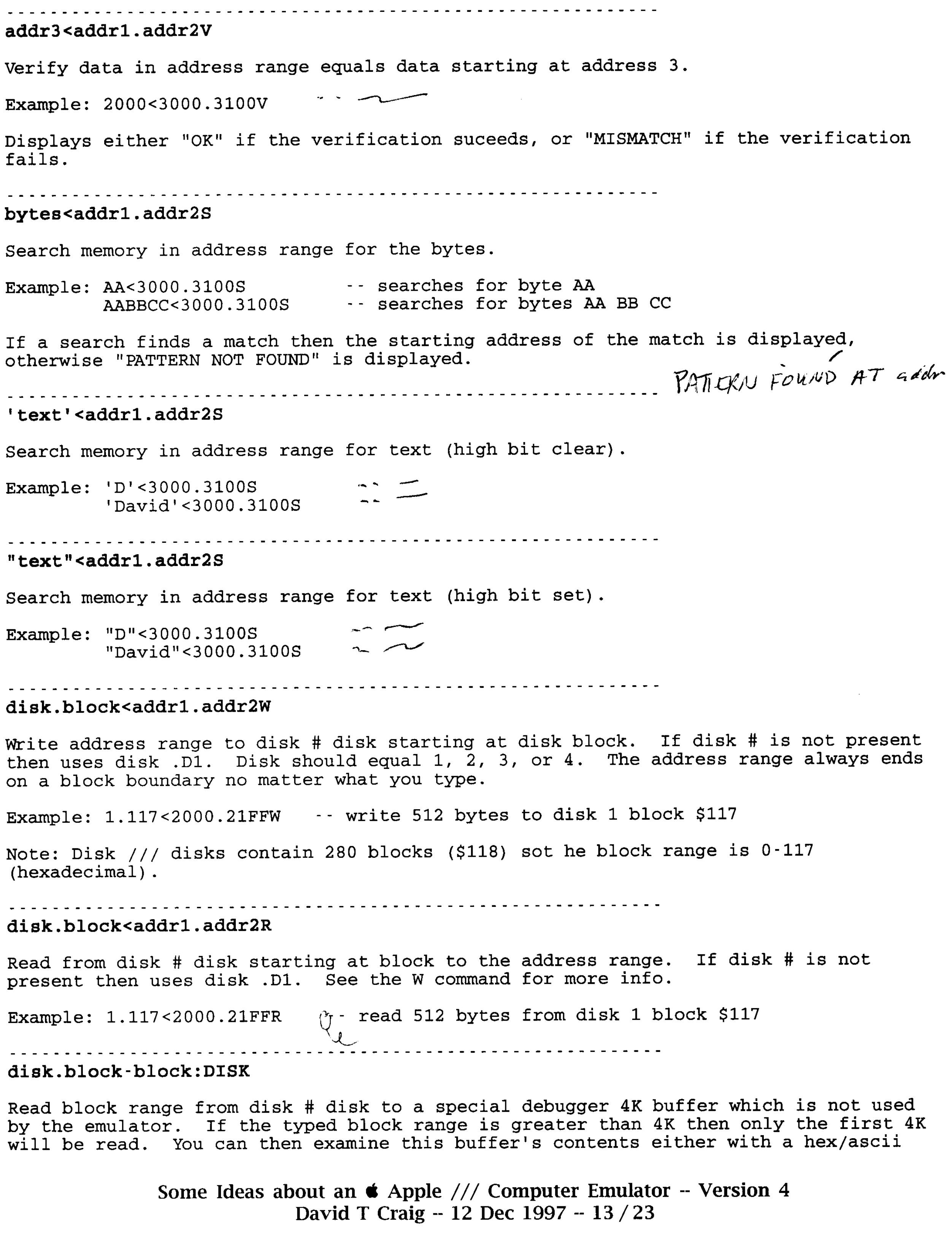
```
Example: 2000:"Hello World"         --
         2000:"David's Dog"         -- this stores David's Dog
         2000:"I said ""Hi"""                      I said "Hi"
```

```
------------------------------------------------------------
```

addr3<addr1.addr2M

Move data in address range to address 3.

```
Example: 2000<3000.3100M       --
```



---

addr3<addr1.addr2V

Verify data in address range equals data starting at address 3.

```
Example: 2000<3000.3100V
```

Displays either "OK" if the verification suceeds, or "MISMATCH" if the verification fails.

---

bytes<addr1.addr2S

Search memory in address range for the bytes.

```
Example: AA<3000.3100S         -- searches for byte AA
         AABBCC<3000.3100S     -- searches for bytes AA BB CC
```

If a search finds a match then the starting address of the match is displayed, otherwise "PATTERN NOT FOUND" is displayed.

PATTERN FOUND AT addr

'text'<addr1.addr2S

Search memory in address range for text (high bit clear).

```
Example: 'D'<3000.3100S
         'David'<3000.3100S
```

"text"<addr1.addr2S

Search memory in address range for text (high bit set).

```
Example: "D"<3000.3100S
         "David"<3000.3100S
```

---

disk.block<addr1.addr2W

Write address range to disk # disk starting at disk block. If disk # is not present then uses disk .D1. Disk should equal 1, 2, 3, or 4. The address range always ends on a block boundary no matter what you type.

```
Example: 1.117<2000.21FFW   -- write 512 bytes to disk 1 block $117
```

Note: Disk /// disks contain 280 blocks ($118) sot he block range is 0-117 (hexadecimal).

---

disk.block<addr1.addr2R

Read from disk # disk starting at block to the address range. If disk # is not present then uses disk .D1. See the W command for more info.

```
Example: 1.117<2000.21FFR   -- read 512 bytes from disk 1 block $117
```

---

disk.block-block:DISK

Read block range from disk # disk to a special debugger 4K buffer which is not used by the emulator. If the typed block range is greater than 4K then only the first 4K will be read. You can then examine this buffer's contents either with a hex/ascii

“APPLE_PAT_4_383_296_F_14” 153 KB 2000-02-28 dpi: 300h x 300v pix: 2206h x 3067v

dump or with a disassembly (command L). This command is useful when you want to examine a disk's contents. For disassembly purposes, you can specify the logical starting address for the buffer. See the DISKBUFFER command.

To disassemble the special disk buffer (see the L command) use bank X (stands for "extra") as part of the disassembly address parameter (e.g. "X/100"). Same for dumping memory or whatever commands you want to use with this special buffer.

```
Example: 1.0-7:DISK      -- read 8 blocks (0 to 7) from disk 1
```

----

addr:DISKBUFFER

Set disk buffer starting logical address. Default address is 2000. See the DISK command.

```
Example: A000:DISKBUFFER   --
```

Range is 0000-FFFF

----

addr1.addr2L

Disassemble instructions in address range. If only addr1 appears then disassemble 20 instructions. Disassembly includes the opcode cycle count.

```
Example: 300L    -- assumes bank 4 is current

4/0300-   A9 C1      'X.'   (2)  LDA #$C1         ;
4/0302-   20 ED FD   '...'  (5)  JSR $FDED        ;
4/0305-   18         '.'    (2)  CLC              ;
4/0306-   69 0A      'T.'   (4)  ADC #$01         ;
4/0308-   C9 DB      '..'   (3)  CMP #$DB         ;
4/030A-   D0 F6      '..'   (3)  BNE $0302        ;
4/030C-   60         'U'    (4)  RTS              ;

1         2          3      4    5                6  (see Note)

Note:   Column 1 = bank register/address
        Column 2 = memory bytes
        Column 3 = ASCII for the memory bytes
        Column 4 = opcode cycle count
        Column 5 = disassembled instructions
        Column 6 = remark character ";" (optional, see DISASMREM)
```

L by itself disassembles the next 20 instructions.

DISASMREM

Display ";" after each disassembly line that is produced by the L command. Default is to not display the remark. Useful if you plan to add comments to a disassembly. See also DISASMREMOFF.

```
Example: DISASMREM
```

DISASMREMOFF

Turn off DISASMREM. See also DISASMREM.

```
Example: DISASMREMOFF
```

----

addrG

Call subroutine at the address.

```
Example: A000G
```

---

addrJ

Jump to the address.

```
Example: A000J
```

---

wordX

Convert word (or up to 4 hex digits) to hexadecimal, decimal, and binary (X stands for "translate"). Prefix character for byte determines its base: no prefix = hex, . = dec, t = binary.

Put in a table for easier viewing

```
Example: AX       ->       A(16)      10(10)  0000 0000 0000 1010(2)
         .10X     ->       A(16)      10(10)  0000 0000 0000 1010(2)
         t1010    ->       A(16)      10(10)  0000 0000 0000 1010(2)
         FFFFX    ->    FFFF(16)   65535(10)  1111 1111 1111 1111(2)
```

---

addr1.addr2:CS

Calculate and display a checksum for address range. Checksum is a 4 byte quantity which is calculated the same as the disk image file checksums.

```
Example: 300.500:CS
         CHECKSUM=AF897CEE
```

---

addrT

Trace instructions starting at the address. Each traced instruction displays register contents. Press the SPACE to pause the trace, press DELETE to stop the trace. The displayed registers contain values _after_ the previously listed command executes.

```
Example: A000T   -- assuming bank 4 is current

4/A000-   A9 C1       'X.'    (2)  LDA #$C1
A=C1  X=01  Y=D8  P=30/00000011  S=F8  PC=A002 : E=77/01110111  Z=1A  B=04
4/A002-   20 ED FD    '...'   (5)  JSR $FDED
A=C1  X=01  Y=D8  P=30/00000011  S=F6  PC=FDED : E=77/01110111  Z=1A  B=04
  Lisp                bit names                    bitnames
Note: Press the DELETE key to stop the trace, SPACE to pause/resume.
```

---

addrSS

Single step trace starting at the address. After each step pause and wait for user to press SPACE to continue or DELETE to stop the single step.

```
Example: A000T SS -- assuming bank 4 is current

4/A000-   A9 C1       'X.'    (2)  LDA #$C1
A=C1  X=01  Y=D8  P=30/00000011  S=F8  PC=A002 : E=77/01110111  Z=1A  B=04
  Lisp                 names                        names
Note: Press SS by itself to single step the next instruction, or press CARRIAGE
RETURN to repeat the SS.
```

---

addr:BP

Set a break point at address. When address is accessed the debugger is entered and displays the registers. Up to 100 break points should be supported.

Example: A000:BP

----

addr:BPC

Clear break point at address.

Example: A000:BPC

SOS:BP

Set a break point when a SOS call is made. This means when the BRK opcode is executed. Same as M00:BP.

Example: SOS:BP

Mopcode:BP

Set a break point when opcode is executed.

Example: M60:BP -- set break point when the RTS instruction (60) is executed.

ROM:BP

Set a break point when a call is made to the ROM.

Example: ROM:BP

----

addr1.addr2:BPW

Set a break point when any address within address range is written to. BPW = Break Point Write.

Example: 300.123AR:BPW

----

addr1.addr2:BPR

Set a break point when any address within address range is read from. BPR = Break Point Read.

Example: 300.123A:BPR

make just 1 BPE command

----

addr.byte:BPE

addr1.addr2.byte1 byte2... : BPE (a2 optional)

Set a break point when the address contents equal the byte value. BPE = Break Point Equals.

Example: 300.AA:BPE

addr1.addr2.byte1-byte2 : BPE (a2 optional)

----

addr.byte1-byte2:BPE

Set a break point when the address contents equal a byte value in the byte range. BPE

new cmd
BPNE BP not equals
same syntax as BPE

= Break Point Equals.

Example: 300.AA-BB:BPE

----

addr.byte1 byte2 ... :BPEA

Set a break point when the address contents equal byte 1 value, or equals byte 2 value, etc. Supports up to 16 byte values. BPEA = Break Point Equals Any.

```
Example: 300.AABBCCDD:BPEA
         300.AA BB CC DD:BPEA
```

----

addr1.addr2.byte1 byte2 ... :BPEA

Set a break point when the address range contains any bytes equalling the byte values. BPEA = Break Point Equals Any.

Example: 300.400.AABBCCDD:BPEA

----

addr1.addr2.byte1-byte2:BPEA

Set a break point when the address range contents equal the byte range. BPEA = Break Point Equals Any.

Example: 300.400.AA-BB:BPEA

BPD

Display break point table.

Example: BPD

```
  #  Address Range  BP    Setting
---  -------------  ----  -------
  1  4/2000-4/21FF  BPEA  AA-BB
```

BPC

Clear break point table.

Example: BPC

----

addr1.addr2:TB

Set timing bucket for address range. When address 1 is accessed timing starts. When address 2 is accessed timing stops. Up to 100 timing buckets should be supported.

Example: A000.A1FF:TB

TBD

Display timing bucket table. Shows all set timing buckets and the time in 1/60th of a second and in seconds spent in each bucket.

Example: TBD

```
#    Address Range  Time (1/60s)  Time (secs)
```

```
---  ------------  -----------  ----------
  1  4/A000-4/A1FF           34       0.567
  2  4/A300-4/A310            5       0.083
---  ------------  -----------  ----------
                             39       0.650
```

----

addr:TBC

Clear timing bucket starting at address.

Example: A000:TBC

TBC

Clear timing bucket table.

Example: TBC

----

error:SOSE

List SOS general error message for the error number. If no error number is present then list all general errors. Error info should be stored in an external text file for easier modification. See the SOS Reference Manual for a list of these errors.

Example: 01:SOSE

BADSCNUM - Invalid SOS call number

----

error:SOSFE

Display SOS fatal error message for the error number. If no error number is present then list all fatal errors. See the SOS Reference Manual for a list of these errors.

Example: 01:SOSFE

BADBRK - Invalid BRK

----

command:SOS

Display SOS command name and SOS command area (e.g. file system) for the command number. If no command number present then list all SOS command numbers and their names. Command info should be stored in an external text file for easier modification. See the SOS Reference Manual for a list of these commands.

Example: C0:SOS

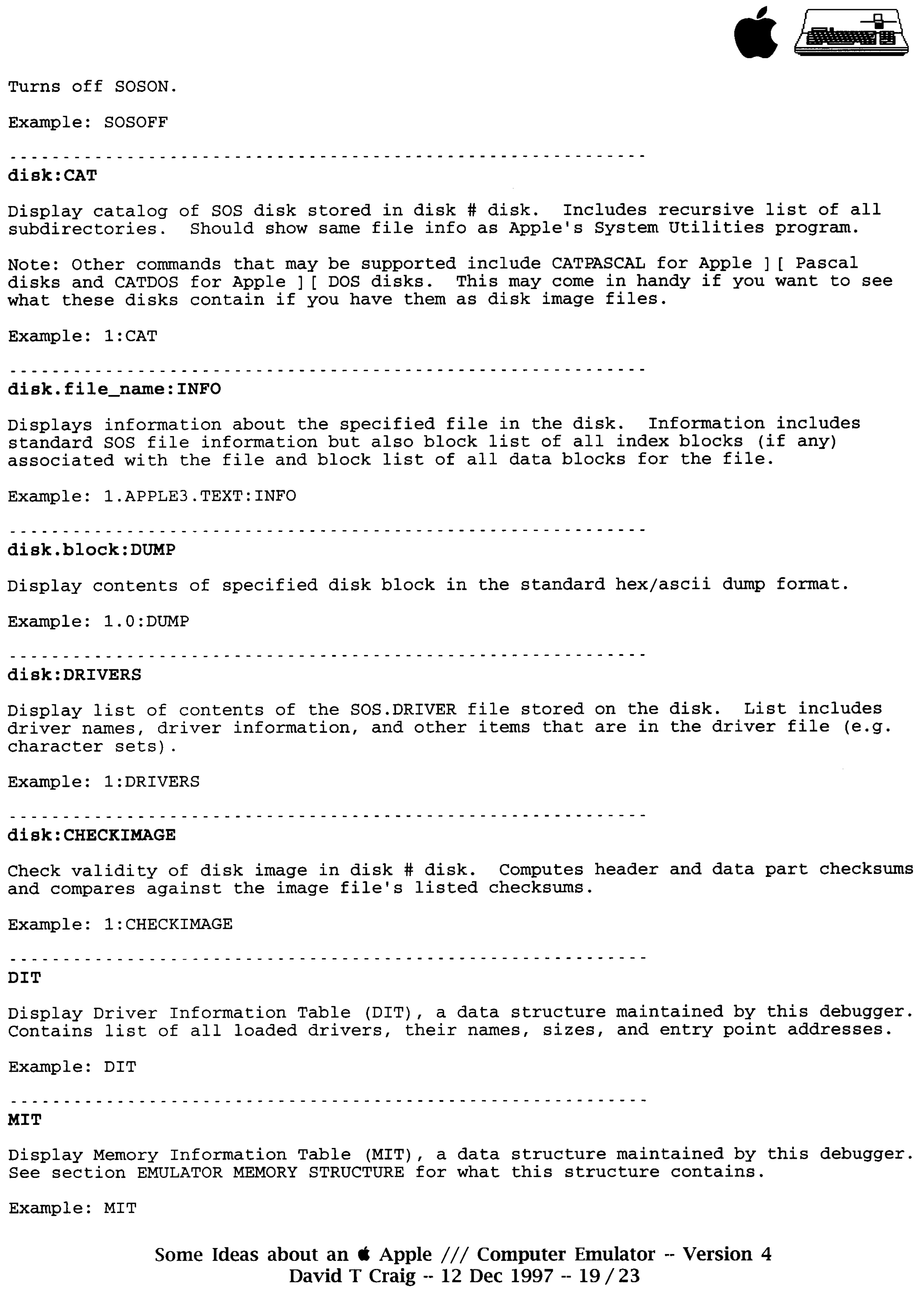
CREATE (File System)

SOSON

Turn on disassembly of SOS calls which displays SOS followed by the command number and parameter address. The emulator defaults to this.

Example: SOSON

SOSOFF



Turns off SOSON.

Example: SOSOFF

----
disk:CAT

Display catalog of SOS disk stored in disk # disk. Includes recursive list of all subdirectories. Should show same file info as Apple's System Utilities program.

Note: Other commands that may be supported include CATPASCAL for Apple][Pascal disks and CATDOS for Apple][DOS disks. This may come in handy if you want to see what these disks contain if you have them as disk image files.

Example: 1:CAT

----
disk.file_name:INFO

Displays information about the specified file in the disk. Information includes standard SOS file information but also block list of all index blocks (if any) associated with the file and block list of all data blocks for the file.

Example: 1.APPLE3.TEXT:INFO

----
disk.block:DUMP

Display contents of specified disk block in the standard hex/ascii dump format.

Example: 1.0:DUMP

----
disk:DRIVERS

Display list of contents of the SOS.DRIVER file stored on the disk. List includes driver names, driver information, and other items that are in the driver file (e.g. character sets).

Example: 1:DRIVERS

----
disk:CHECKIMAGE

Check validity of disk image in disk # disk. Computes header and data part checksums and compares against the image file's listed checksums.

Example: 1:CHECKIMAGE

DIT

Display Driver Information Table (DIT), a data structure maintained by this debugger. Contains list of all loaded drivers, their names, sizes, and entry point addresses.

Example: DIT

MIT

Display Memory Information Table (MIT), a data structure maintained by this debugger. See section EMULATOR MEMORY STRUCTURE for what this structure contains.

Example: MIT

OPCODES

Display a histogram of opcode execution counts. Includes the actual number of the counts. Sorted by frequency. Opcodes not executed are listed below the histogram.

Example: OPCODES

```
LDA   2,188,973  ***********************************************
STA      12,123  ******************************
CMP         467  **************
      ---------
      2,201,563

Unexecuted opcodes: TXS NOP
```

OPCODESCLR

Reset opcode histogram table.

Example: OPCODESCLR

----

page1.page2:MEMORYR

Display memory write access table. This table lists on a 256 byte page basis counts for each time the page was read. If page1.page2 specified then lists only those pages. If a single page is specified then display only that page's access count.

Example: 0.5:MEMORYR

----

page1.page2:MEMORYW

Display memory read access table. This table lists on a 256 byte page basis counts for each time the page was written. See MEMORYW for page options.

Example: 0.5:MEMORYW

MEMORYCLR

Reset both memory access tables.

Example: MEMORYCLR

----

value:SCROLL

if SCROLL > 0 then cmds showing more than screen of info do not pause for user when screen full

Set debugger display scrolling rate interline delay. Value is in 1/10th of a second. Default is no delay (value = 0). Useful if you want to for example dump lots of memory and don't want to mess with the SPACE key to read what is displayed. Set the scrolling delay to a comfortable value, sit back, and enjoy the show.

Example: 10:SCROLL   -- sets scrolling delay to 1 second

----

filename:LOG

Close log file, create a new one with filename, and output all debugger displays to this new file. Useful if you're running the emulator from a write-protected disk and you want to re-direct the output to a writable disk file.

Example: MyDiary:LOG

SNAPSHOTW

Write the contents of all of the emulator's memory to binary file on the host computer's hard disk. This snapshot could prove useful in diagnosing an emulator problem. The binary file should be named "Snapshot_YYYYMMDD_HHMMSS.BIN".

Example: SNAPSHOTW

SNAPSHOTRfile-name

Read a snapshot file into the emulator's memory.

Example: SNAPSHOTR Snapshot_19971225_123456.BIN

MACRO name commands

Define a macro name and commands for this macro. You can use any name containing alphnumeric characters or periods with a maximum length of 31 characters. Up to 25 macros may be defined. All commands are verified and if any syntax errors occur you will be told and the macro will not be defined. Macro commands cannot include other macro commands.

Example: MACRO my.dump 300.400 A000.A1FF A000L

MACROL

List all defined macros.

Example: MACROL

```
 #  Name / Contents
--  -----------------------------
 1  my.dump
    300.400 A000.A1FF A000L
```

!macro-name

Execute a macro with the name "macro-name". Each command within the macro is displayed followed by the commands' display.

Example: !my.dump

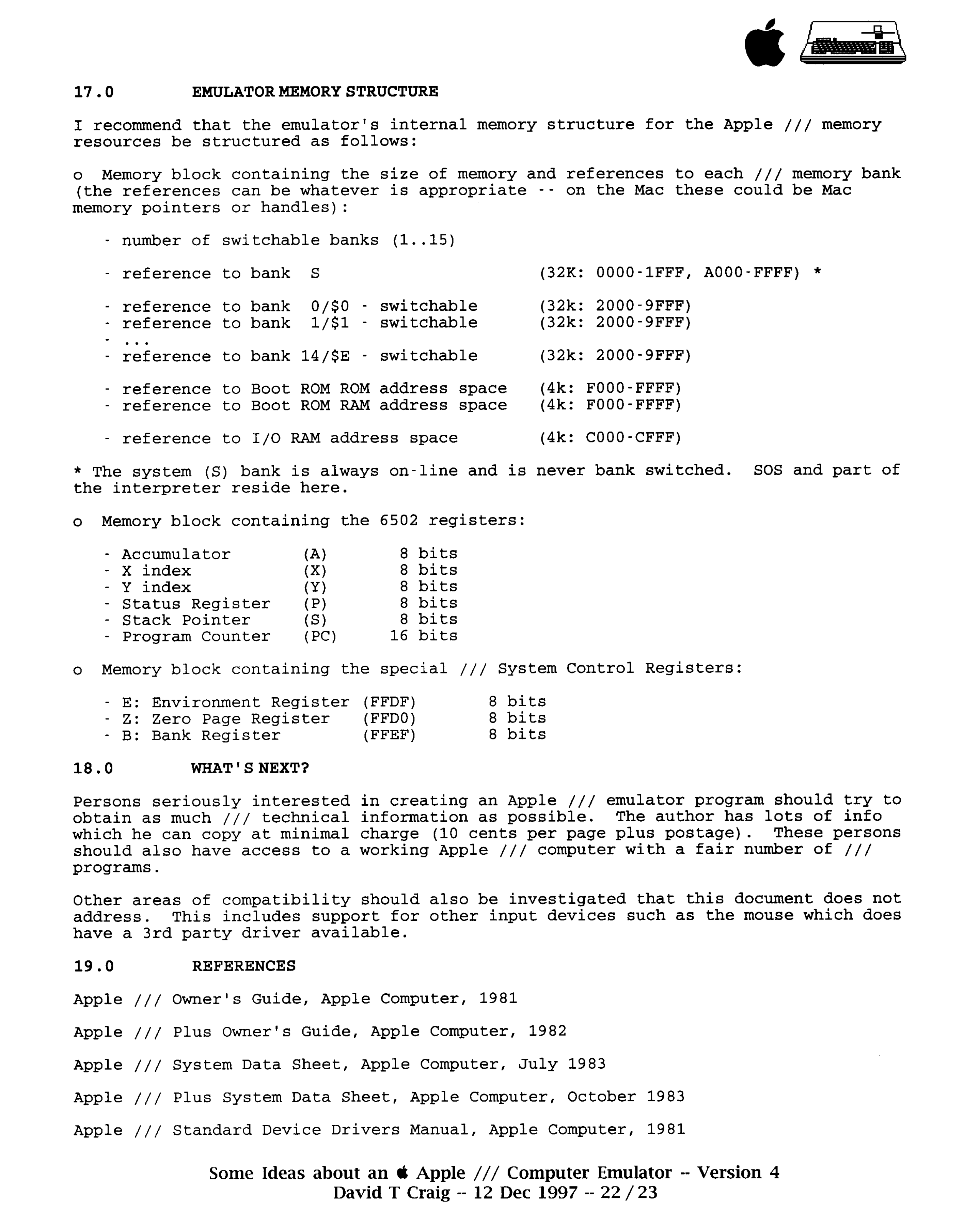
```
300.400
...
A000.A1FF
...
A000L
...
```

char
FONT display current font bitmap FONTROM ROM font bitmap

VERSION

Display debugger version information. Includes version number and creation date/time.

===



17.0 EMULATOR MEMORY STRUCTURE

I recommend that the emulator's internal memory structure for the Apple /// memory resources be structured as follows:

o Memory block containing the size of memory and references to each /// memory bank (the references can be whatever is appropriate -- on the Mac these could be Mac memory pointers or handles):

- number of switchable banks (1..15)

- reference to bank S	(32K: 0000-1FFF, A000-FFFF) *
- reference to bank 0/$0 - switchable	(32k: 2000-9FFF)
- reference to bank 1/$1 - switchable	(32k: 2000-9FFF)
- ...	
- reference to bank 14/$E - switchable	(32k: 2000-9FFF)
- reference to Boot ROM ROM address space	(4k: F000-FFFF)
- reference to Boot ROM RAM address space	(4k: F000-FFFF)
- reference to I/O RAM address space	(4k: C000-CFFF)

* The system (S) bank is always on-line and is never bank switched. SOS and part of the interpreter reside here.

o Memory block containing the 6502 registers:

- Accumulator	(A)	8 bits
- X index	(X)	8 bits
- Y index	(Y)	8 bits
- Status Register	(P)	8 bits
- Stack Pointer	(S)	8 bits
- Program Counter	(PC)	16 bits

o Memory block containing the special /// System Control Registers:

- E: Environment Register	(FFDF)	8 bits
- Z: Zero Page Register	(FFD0)	8 bits
- B: Bank Register	(FFEF)	8 bits

18.0 WHAT'S NEXT?

Persons seriously interested in creating an Apple /// emulator program should try to obtain as much /// technical information as possible. The author has lots of info which he can copy at minimal charge (10 cents per page plus postage). These persons should also have access to a working Apple /// computer with a fair number of /// programs.

Other areas of compatibility should also be investigated that this document does not address. This includes support for other input devices such as the mouse which does have a 3rd party driver available.

19.0 REFERENCES

Apple /// Owner's Guide, Apple Computer, 1981

Apple /// Plus Owner's Guide, Apple Computer, 1982

Apple /// System Data Sheet, Apple Computer, July 1983

Apple /// Plus System Data Sheet, Apple Computer, October 1983

Apple /// Standard Device Drivers Manual, Apple Computer, 1981

“APPLE_PAT_4_383_296_F_23” 176 KB 2000-02-28 dpi: 300h x 300v pix: 2188h x 3042v

Apple /// SOS Reference Manual, Apple Computer, 1982

Apple /// SOS Device Driver Writer’s Guide, Apple Computer, 1982

Apple /// Service Reference Manual (Level 2), Apple Computer, 1983

/// Bits: John Jeppson’s Guided Tour of Highway ///, Softalk magazine, May 1983

Bank Switch Razzle-Dazzle, Softalk magazine, August 1982

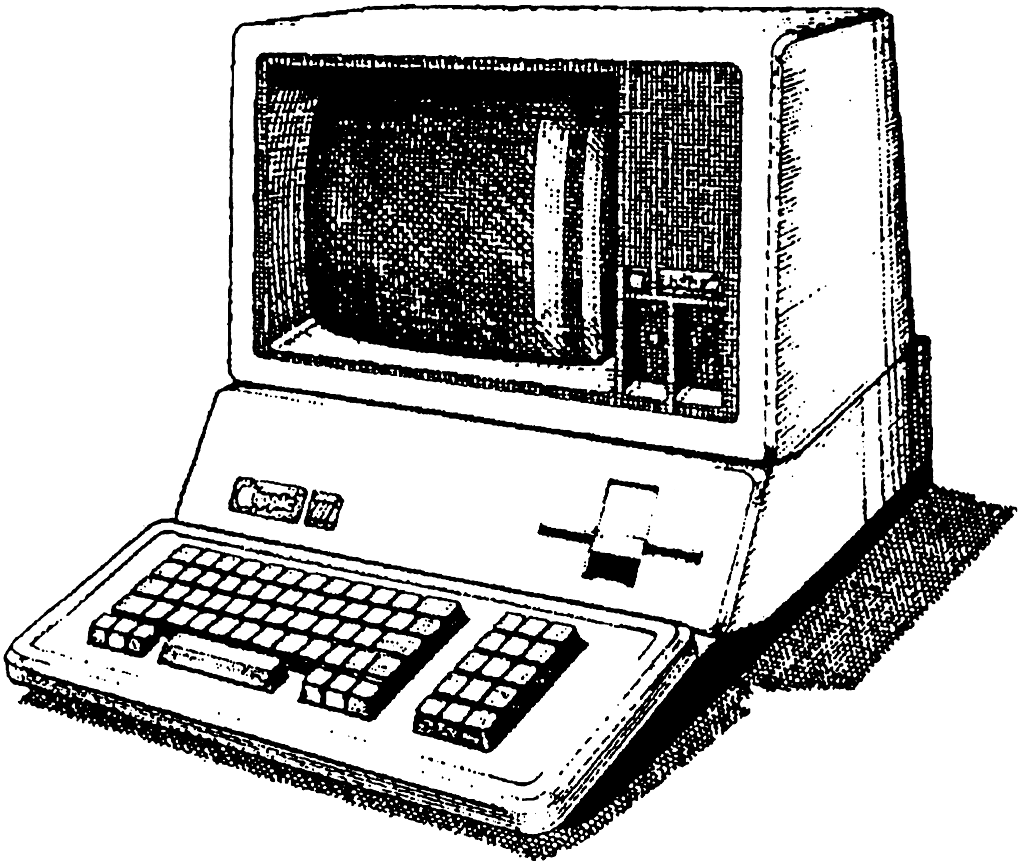
The Apple Nobody Knows, Apple Orchard magazine, Fall 1981

Apple /// Entry Points, Andy Wells, Call-APPLE, October 1981

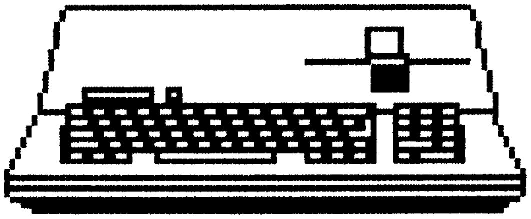
Inside the Apple /// Computer ROM, David Craig, November 1997

###

Apple III Computer Information



“APPLE_PAT_4_383_296_F_25” 675 KB 2000-02-28 dpi: 1200h x 1200v pix: 6368h x 6713v

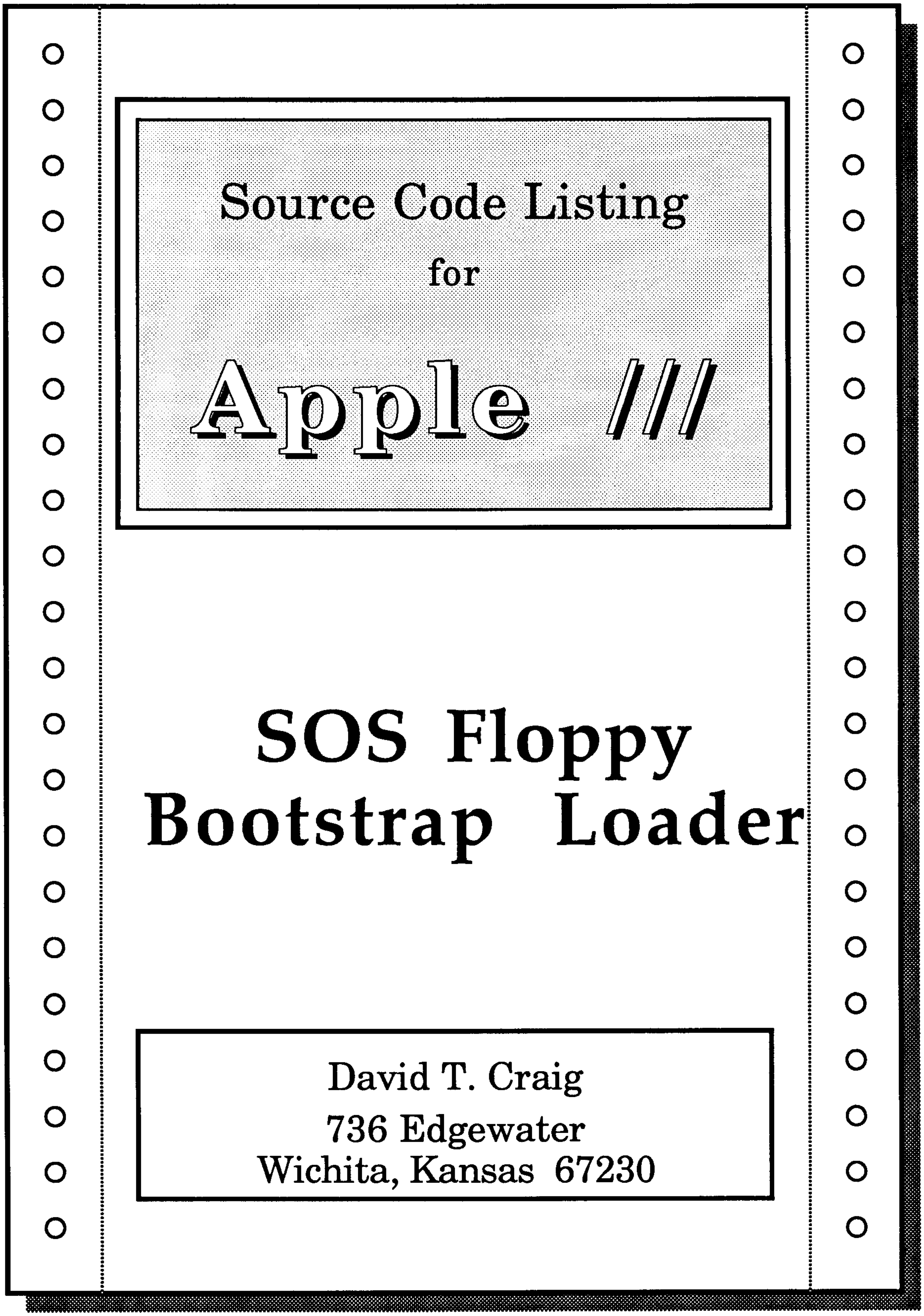


Apple ///
Apple ///+

Apple /// SOS Technical Information

SOS 1.3 Floppy Bootstrap Loader Source Code Listing

This listing shows the code which is found at the beginning of a SOS boot disk. When the Apple /// computer starts the computer's ROM loads this code from the floppy disk and executes the code. This code loads the Apple ///'s operating system, SOS.

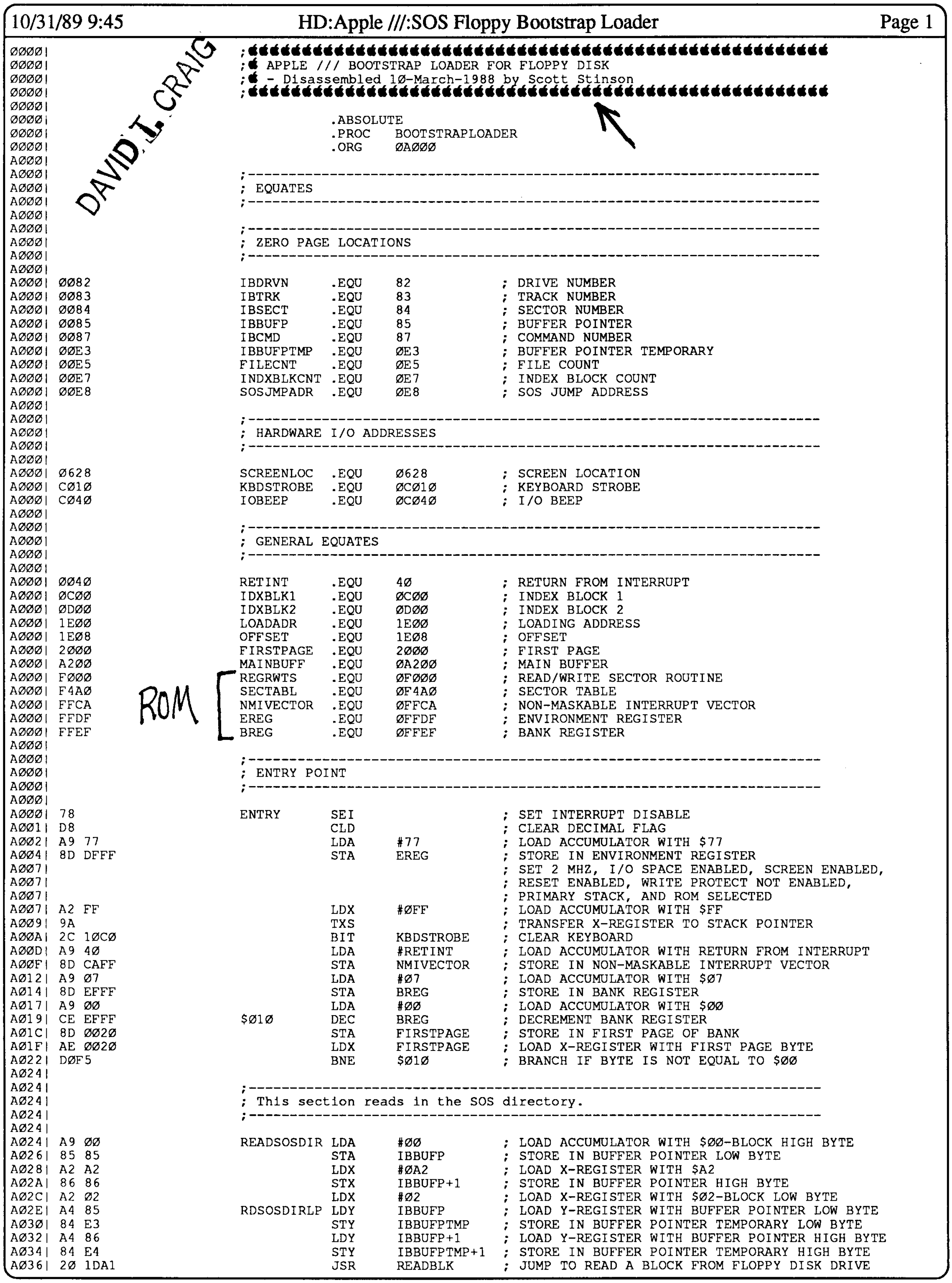


# Source Code Listing
for
Apple ///

SOS Floppy Bootstrap Loader

David T. Craig
736 Edgewater
Wichita, Kansas 67230

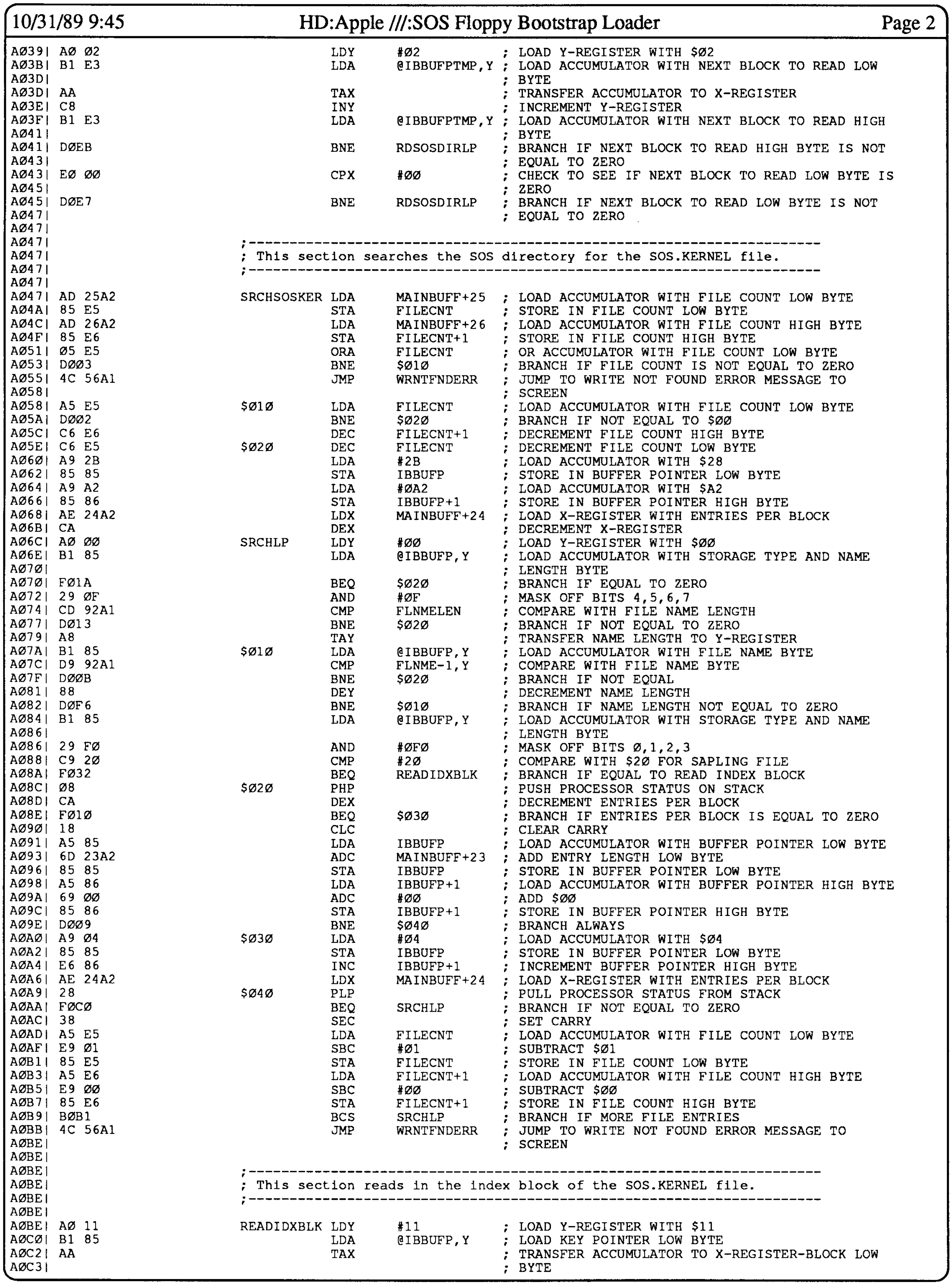
"APPLE_PAT_4_383_296_G_02" 172 KB 2000-02-29 dpi: 300h x 300v pix: 1928h x 2709v



10/31/89 9:45 HD:Apple ///:SOS Floppy Bootstrap Loader Page 1

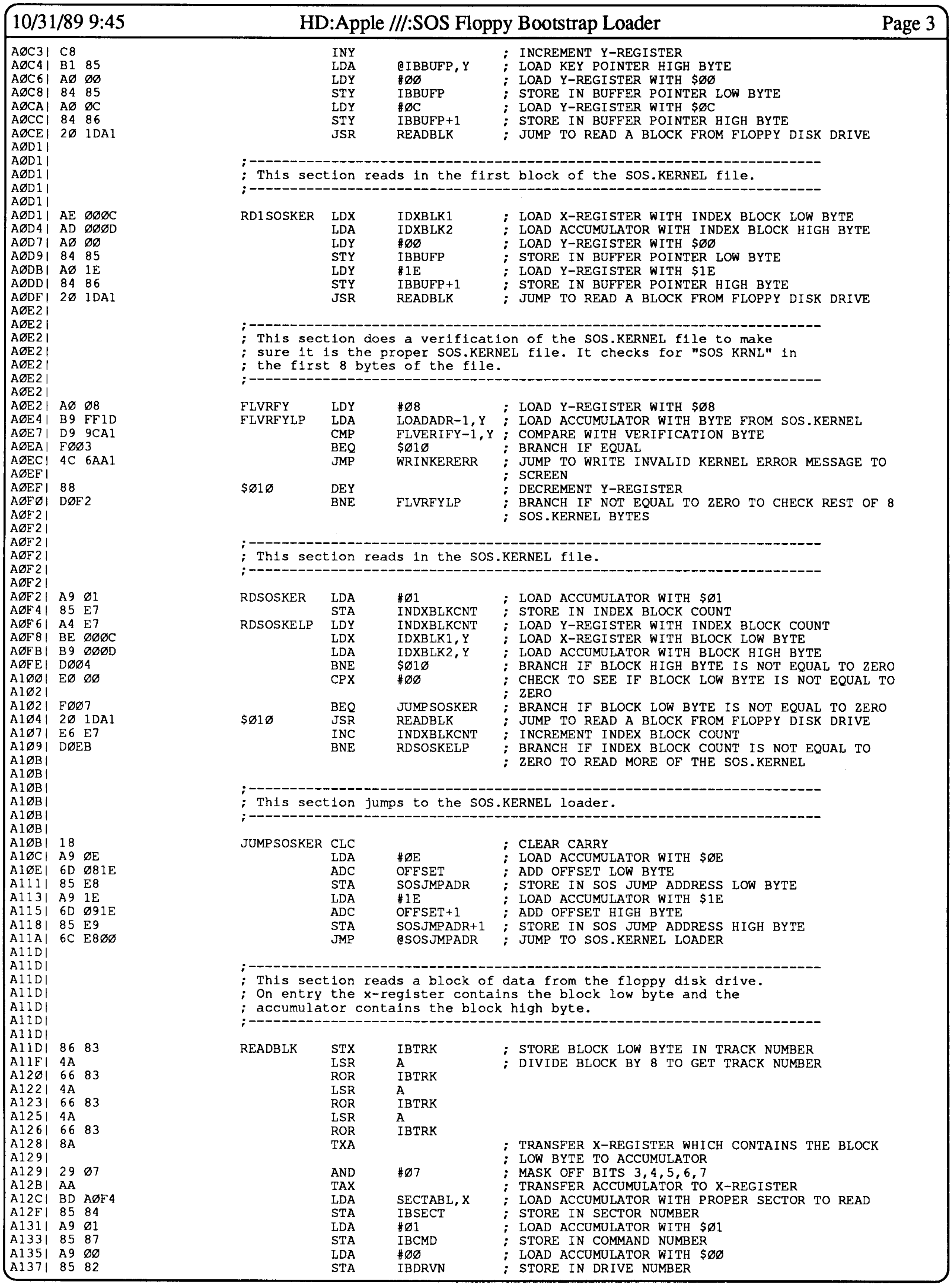
DAVID T. CRAIG

```
ØØØØ|                      ;
ØØØØ|                      ; APPLE /// BOOTSTRAP LOADER FOR FLOPPY DISK
ØØØØ|                      ; - Disassembled 1Ø-March-1988 by Scott Stinson
ØØØØ|                      ;
ØØØØ|
ØØØØ|                                 .ABSOLUTE
ØØØØ|                                 .PROC   BOOTSTRAPLOADER
ØØØØ|                                 .ORG    ØAØØØ
AØØØ|
AØØØ|                      ;----------------------------------------------------------------------
AØØØ|                      ; EQUATES
AØØØ|                      ;----------------------------------------------------------------------
AØØØ|
AØØØ|                      ;----------------------------------------------------------------------
AØØØ|                      ; ZERO PAGE LOCATIONS
AØØØ|                      ;----------------------------------------------------------------------
AØØØ|
AØØØ| ØØ82                 IBDRVN     .EQU    82           ; DRIVE NUMBER
AØØØ| ØØ83                 IBTRK      .EQU    83           ; TRACK NUMBER
AØØØ| ØØ84                 IBSECT     .EQU    84           ; SECTOR NUMBER
AØØØ| ØØ85                 IBBUFP     .EQU    85           ; BUFFER POINTER
AØØØ| ØØ87                 IBCMD      .EQU    87           ; COMMAND NUMBER
AØØØ| ØØE3                 IBBUFPTMP  .EQU    ØE3          ; BUFFER POINTER TEMPORARY
AØØØ| ØØE5                 FILECNT    .EQU    ØE5          ; FILE COUNT
AØØØ| ØØE7                 INDXBLKCNT .EQU    ØE7          ; INDEX BLOCK COUNT
AØØØ| ØØE8                 SOSJMPADR  .EQU    ØE8          ; SOS JUMP ADDRESS
AØØØ|
AØØØ|                      ;----------------------------------------------------------------------
AØØØ|                      ; HARDWARE I/O ADDRESSES
AØØØ|                      ;----------------------------------------------------------------------
AØØØ|
AØØØ| Ø628                 SCREENLOC  .EQU    Ø628         ; SCREEN LOCATION
AØØØ| CØ1Ø                 KBDSTROBE  .EQU    ØCØ1Ø        ; KEYBOARD STROBE
AØØØ| CØ4Ø                 IOBEEP     .EQU    ØCØ4Ø        ; I/O BEEP
AØØØ|
AØØØ|                      ;----------------------------------------------------------------------
AØØØ|                      ; GENERAL EQUATES
AØØØ|                      ;----------------------------------------------------------------------
AØØØ|
AØØØ| ØØ4Ø                 RETINT     .EQU    4Ø           ; RETURN FROM INTERRUPT
AØØØ| ØCØØ                 IDXBLK1    .EQU    ØCØØ         ; INDEX BLOCK 1
AØØØ| ØDØØ                 IDXBLK2    .EQU    ØDØØ         ; INDEX BLOCK 2
AØØØ| 1EØØ                 LOADADR    .EQU    1EØØ         ; LOADING ADDRESS
AØØØ| 1EØ8                 OFFSET     .EQU    1EØ8         ; OFFSET
AØØØ| 2ØØØ                 FIRSTPAGE  .EQU    2ØØØ         ; FIRST PAGE
AØØØ| A2ØØ                 MAINBUFF   .EQU    ØA2ØØ        ; MAIN BUFFER
AØØØ| FØØØ                 REGRWTS    .EQU    ØFØØØ        ; READ/WRITE SECTOR ROUTINE
AØØØ| F4AØ                 SECTABL    .EQU    ØF4AØ        ; SECTOR TABLE
AØØØ| FFCA      ROM        NMIVECTOR  .EQU    ØFFCA        ; NON-MASKABLE INTERRUPT VECTOR
AØØØ| FFDF                 EREG       .EQU    ØFFDF        ; ENVIRONMENT REGISTER
AØØØ| FFEF                 BREG       .EQU    ØFFEF        ; BANK REGISTER
AØØØ|
AØØØ|                      ;----------------------------------------------------------------------
AØØØ|                      ; ENTRY POINT
AØØØ|                      ;----------------------------------------------------------------------
AØØØ|
AØØØ| 78                   ENTRY      SEI                  ; SET INTERRUPT DISABLE
AØØ1| D8                              CLD                  ; CLEAR DECIMAL FLAG
AØØ2| A9 77                           LDA     #77          ; LOAD ACCUMULATOR WITH $77
AØØ4| 8D DFFF                         STA     EREG         ; STORE IN ENVIRONMENT REGISTER
AØØ7|                                                      ; SET 2 MHZ, I/O SPACE ENABLED, SCREEN ENABLED,
AØØ7|                                                      ; RESET ENABLED, WRITE PROTECT NOT ENABLED,
AØØ7|                                                      ; PRIMARY STACK, AND ROM SELECTED
AØØ7| A2 FF                           LDX     #ØFF         ; LOAD ACCUMULATOR WITH $FF
AØØ9| 9A                              TXS                  ; TRANSFER X-REGISTER TO STACK POINTER
AØØA| 2C 1ØCØ                         BIT     KBDSTROBE    ; CLEAR KEYBOARD
AØØD| A9 4Ø                           LDA     #RETINT      ; LOAD ACCUMULATOR WITH RETURN FROM INTERRUPT
AØØF| 8D CAFF                         STA     NMIVECTOR    ; STORE IN NON-MASKABLE INTERRUPT VECTOR
AØ12| A9 Ø7                           LDA     #Ø7          ; LOAD ACCUMULATOR WITH $Ø7
AØ14| 8D EFFF                         STA     BREG         ; STORE IN BANK REGISTER
AØ17| A9 ØØ                           LDA     #ØØ          ; LOAD ACCUMULATOR WITH $ØØ
AØ19| CE EFFF              $Ø1Ø       DEC     BREG         ; DECREMENT BANK REGISTER
AØ1C| 8D ØØ2Ø                         STA     FIRSTPAGE    ; STORE IN FIRST PAGE OF BANK
AØ1F| AE ØØ2Ø                         LDX     FIRSTPAGE    ; LOAD X-REGISTER WITH FIRST PAGE BYTE
AØ22| DØF5                            BNE     $Ø1Ø         ; BRANCH IF BYTE IS NOT EQUAL TO $ØØ
AØ24|
AØ24|                      ;----------------------------------------------------------------------
AØ24|                      ; This section reads in the SOS directory.
AØ24|                      ;----------------------------------------------------------------------
AØ24|
AØ24| A9 ØØ                READSOSDIR LDA     #ØØ          ; LOAD ACCUMULATOR WITH $ØØ-BLOCK HIGH BYTE
AØ26| 85 85                           STA     IBBUFP       ; STORE IN BUFFER POINTER LOW BYTE
AØ28| A2 A2                           LDX     #ØA2         ; LOAD X-REGISTER WITH $A2
AØ2A| 86 86                           STX     IBBUFP+1     ; STORE IN BUFFER POINTER HIGH BYTE
AØ2C| A2 Ø2                           LDX     #Ø2          ; LOAD X-REGISTER WITH $Ø2-BLOCK LOW BYTE
AØ2E| A4 85                RDSOSDIRLP LDY     IBBUFP       ; LOAD Y-REGISTER WITH BUFFER POINTER LOW BYTE
AØ3Ø| 84 E3                           STY     IBBUFPTMP    ; STORE IN BUFFER POINTER TEMPORARY LOW BYTE
AØ32| A4 86                           LDY     IBBUFP+1     ; LOAD Y-REGISTER WITH BUFFER POINTER HIGH BYTE
AØ34| 84 E4                           STY     IBBUFPTMP+1  ; STORE IN BUFFER POINTER TEMPORARY HIGH BYTE
AØ36| 2Ø 1DA1                         JSR     READBLK      ; JUMP TO READ A BLOCK FROM FLOPPY DISK DRIVE
```

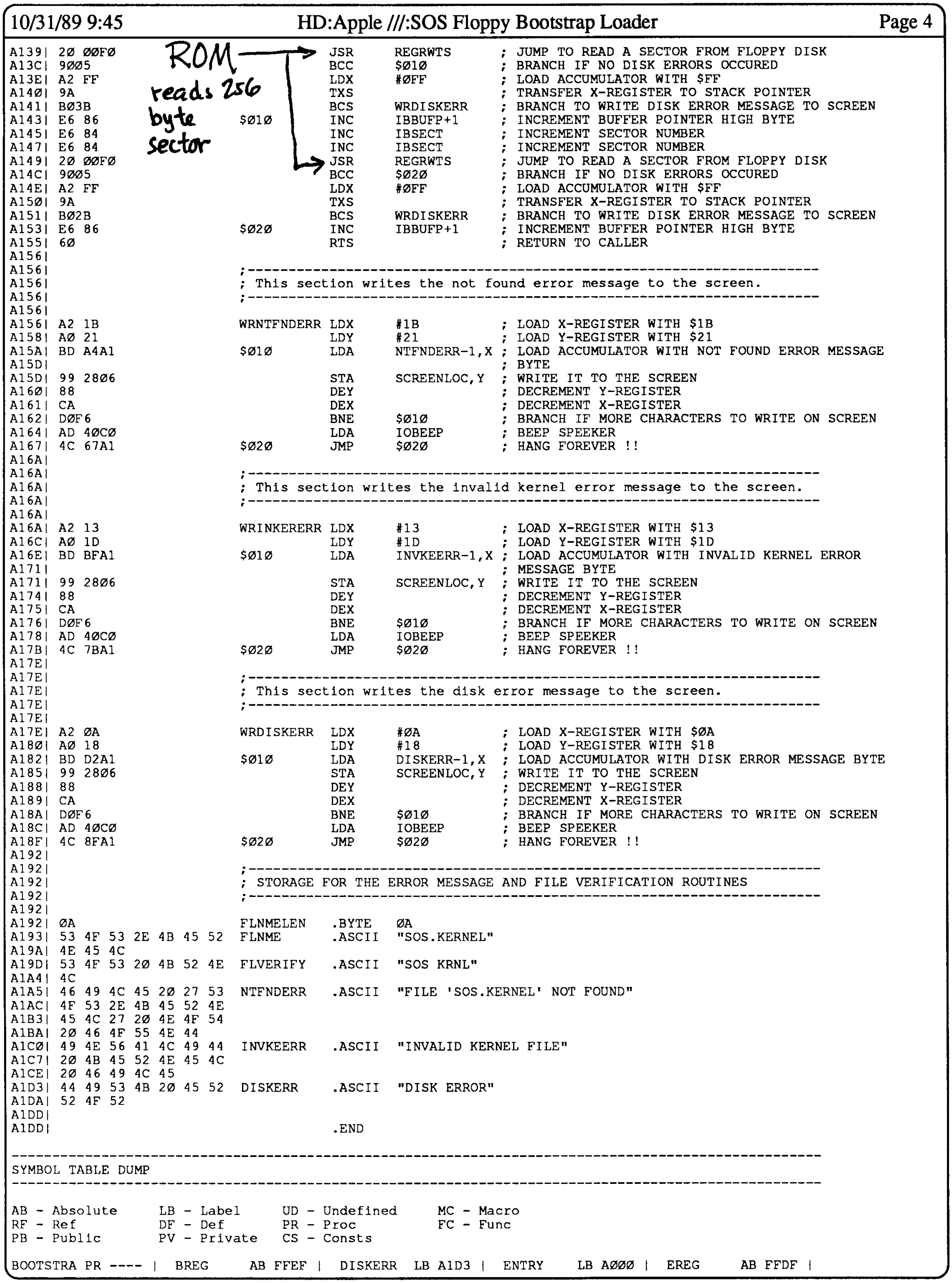


```
AØ39| AØ Ø2                             LDY     #Ø2          ; LOAD Y-REGISTER WITH $Ø2
AØ3B| B1 E3                             LDA     @IBBUFPTMP,Y ; LOAD ACCUMULATOR WITH NEXT BLOCK TO READ LOW
AØ3D|                                                        ; BYTE
AØ3D| AA                                TAX                  ; TRANSFER ACCUMULATOR TO X-REGISTER
AØ3E| C8                                INY                  ; INCREMENT Y-REGISTER
AØ3F| B1 E3                             LDA     @IBBUFPTMP,Y ; LOAD ACCUMULATOR WITH NEXT BLOCK TO READ HIGH
AØ41|                                                        ; BYTE
AØ41| DØEB                              BNE     RDSOSDIRLP   ; BRANCH IF NEXT BLOCK TO READ HIGH BYTE IS NOT
AØ43|                                                        ; EQUAL TO ZERO
AØ43| EØ ØØ                             CPX     #ØØ          ; CHECK TO SEE IF NEXT BLOCK TO READ LOW BYTE IS
AØ45|                                                        ; ZERO
AØ45| DØE7                              BNE     RDSOSDIRLP   ; BRANCH IF NEXT BLOCK TO READ LOW BYTE IS NOT
AØ47|                                                        ; EQUAL TO ZERO
AØ47|
AØ47|                       ;----------------------------------------------------------------------
AØ47|                       ; This section searches the SOS directory for the SOS.KERNEL file.
AØ47|                       ;----------------------------------------------------------------------
AØ47|
AØ47| AD 25A2               SRCHSOSKER  LDA     MAINBUFF+25  ; LOAD ACCUMULATOR WITH FILE COUNT LOW BYTE
AØ4A| 85 E5                             STA     FILECNT      ; STORE IN FILE COUNT LOW BYTE
AØ4C| AD 26A2                           LDA     MAINBUFF+26  ; LOAD ACCUMULATOR WITH FILE COUNT HIGH BYTE
AØ4F| 85 E6                             STA     FILECNT+1    ; STORE IN FILE COUNT HIGH BYTE
AØ51| Ø5 E5                             ORA     FILECNT      ; OR ACCUMULATOR WITH FILE COUNT LOW BYTE
AØ53| DØØ3                              BNE     $Ø1Ø         ; BRANCH IF FILE COUNT IS NOT EQUAL TO ZERO
AØ55| 4C 56A1                           JMP     WRNTFNDERR   ; JUMP TO WRITE NOT FOUND ERROR MESSAGE TO
AØ58|                                                        ; SCREEN
AØ58| A5 E5                 $Ø1Ø        LDA     FILECNT      ; LOAD ACCUMULATOR WITH FILE COUNT LOW BYTE
AØ5A| DØØ2                              BNE     $Ø2Ø         ; BRANCH IF NOT EQUAL TO $ØØ
AØ5C| C6 E6                             DEC     FILECNT+1    ; DECREMENT FILE COUNT HIGH BYTE
AØ5E| C6 E5                 $Ø2Ø        DEC     FILECNT      ; DECREMENT FILE COUNT LOW BYTE
AØ6Ø| A9 2B                             LDA     #2B          ; LOAD ACCUMULATOR WITH $28
AØ62| 85 85                             STA     IBBUFP       ; STORE IN BUFFER POINTER LOW BYTE
AØ64| A9 A2                             LDA     #ØA2         ; LOAD ACCUMULATOR WITH $A2
AØ66| 85 86                             STA     IBBUFP+1     ; STORE IN BUFFER POINTER HIGH BYTE
AØ68| AE 24A2                           LDX     MAINBUFF+24  ; LOAD X-REGISTER WITH ENTRIES PER BLOCK
AØ6B| CA                                DEX                  ; DECREMENT X-REGISTER
AØ6C| AØ ØØ                 SRCHLP      LDY     #ØØ          ; LOAD Y-REGISTER WITH $ØØ
AØ6E| B1 85                             LDA     @IBBUFP,Y    ; LOAD ACCUMULATOR WITH STORAGE TYPE AND NAME
AØ7Ø|                                                        ; LENGTH BYTE
AØ7Ø| FØ1A                              BEQ     $Ø2Ø         ; BRANCH IF EQUAL TO ZERO
AØ72| 29 ØF                             AND     #ØF          ; MASK OFF BITS 4,5,6,7
AØ74| CD 92A1                           CMP     FLNMELEN     ; COMPARE WITH FILE NAME LENGTH
AØ77| DØ13                              BNE     $Ø2Ø         ; BRANCH IF NOT EQUAL TO ZERO
AØ79| A8                                TAY                  ; TRANSFER NAME LENGTH TO Y-REGISTER
AØ7A| B1 85                 $Ø1Ø        LDA     @IBBUFP,Y    ; LOAD ACCUMULATOR WITH FILE NAME BYTE
AØ7C| D9 92A1                           CMP     FLNME-1,Y    ; COMPARE WITH FILE NAME BYTE
AØ7F| DØØB                              BNE     $Ø2Ø         ; BRANCH IF NOT EQUAL
AØ81| 88                                DEY                  ; DECREMENT NAME LENGTH
AØ82| DØF6                              BNE     $Ø1Ø         ; BRANCH IF NAME LENGTH NOT EQUAL TO ZERO
AØ84| B1 85                             LDA     @IBBUFP,Y    ; LOAD ACCUMULATOR WITH STORAGE TYPE AND NAME
AØ86|                                                        ; LENGTH BYTE
AØ86| 29 FØ                             AND     #ØFØ         ; MASK OFF BITS Ø,1,2,3
AØ88| C9 2Ø                             CMP     #2Ø          ; COMPARE WITH $2Ø FOR SAPLING FILE
AØ8A| FØ32                              BEQ     READIDXBLK   ; BRANCH IF EQUAL TO READ INDEX BLOCK
AØ8C| Ø8                    $Ø2Ø        PHP                  ; PUSH PROCESSOR STATUS ON STACK
AØ8D| CA                                DEX                  ; DECREMENT ENTRIES PER BLOCK
AØ8E| FØ1Ø                              BEQ     $Ø3Ø         ; BRANCH IF ENTRIES PER BLOCK IS EQUAL TO ZERO
AØ9Ø| 18                                CLC                  ; CLEAR CARRY
AØ91| A5 85                             LDA     IBBUFP       ; LOAD ACCUMULATOR WITH BUFFER POINTER LOW BYTE
AØ93| 6D 23A2                           ADC     MAINBUFF+23  ; ADD ENTRY LENGTH LOW BYTE
AØ96| 85 85                             STA     IBBUFP       ; STORE IN BUFFER POINTER LOW BYTE
AØ98| A5 86                             LDA     IBBUFP+1     ; LOAD ACCUMULATOR WITH BUFFER POINTER HIGH BYTE
AØ9A| 69 ØØ                             ADC     #ØØ          ; ADD $ØØ
AØ9C| 85 86                             STA     IBBUFP+1     ; STORE IN BUFFER POINTER HIGH BYTE
AØ9E| DØØ9                              BNE     $Ø4Ø         ; BRANCH ALWAYS
AØAØ| A9 Ø4                 $Ø3Ø        LDA     #Ø4          ; LOAD ACCUMULATOR WITH $Ø4
AØA2| 85 85                             STA     IBBUFP       ; STORE IN BUFFER POINTER LOW BYTE
AØA4| E6 86                             INC     IBBUFP+1     ; INCREMENT BUFFER POINTER HIGH BYTE
AØA6| AE 24A2                           LDX     MAINBUFF+24  ; LOAD X-REGISTER WITH ENTRIES PER BLOCK
AØA9| 28                    $Ø4Ø        PLP                  ; PULL PROCESSOR STATUS FROM STACK
AØAA| FØCØ                              BEQ     SRCHLP       ; BRANCH IF NOT EQUAL TO ZERO
AØAC| 38                                SEC                  ; SET CARRY
AØAD| A5 E5                             LDA     FILECNT      ; LOAD ACCUMULATOR WITH FILE COUNT LOW BYTE
AØAF| E9 Ø1                             SBC     #Ø1          ; SUBTRACT $Ø1
AØB1| 85 E5                             STA     FILECNT      ; STORE IN FILE COUNT LOW BYTE
AØB3| A5 E6                             LDA     FILECNT+1    ; LOAD ACCUMULATOR WITH FILE COUNT HIGH BYTE
AØB5| E9 ØØ                             SBC     #ØØ          ; SUBTRACT $ØØ
AØB7| 85 E6                             STA     FILECNT+1    ; STORE IN FILE COUNT HIGH BYTE
AØB9| BØB1                              BCS     SRCHLP       ; BRANCH IF MORE FILE ENTRIES
AØBB| 4C 56A1                           JMP     WRNTFNDERR   ; JUMP TO WRITE NOT FOUND ERROR MESSAGE TO
AØBE|                                                        ; SCREEN
AØBE|
AØBE|                       ;----------------------------------------------------------------------
AØBE|                       ; This section reads in the index block of the SOS.KERNEL file.
AØBE|                       ;----------------------------------------------------------------------
AØBE|
AØBE| AØ 11                 READIDXBLK  LDY     #11          ; LOAD Y-REGISTER WITH $11
AØCØ| B1 85                             LDA     @IBBUFP,Y    ; LOAD KEY POINTER LOW BYTE
AØC2| AA                                TAX                  ; TRANSFER ACCUMULATOR TO X-REGISTER-BLOCK LOW
AØC3|                                                        ; BYTE
```

“APPLE_PAT_4_383_296_G_04” 265 KB 2000-02-29 dpi: 300h x 300v pix: 2305h x 3095v

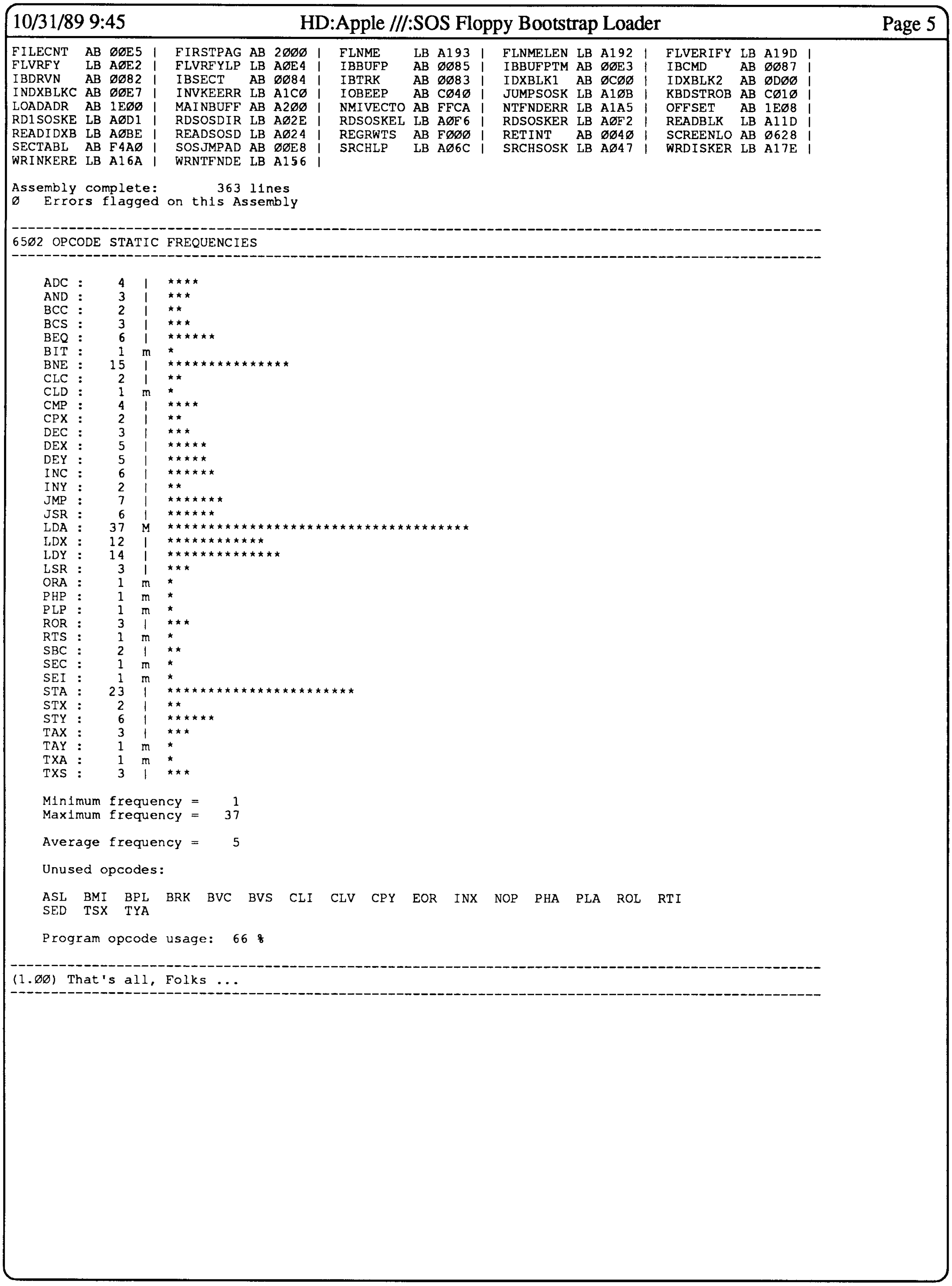


```
AØC3| C8                               INY                ; INCREMENT Y-REGISTER
AØC4| B1 85                            LDA   @IBBUFP,Y    ; LOAD KEY POINTER HIGH BYTE
AØC6| AØ ØØ                            LDY   #ØØ          ; LOAD Y-REGISTER WITH $ØØ
AØC8| 84 85                            STY   IBBUFP       ; STORE IN BUFFER POINTER LOW BYTE
AØCA| AØ ØC                            LDY   #ØC          ; LOAD Y-REGISTER WITH $ØC
AØCC| 84 86                            STY   IBBUFP+1     ; STORE IN BUFFER POINTER HIGH BYTE
AØCE| 2Ø 1DA1                          JSR   READBLK      ; JUMP TO READ A BLOCK FROM FLOPPY DISK DRIVE
AØD1|
AØD1|                       ;----------------------------------------------------------------------
AØD1|                       ; This section reads in the first block of the SOS.KERNEL file.
AØD1|                       ;----------------------------------------------------------------------
AØD1|
AØD1| AE ØØØC               RD1SOSKER  LDX   IDXBLK1      ; LOAD X-REGISTER WITH INDEX BLOCK LOW BYTE
AØD4| AD ØØØD                          LDA   IDXBLK2      ; LOAD ACCUMULATOR WITH INDEX BLOCK HIGH BYTE
AØD7| AØ ØØ                            LDY   #ØØ          ; LOAD Y-REGISTER WITH $ØØ
AØD9| 84 85                            STY   IBBUFP       ; STORE IN BUFFER POINTER LOW BYTE
AØDB| AØ 1E                            LDY   #1E          ; LOAD Y-REGISTER WITH $1E
AØDD| 84 86                            STY   IBBUFP+1     ; STORE IN BUFFER POINTER HIGH BYTE
AØDF| 2Ø 1DA1                          JSR   READBLK      ; JUMP TO READ A BLOCK FROM FLOPPY DISK DRIVE
AØE2|
AØE2|                       ;----------------------------------------------------------------------
AØE2|                       ; This section does a verification of the SOS.KERNEL file to make
AØE2|                       ; sure it is the proper SOS.KERNEL file. It checks for "SOS KRNL" in
AØE2|                       ; the first 8 bytes of the file.
AØE2|                       ;----------------------------------------------------------------------
AØE2|
AØE2| AØ Ø8                 FLVRFY     LDY   #Ø8          ; LOAD Y-REGISTER WITH $Ø8
AØE4| B9 FF1D               FLVRFYLP   LDA   LOADADR-1,Y  ; LOAD ACCUMULATOR WITH BYTE FROM SOS.KERNEL
AØE7| D9 9CA1                          CMP   FLVERIFY-1,Y ; COMPARE WITH VERIFICATION BYTE
AØEA| FØØ3                             BEQ   $Ø1Ø         ; BRANCH IF EQUAL
AØEC| 4C 6AA1                          JMP   WRINKERERR   ; JUMP TO WRITE INVALID KERNEL ERROR MESSAGE TO
AØEF|                                                     ; SCREEN
AØEF| 88                    $Ø1Ø       DEY                ; DECREMENT Y-REGISTER
AØFØ| DØF2                             BNE   FLVRFYLP     ; BRANCH IF NOT EQUAL TO ZERO TO CHECK REST OF 8
AØF2|                                                     ; SOS.KERNEL BYTES
AØF2|
AØF2|                       ;----------------------------------------------------------------------
AØF2|                       ; This section reads in the SOS.KERNEL file.
AØF2|                       ;----------------------------------------------------------------------
AØF2|
AØF2| A9 Ø1                 RDSOSKER   LDA   #Ø1          ; LOAD ACCUMULATOR WITH $Ø1
AØF4| 85 E7                            STA   INDXBLKCNT   ; STORE IN INDEX BLOCK COUNT
AØF6| A4 E7                 RDSOSKELP  LDY   INDXBLKCNT   ; LOAD Y-REGISTER WITH INDEX BLOCK COUNT
AØF8| BE ØØØC                          LDX   IDXBLK1,Y    ; LOAD X-REGISTER WITH BLOCK LOW BYTE
AØFB| B9 ØØØD                          LDA   IDXBLK2,Y    ; LOAD ACCUMULATOR WITH BLOCK HIGH BYTE
AØFE| DØØ4                             BNE   $Ø1Ø         ; BRANCH IF BLOCK HIGH BYTE IS NOT EQUAL TO ZERO
A1ØØ| EØ ØØ                            CPX   #ØØ          ; CHECK TO SEE IF BLOCK LOW BYTE IS NOT EQUAL TO
A1Ø2|                                                     ; ZERO
A1Ø2| FØØ7                             BEQ   JUMPSOSKER   ; BRANCH IF BLOCK LOW BYTE IS NOT EQUAL TO ZERO
A1Ø4| 2Ø 1DA1               $Ø1Ø       JSR   READBLK      ; JUMP TO READ A BLOCK FROM FLOPPY DISK DRIVE
A1Ø7| E6 E7                            INC   INDXBLKCNT   ; INCREMENT INDEX BLOCK COUNT
A1Ø9| DØEB                             BNE   RDSOSKELP    ; BRANCH IF INDEX BLOCK COUNT IS NOT EQUAL TO
A1ØB|                                                     ; ZERO TO READ MORE OF THE SOS.KERNEL
A1ØB|
A1ØB|                       ;----------------------------------------------------------------------
A1ØB|                       ; This section jumps to the SOS.KERNEL loader.
A1ØB|                       ;----------------------------------------------------------------------
A1ØB|
A1ØB| 18                    JUMPSOSKER CLC                ; CLEAR CARRY
A1ØC| A9 ØE                            LDA   #ØE          ; LOAD ACCUMULATOR WITH $ØE
A1ØE| 6D Ø81E                          ADC   OFFSET       ; ADD OFFSET LOW BYTE
A111| 85 E8                            STA   SOSJMPADR    ; STORE IN SOS JUMP ADDRESS LOW BYTE
A113| A9 1E                            LDA   #1E          ; LOAD ACCUMULATOR WITH $1E
A115| 6D Ø91E                          ADC   OFFSET+1     ; ADD OFFSET HIGH BYTE
A118| 85 E9                            STA   SOSJMPADR+1  ; STORE IN SOS JUMP ADDRESS HIGH BYTE
A11A| 6C E8ØØ                          JMP   @SOSJMPADR   ; JUMP TO SOS.KERNEL LOADER
A11D|
A11D|                       ;----------------------------------------------------------------------
A11D|                       ; This section reads a block of data from the floppy disk drive.
A11D|                       ; On entry the x-register contains the block low byte and the
A11D|                       ; accumulator contains the block high byte.
A11D|                       ;----------------------------------------------------------------------
A11D|
A11D| 86 83                 READBLK    STX   IBTRK        ; STORE BLOCK LOW BYTE IN TRACK NUMBER
A11F| 4A                               LSR   A            ; DIVIDE BLOCK BY 8 TO GET TRACK NUMBER
A12Ø| 66 83                            ROR   IBTRK
A122| 4A                               LSR   A
A123| 66 83                            ROR   IBTRK
A125| 4A                               LSR   A
A126| 66 83                            ROR   IBTRK
A128| 8A                               TXA                ; TRANSFER X-REGISTER WHICH CONTAINS THE BLOCK
A129|                                                     ; LOW BYTE TO ACCUMULATOR
A129| 29 Ø7                            AND   #Ø7          ; MASK OFF BITS 3,4,5,6,7
A12B| AA                               TAX                ; TRANSFER ACCUMULATOR TO X-REGISTER
A12C| BD AØF4                          LDA   SECTABL,X    ; LOAD ACCUMULATOR WITH PROPER SECTOR TO READ
A12F| 85 84                            STA   IBSECT       ; STORE IN SECTOR NUMBER
A131| A9 Ø1                            LDA   #Ø1          ; LOAD ACCUMULATOR WITH $Ø1
A133| 85 87                            STA   IBCMD        ; STORE IN COMMAND NUMBER
A135| A9 ØØ                            LDA   #ØØ          ; LOAD ACCUMULATOR WITH $ØØ
A137| 85 82                            STA   IBDRVN       ; STORE IN DRIVE NUMBER
```



ROM
reads 256
byte
sector

```
A139| 2Ø ØØFØ                 JSR    REGRWTS       ; JUMP TO READ A SECTOR FROM FLOPPY DISK
A13C| 9ØØ5                    BCC    $Ø1Ø          ; BRANCH IF NO DISK ERRORS OCCURED
A13E| A2 FF                   LDX    #ØFF          ; LOAD ACCUMULATOR WITH $FF
A14Ø| 9A                      TXS                  ; TRANSFER X-REGISTER TO STACK POINTER
A141| BØ3B                    BCS    WRDISKERR     ; BRANCH TO WRITE DISK ERROR MESSAGE TO SCREEN
A143| E6 86          $Ø1Ø     INC    IBBUFP+1      ; INCREMENT BUFFER POINTER HIGH BYTE
A145| E6 84                   INC    IBSECT        ; INCREMENT SECTOR NUMBER
A147| E6 84                   INC    IBSECT        ; INCREMENT SECTOR NUMBER
A149| 2Ø ØØFØ                 JSR    REGRWTS       ; JUMP TO READ A SECTOR FROM FLOPPY DISK
A14C| 9ØØ5                    BCC    $Ø2Ø          ; BRANCH IF NO DISK ERRORS OCCURED
A14E| A2 FF                   LDX    #ØFF          ; LOAD ACCUMULATOR WITH $FF
A15Ø| 9A                      TXS                  ; TRANSFER X-REGISTER TO STACK POINTER
A151| BØ2B                    BCS    WRDISKERR     ; BRANCH TO WRITE DISK ERROR MESSAGE TO SCREEN
A153| E6 86          $Ø2Ø     INC    IBBUFP+1      ; INCREMENT BUFFER POINTER HIGH BYTE
A155| 6Ø                      RTS                  ; RETURN TO CALLER
A156|
A156|                ;---------------------------------------------------------------
A156|                ; This section writes the not found error message to the screen.
A156|                ;---------------------------------------------------------------
A156|
A156| A2 1B          WRNTFNDERR LDX  #1B           ; LOAD X-REGISTER WITH $1B
A158| AØ 21                   LDY    #21           ; LOAD Y-REGISTER WITH $21
A15A| BD A4A1        $Ø1Ø     LDA    NTFNDERR-1,X  ; LOAD ACCUMULATOR WITH NOT FOUND ERROR MESSAGE
A15D|                                              ; BYTE
A15D| 99 28Ø6                 STA    SCREENLOC,Y   ; WRITE IT TO THE SCREEN
A16Ø| 88                      DEY                  ; DECREMENT Y-REGISTER
A161| CA                      DEX                  ; DECREMENT X-REGISTER
A162| DØF6                    BNE    $Ø1Ø          ; BRANCH IF MORE CHARACTERS TO WRITE ON SCREEN
A164| AD 4ØCØ                 LDA    IOBEEP        ; BEEP SPEEKER
A167| 4C 67A1        $Ø2Ø     JMP    $Ø2Ø          ; HANG FOREVER !!
A16A|
A16A|                ;---------------------------------------------------------------
A16A|                ; This section writes the invalid kernel error message to the screen.
A16A|                ;---------------------------------------------------------------
A16A|
A16A| A2 13          WRINKERERR LDX  #13           ; LOAD X-REGISTER WITH $13
A16C| AØ 1D                   LDY    #1D           ; LOAD Y-REGISTER WITH $1D
A16E| BD BFA1        $Ø1Ø     LDA    INVKEERR-1,X  ; LOAD ACCUMULATOR WITH INVALID KERNEL ERROR
A171|                                              ; MESSAGE BYTE
A171| 99 28Ø6                 STA    SCREENLOC,Y   ; WRITE IT TO THE SCREEN
A174| 88                      DEY                  ; DECREMENT Y-REGISTER
A175| CA                      DEX                  ; DECREMENT X-REGISTER
A176| DØF6                    BNE    $Ø1Ø          ; BRANCH IF MORE CHARACTERS TO WRITE ON SCREEN
A178| AD 4ØCØ                 LDA    IOBEEP        ; BEEP SPEEKER
A17B| 4C 7BA1        $Ø2Ø     JMP    $Ø2Ø          ; HANG FOREVER !!
A17E|
A17E|                ;---------------------------------------------------------------
A17E|                ; This section writes the disk error message to the screen.
A17E|                ;---------------------------------------------------------------
A17E|
A17E| A2 ØA          WRDISKERR LDX   #ØA           ; LOAD X-REGISTER WITH $ØA
A18Ø| AØ 18                   LDY    #18           ; LOAD Y-REGISTER WITH $18
A182| BD D2A1        $Ø1Ø     LDA    DISKERR-1,X   ; LOAD ACCUMULATOR WITH DISK ERROR MESSAGE BYTE
A185| 99 28Ø6                 STA    SCREENLOC,Y   ; WRITE IT TO THE SCREEN
A188| 88                      DEY                  ; DECREMENT Y-REGISTER
A189| CA                      DEX                  ; DECREMENT X-REGISTER
A18A| DØF6                    BNE    $Ø1Ø          ; BRANCH IF MORE CHARACTERS TO WRITE ON SCREEN
A18C| AD 4ØCØ                 LDA    IOBEEP        ; BEEP SPEEKER
A18F| 4C 8FA1        $Ø2Ø     JMP    $Ø2Ø          ; HANG FOREVER !!
A192|
A192|                ;---------------------------------------------------------------
A192|                ; STORAGE FOR THE ERROR MESSAGE AND FILE VERIFICATION ROUTINES
A192|                ;---------------------------------------------------------------
A192|
A192| ØA                   FLNMELEN   .BYTE  ØA
A193| 53 4F 53 2E 4B 45 52 FLNME      .ASCII "SOS.KERNEL"
A19A| 4E 45 4C
A19D| 53 4F 53 2Ø 4B 52 4E FLVERIFY   .ASCII "SOS KRNL"
A1A4| 4C
A1A5| 46 49 4C 45 2Ø 27 53 NTFNDERR   .ASCII "FILE 'SOS.KERNEL' NOT FOUND"
A1AC| 4F 53 2E 4B 45 52 4E
A1B3| 45 4C 27 2Ø 4E 4F 54
A1BA| 2Ø 46 4F 55 4E 44
A1CØ| 49 4E 56 41 4C 49 44 INVKEERR   .ASCII "INVALID KERNEL FILE"
A1C7| 2Ø 4B 45 52 4E 45 4C
A1CE| 2Ø 46 49 4C 45
A1D3| 44 49 53 4B 2Ø 45 52 DISKERR    .ASCII "DISK ERROR"
A1DA| 52 4F 52
A1DD|
A1DD|                           .END

----------------------------------------------------------------------------------------
SYMBOL TABLE DUMP
----------------------------------------------------------------------------------------

AB - Absolute      LB - Label      UD - Undefined    MC - Macro
RF - Ref           DF - Def        PR - Proc         FC - Func
PB - Public        PV - Private    CS - Consts

BOOTSTRA PR ---- |  BREG     AB FFEF |  DISKERR  LB A1D3 |  ENTRY    LB AØØØ |  EREG     AB FFDF |
```



10/31/89 9:45 HD:Apple ///:SOS Floppy Bootstrap Loader Page 5

```
FILECNT  AB ØØE5 | FIRSTPAG AB 2ØØØ | FLNME    LB A193 | FLNMELEN LB A192 | FLVERIFY LB A19D |
FLVRFY   LB AØE2 | FLVRFYLP LB AØE4 | IBBUFP   AB ØØ85 | IBBUFPTM AB ØØE3 | IBCMD    AB ØØ87 |
IBDRVN   AB ØØ82 | IBSECT   AB ØØ84 | IBTRK    AB ØØ83 | IDXBLK1  AB ØCØØ | IDXBLK2  AB ØDØØ |
INDXBLKC AB ØØE7 | INVKEERR LB A1CØ | IOBEEP   AB CØ4Ø | JUMPSOSK LB A1ØB | KBDSTROB AB CØ1Ø |
LOADADR  AB 1EØØ | MAINBUFF AB A2ØØ | NMIVECTO AB FFCA | NTFNDERR LB A1A5 | OFFSET   AB 1EØ8 |
RD1SOSKE LB AØD1 | RDSOSDIR LB AØ2E | RDSOSKEL LB AØF6 | RDSOSKER LB AØF2 | READBLK  LB A11D |
READIDXB LB AØBE | READSOSD LB AØ24 | REGRWTS  AB FØØØ | RETINT   AB ØØ4Ø | SCREENLO AB Ø628 |
SECTABL  AB F4AØ | SOSJMPAD AB ØØE8 | SRCHLP   LB AØ6C | SRCHSOSK LB AØ47 | WRDISKER LB A17E |
WRINKERE LB A16A | WRNTFNDE LB A156 |

Assembly complete:       363 lines
Ø   Errors flagged on this Assembly

----------------------------------------------------------------------------------------------------
65Ø2 OPCODE STATIC FREQUENCIES
----------------------------------------------------------------------------------------------------

    ADC :     4  |  ****
    AND :     3  |  ***
    BCC :     2  |  **
    BCS :     3  |  ***
    BEQ :     6  |  ******
    BIT :     1  m  *
    BNE :    15  |  ***************
    CLC :     2  |  **
    CLD :     1  m  *
    CMP :     4  |  ****
    CPX :     2  |  **
    DEC :     3  |  ***
    DEX :     5  |  *****
    DEY :     5  |  *****
    INC :     6  |  ******
    INY :     2  |  **
    JMP :     7  |  *******
    JSR :     6  |  ******
    LDA :    37  M  *************************************
    LDX :    12  |  ************
    LDY :    14  |  **************
    LSR :     3  |  ***
    ORA :     1  m  *
    PHP :     1  m  *
    PLP :     1  m  *
    ROR :     3  |  ***
    RTS :     1  m  *
    SBC :     2  |  **
    SEC :     1  m  *
    SEI :     1  m  *
    STA :    23  |  ***********************
    STX :     2  |  **
    STY :     6  |  ******
    TAX :     3  |  ***
    TAY :     1  m  *
    TXA :     1  m  *
    TXS :     3  |  ***

    Minimum frequency =    1
    Maximum frequency =   37

    Average frequency =    5

    Unused opcodes:

    ASL  BMI  BPL  BRK  BVC  BVS  CLI  CLV  CPY  EOR  INX  NOP  PHA  PLA  ROL  RTI
    SED  TSX  TYA

    Program opcode usage:  66 %

----------------------------------------------------------------------------------------------------
(1.ØØ) That's all, Folks ...
----------------------------------------------------------------------------------------------------
```

seems like an
early version

 Apple /// Computer Information

APPLE /// SOS BOOTSTRAP LOADER HEXADECIMAL DUMP

Source
DISK1.dofile as found with Chris Smolinski's Macintosh SARA emulator application

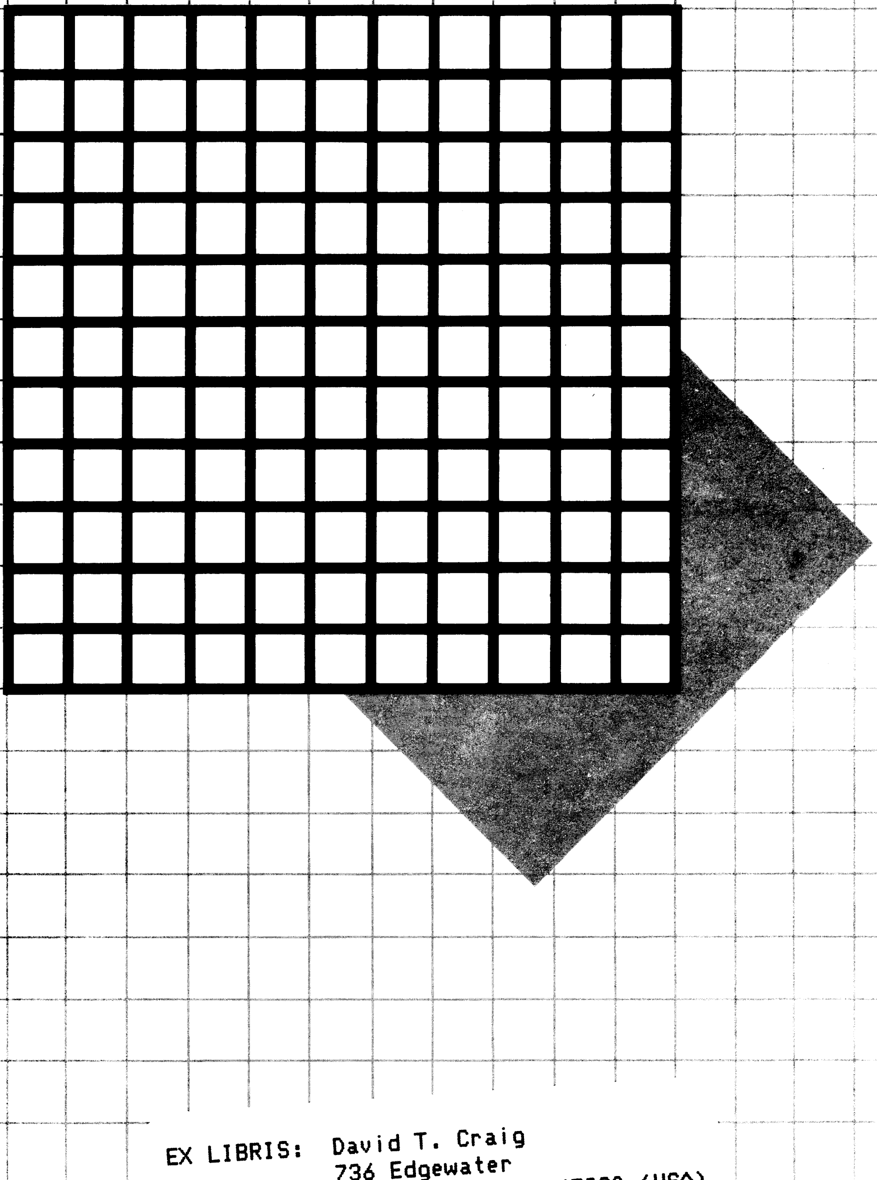
Printed by David T. Craig • December 1997

This hex dump, which was produced by the Apple Macintosh MPW DumpFile tool, lists the Apple /// SOS Bootstrap Loader. This 512 byte loader exists at block 0 of SOS disks and is loaded by the Apple /// ROM into memory addresses $A000-$A1FF. This code's purpose is to begin the loading of SOS from the floppy disk into the ///'s memory.

```
  0: 4C 6E A0 53 4F 53 20 42 4F 4F 54 20 20 31 2E 31 Ln†SOS.BOOT..1.1
 10: 20 0A 53 4F 53 2E 4B 45 52 4E 45 4C 20 20 20 20 ..SOS.KERNEL....
 20: 20 53 4F 53 20 4B 52 4E 4C 49 2F 4F 20 45 52 52 .SOS.KRNLI/O.ERR
 30: 4F 52 08 00 46 49 4C 45 20 27 53 4F 53 2E 4B 45 OR..FILE.'SOS.KE
 40: 52 4E 45 4C 27 20 4E 4F 54 20 46 4F 55 4E 44 25 RNEL'.NOT.FOUND%
 50: 00 49 4E 56 41 4C 49 44 20 4B 45 52 4E 45 4C 20 .INVALID.KERNEL.
 60: 46 49 4C 45 3A 00 00 0C 00 1E 0E 1E 04 A4 78 D8 FILE:........§xÿ
 70: A9 77 8D DF FF A2 FB 9A 2C 10 C0 A9 40 8D CA FF ©wçflˇ¢˚ö,.¿©@ç ˇ
 80: A9 07 8D EF FF A2 00 CE EF FF 8E 00 20 AD 00 20 ©.çÔˇ¢.ŒÔˇé..≠..
 90: D0 F5 A9 01 85 E0 A9 00 85 E1 A9 00 85 85 A9 A2 –ı©.Ö‡©.Ö·©.ÖÖ©¢
 A0: 85 86 20 BE A1 E6 E0 A9 00 85 E6 E6 86 E6 86 E6 ÖÜ.æ°Ê‡©.ÖÊÊÜÊÜÊ
 B0: E6 20 BE A1 A0 02 B1 85 85 E0 C8 B1 85 85 E1 D0 Ê.æ°†.±ÖÖ‡»±ÖÖ·–
 C0: EA A5 E0 D0 E6 AD 6C A0 85 E2 AD 6D A0 85 E3 18 Í•‡–Ê≠l†Ö‚≠m†Ö„.
 D0: A5 E3 69 02 85 E5 38 A5 E2 ED 23 A4 85 E4 A5 E5 •„i.ÖÂ8•‚Ì#§Ö‰•Â
 E0: E9 00 85 E5 A0 00 B1 E2 29 0F CD 11 A0 D0 21 A8 È.ÖÂ†.±‚).Õ.†–!®
 F0: B1 E2 D9 11 A0 D0 19 88 D0 F6 A0 00 B1 E2 29 F0 ±‚Ÿ.†–.à–ˆ†.±‚)
100: 53 4F 53 20 4B 52 4E 4C 62 00 01 00 0E 2E 44 31 SOS.KRNLb.....D1
110: 2F 53 4F 53 2E 49 4E 54 45 52 50 AA A5 A0 F9 A0 /SOS.INTERP™•†˘†
120: A0 A5 A0 A0 A5 A0 A0 C5 A0 A0 98 A0 F0 A1 A0 CC †•††•††≈††ò†°†Ã
130: A0 A0 C5 A0 A0 A0 A0 A0 EE A0 A0 C4 0E 2E 44 31 ††≈†††††Ó††ƒ..D1
140: 2F 53 4F 53 2E 44 52 49 56 45 52 FF 9A A0 FF 9A /SOS.DRIVERˇö†ˇö
150: A0 A0 A0 A0 D0 A0 A0 C1 A0 A0 8A A0 A0 F9 A0 C1 ††††–††¡††ä††˘†¡
160: E9 A0 9E A1 A0 F5 A0 A0 A5 A0 A0 88 00 00 88 0C È†û°†ı††•††à..à.
170: A9 00 AA 9D 00 1A 9D 00 16 9D 00 1B 9D 00 18 9D ©.™ù..ù..ù..ù..ù
180: 00 14 9D 00 01 CA D0 EB A9 30 8D DF FF A2 FB 9A ..ù.. –Î©0çflˇ¢˚ö
190: A9 1A 8D D0 FF 20 D4 1F AD DF FF 29 10 09 28 8D ©.ç–ˇ.‘.≠flˇ)..(ç
1A0: DF FF A2 FF 9A A9 1A 8D D0 FF AD 01 19 8D EF FF flˇ¢ˇö©.ç–ˇ≠..çÔˇ
1B0: 6C 02 00 AA AD EF FF 48 8E EF FF A5 27 05 26 F0 l..™≠ÔˇHéÔˇ•'.&
1C0: 33 A5 26 D0 02 C6 27 C6 26 18 A5 23 65 27 85 23 3•&–.∆'∆&.•#e'Ö#
1D0: A5 25 65 27 85 25 E6 27 A4 26 F0 07 B1 22 91 24 •%e'Ö%Ê'§&.±"ë$
1E0: 88 D0 F9 B1 22 91 24 88 C6 23 C6 25 C6 27 D0 EC à–˘±"ë$à∆#∆%∆'–Ï
1F0: E6 23 E6 25 68 8D EF FF 60 18 A5 24 65 10 85 10 Ê#Ê%hçÔˇ`.•$e.Ö.
```

###

Apple /// System Data Sheet



The Apple *///*

The Most Powerful Personal Computer In Its Class

Too much information? Not enough time? The Apple *///* was created to meet the information-handling needs of decision makers at all levels, in every size and kind of company. And the Apple *///* can grow with you, so as your responsibilities increase, your ability to handle them stays one step ahead.

You can use the power of your Apple *///* to create financial forecasts, budgets, and reports; for accounting, resource management, and project scheduling; in electronic communications, software development, and computer-assisted training. Over 400 business programs are available today for the Apple *///* — plus the extensive library of CP/M® business software (with the Apple SoftCard™ *///*). And most Apple *//* Plus programs will run in the Apple *///*'s "emulation" mode.

The Apple *///*: the personal computer for business.

Powerful features for professional needs.

The Apple *///* is ready to go as soon as you unpack it, connect a monitor, and provide power. No interface cards are required, and you don't have to open the computer. The Apple *///* already has a built-in disk drive, video outputs for color and monochromatic displays, and a numeric keypad.

Other built-in features include:

Large User Memory. The Apple *///*'s 256K of internal memory means you can work with sophisticated programs and large financial and text documents, quickly and efficiently.

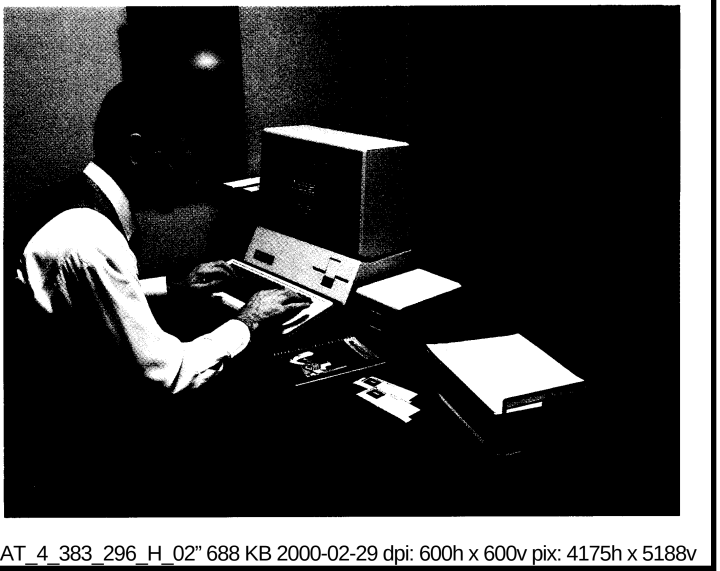
Color Graphics. The 16-color graphics capability of the Apple *///* allows you to grasp the meaning of charts and graphs quickly. If you're not using a color monitor, your information is displayed in 16 shades, so the facts still stand out clearly.

High-Resolution Video. The Apple *///* displays 107, 520 points of information on the screen (560 horizontal x 192 vertical) in text and monochromatic graphics modes. While text is normally presented in an 80-column by 24-line monochromatic format, it can be switched to 40-column monochromatic or color-on-color.

Accessory Connectors. The most common accessories plug right into the Apple *///*. Connectors and interfacing hardware are already built in for the Apple Daisy Wheel Printer (or other serial printer), the Apple Silentype Printer, external floppy disk drives, color and monochromatic video displays (NTSC, RGB, and composite), a modem, and hand controls. The Apple *///* also has four inside expansion slots for additional accessories.

Apple *///* Sophisticated Operating System: it does it all for you.

Today . . . you can bring financial models into reports, insert names into form letters automatically, and turn numbers into charts, because the Apple *///*'s Sophisticated Operating System (SOS) treats all your files identically. And, since applications programs written for the Apple *///* are all based on this common SOS formatting, you can combine them on a ProFile™ mass storage system and move freely from one to another. The uniformity of SOS also provides an ideal environment for software development.



"APPLE_PAT_4_383_296_H_02" 688 KB 2000-02-29 dpi: 600h x 600v pix: 4175h x 5188v

Tomorrow . . . you can expand your Apple *///* elegantly. Because SOS controls all communications with accessories, you don't have to figure out how to make the computer work with a new printer, disk drive, or modem. SOS does this for you by using special files known as "device drivers." Apple *///* programs come with the most commonly-used device drivers, and you can make programs compatible with new equipment by copying a driver file for the new device onto a program disk. Your software can just as easily be revised to take advantage of SOS upgrades, and of hardware enhancements to the computer itself.

Installation's easy. Learning is, too.

Because the Apple *///* already has a built-in disk drive and video connector, the computer is ready to work as soon as you connect a monitor and provide power. Then, Apple makes it just as easy to learn how to use it. A comprehensive Owner's Guide gets you started, and a System Demonstration disk introduces you to the computer's text editing and graphics capabilities. Reference manuals and SOS utilities disks are included for more advanced needs, and additional tutorials on the computer and its programs are also available.

Durable. Dependable. Reliable.

The Apple *///* is dependable, inside and out. Outside, it has a rugged die-cast aluminum chassis. Inside, electronics based on advanced microprocessor circuitry assure reliable operation. The system also meets UL and CSA standards.

Every time the computer is powered up, it performs a brief self-diagnostic routine. Should problems arise, help is close at hand, because of Apple's extensive dealer/service network. Average turnaround time on Apple *///* servicing is less than one day.

Standard Features

- 256K internal memory (RAM)
- Built-in disk drive
- Custom microprocessor circuitry
- High-resolution color graphics (16 colors)
- 80-column, 24-line text display, upper and lower case
- Contoured typewriter-style keyboard; 61 keys; all 128 ASCII codes; auto-repeat on all keys
- Numeric keypad (13 keys)
- Special-purpose keys: Up-Arrow, Down-Arrow, Left-Arrow, Right-Arrow; programmable Open-Apple and Solid-Apple; TAB; SHIFT; ALPHA LOCK; CONTROL; RETURN; ENTER; ESCAPE
- Quick-connect plugs for disk drives, video and audio devices, serial printers, modems, and hand controls
- Four expansion slots for accessory interface cards
- Apple *//* Plus emulation mode
- High-quality sound generation
- Lockable case
- Self-testing diagnostics on powerup

Optional Accessories

- Monitor *///* or color monitor
- Apple Daisy Wheel Printer
- Apple Dot Matrix Printer
- Apple Silentype Printer
- Disk *///* floppy-disk drives
- ProFile hard-disk systems
- Apple SoftCard *///* System (for CP/M capability)
- Parallel Card *///*
- Serial Card *///*
- Programming languages (Business BASIC, Pascal, COBOL)
- Cursor *///* joysticks

"APPLE_PAT_4_383_296_H_03" 671 KB 2000-02-29 dpi: 600h x 600v pix: 4255h x 4342v

Technical Specifications

■ *Video Display:*
Text and graphics may be displayed simultaneously. Graphics modes:
—280 x 192, 16 colors (with some limitations);
—280 x 192, monochromatic;
—140 x 192, 16 colors;
—560 x 192, monochromatic;
—All Apple *II* modes (in emulation)
Graphics commands allow either of two screen buffers to be displayed.
Text modes:
—80-column, 24-line monochromatic;
—40-column, 24-line, 16-color foreground and background;
—40-column, 24-line monochromatic.
All text modes have a software-definable, 128-character set (upper- and lower-case), with normal or inverse display.

■ *Central Processing Unit (CPU):*
The custom-designed microprocessor circuitry of the Apple *III* utilizes the 6502B as one of its major components. Other circuitry provides extended addressing capability, relocatable stack, zero page, and memory mapping.
Type:
6502B.
Clock Speed:
1.4 MHz average; 1.8 MHz maximum.
Operations Per Second (8-bit):
Up to 750,000.
Data Bus:
Two 8-bit formats, combined for extended addressing.
Address Bus:
19 bits.
Address Range:
262,144 bytes (256K).
Registers:
Accumulator (A); Index Registers (X,Y); Stack Pointer (S); Program Counter (PC); Environmental Register (E); Bank (B); Zero Page (Z); Processor Status (P).

■ *Memory:*
256K dynamic RAM;
4K ROM (initialization and self-test diagnostics).

■ *SOS (Sophisticated Operating System):*
Handles all system I/O;
Can be configured to handle standard or custom I/O devices and peripherals by adding or deleting "device drivers";
All languages and application programs access data through the SOS file system.

■ *Inputs and Outputs:*
Keyboard:
—61 keys on main keyboard;
—13 keys on numeric keypad;
—Full 128-character, ASCII encoded;
—All keys have automatic repeat;
—Four directional-arrow keys with two-speed repeat;
—Two user-definable Apple keys;
—Seven other special keys: SHIFT, CONTROL, ALPHA LOCK, TAB, ESCAPE, RETURN, ENTER.
Storage Devices:
—One 5.25-inch floppy disk drive built in, 140K (143, 360) bytes per diskette;
—Three additional drives can be connected by daisy-chain cable (Total: 560K bytes on-line storage);
—Up to four ProFile hard-disk drives (5 megabytes each) may be added with plug-in interface cards.
Video Output:
—RCA phono connector for NTSC monochromatic composite video;
—DB-15 connector for:
NTSC color composite video;
NTSC monochromatic composite video;
RGB color video;
Composite sync signal;
Power supply voltages.
—Color signals appear as 16-level grey scale on monochromatic displays.
Audio Output:
—Built-in two-inch speaker; miniature phono jack on back panel;
—Driven by 6-bit D/A converter or fixed-frequency "beep" generator.
Serial (Printer/Modem) Port:
—RS-232C compatible, DB-25 female connector;
—Software-selectable baud rate and duplex mode.
One port may be used for the Silentype printer.

One port may be used for the Silentype printer.

Expansion:
—Four 50-pin expansion slots (fully buffered, with interrupt and DMA priority structure).
Joystick/Silentype Ports:
—Two DB-9 connectors.

■ *Languages Available:*
Apple Business BASIC, Apple *III* Pascal, Apple *III* COBOL.

■ *Emulation Mode:*
Provides hardware emulation of 48K byte Apple *II* Plus. Allows most Apple *II* programs, with the exception of Pascal and FORTRAN, to run without modification.

■ *Electrical Specifications:*
The Apple *III*'s power cord should be plugged into a three-wire 110-120 volt outlet.

■ *Physical Specifications:*
Height: 4.8 inches (12.20 cm)
Depth: 18.2 inches (46.22 cm)
Width: 17.5 inches (44.45 cm)
Weight: 26 lbs. (11.8 kg)
The Apple *III* meets the following agency regulations:
UL 114 — Office Appliances and Business Equipment.
CSA 22.2, No. 154—Data Processing Equipment.

The Apple *III* Personal Computer System Package
U.S. Order Number A3S0256

With your order for an Apple *III* System you will receive:
256K Apple *III*;
Power cord;
Monitor cable;
System Demonstration disk;
System Utilities disk;
System Utilities Data disk (contains device driver files, character sets, and keyboard layouts);
Apple *II* Plus Emulation disk;
Owner's Guide;
Standard Device Drivers Manual;
Warranty and service information.

Specifications or products may change without notice.

apple computer®
20525 Mariani Avenue
Cupertino, California 95014
(408) 996-1010
TLX 171-576

A3F0147 300K 7/83

PRINTED IN U.S.A.

United States Patent [19]

Smith

[11] **4,445,414**

[45] **May 1, 1984**

[54] **DIGITAL, SIMULTANEOUS, DISCRETE FREQUENCY GENERATOR**

[75] Inventor: **Burrell Smith**, Palo Alto, Calif.

[73] Assignee: **Apple Computer, Inc.**, Cupertino, Calif.

[21] Appl. No.: **351,653**

[22] Filed: **Feb. 24, 1982**

[51] Int. Cl.³ **G10H 7/00**

[52] U.S. Cl. **84/1.01**; 328/16; 364/770

[58] Field of Search 84/1.01; 328/16–18; 364/770

[56] **References Cited**

U.S. PATENT DOCUMENTS

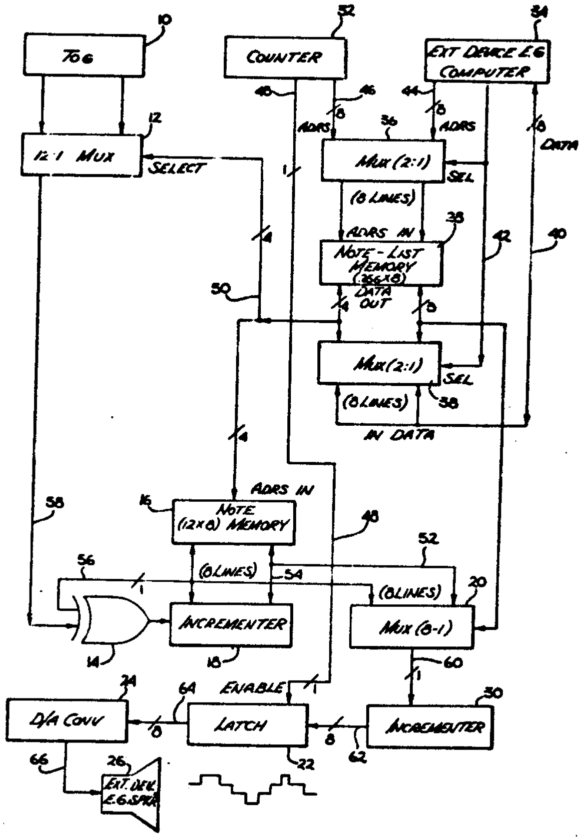
3,989,940	11/1976	Kihara	364/770
4,134,321	1/1979	Woron	84/1.24 X
4,193,892	4/1980	Gross	84/1.24 X
4,256,003	3/1981	Deutsch	84/1.01
4,269,101	5/1981	Deutsch et al.	84/1.01
4,333,374	6/1982	Okumura et al.	84/1.01
4,338,674	7/1982	Hamada	84/1.01 X
4,345,500	8/1982	Alonso et al.	84/1.01
4,351,212	9/1982	Okamoto et al.	84/1.01

Primary Examiner—Stanley J. Witkowski
Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman

[57] **ABSTRACT**

A waveform of arbitrary complexity may be generated using a minimal number of circuit elements and minimal complexity by generating a frequency domain from a minimum set of base frequencies by storing lower octave frequencies of the minimal base set within a note memory. Selected octave and pitch, or note may be read from the note memory according to a list of notes to be thusly read as stored within a note list memory. Each of the instantaneous values of the base frequencies read from the note memory is then added in an accumulator to represent the instantaneous value of the sum of notes or tones comprising the complex frequency at that time. The application of process time periods will replicate an arbitrary complex waveform. Such a frequency generator can find wide application within electronic musical devices, tests and analysis instrumentation, communications and many other fields.

5 Claims, 1 Drawing Figure



Macintosh

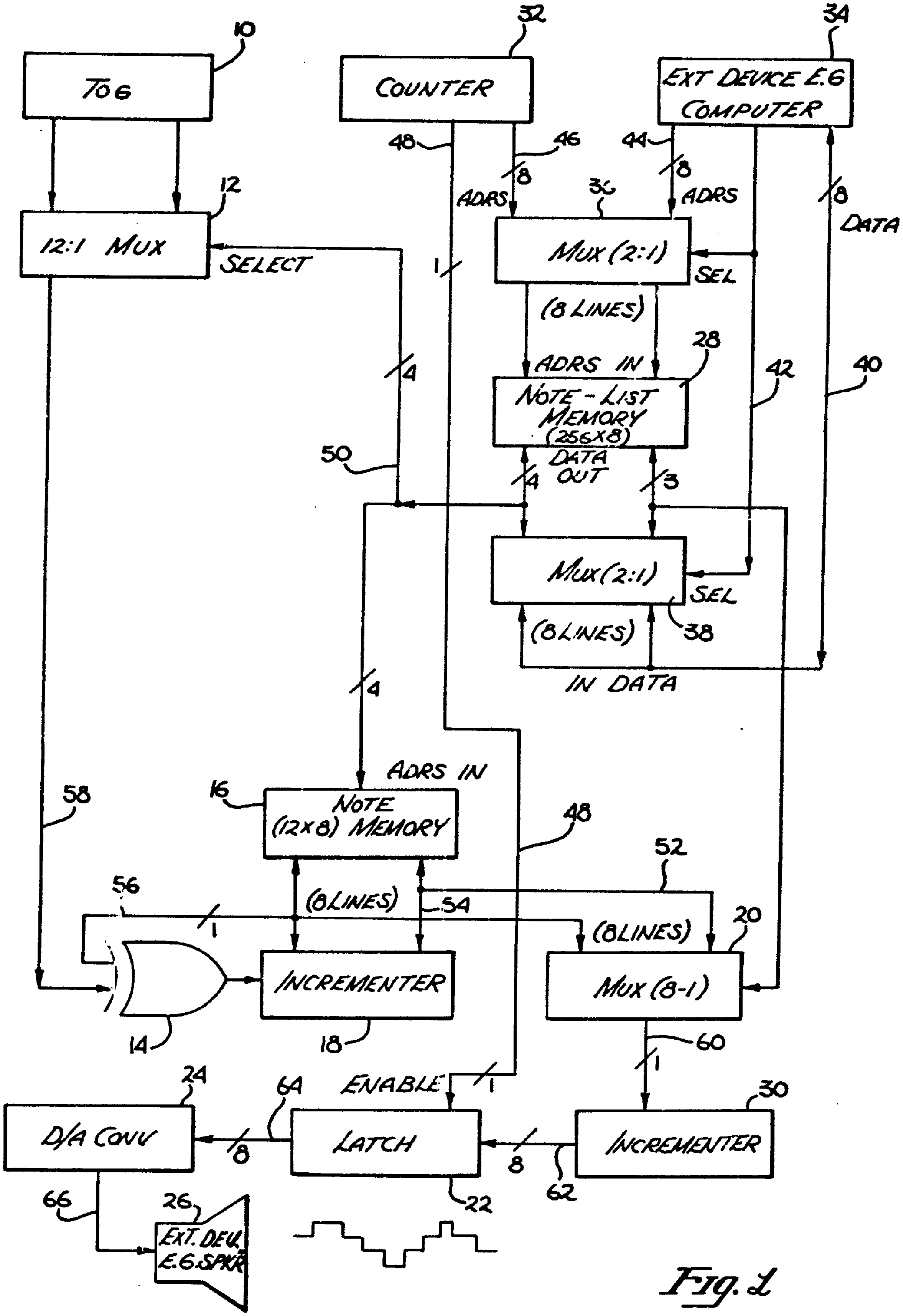

10
TOG
12
12:1 MUX
SELECT
32
COUNTER
34
EXT DEVICE E.G
COMPUTER
48
46
8
ADRS
44
8
ADRS
8
DATA
36
MUX (2:1)
SEL
1
(8 LINES)
28
ADRS IN
NOTE - LIST
MEMORY
(256X8)
DATA
OUT
4
3
42
40
4
50
MUX (2:1)
SEL
38
(8 LINES)
IN DATA
4
ADRS IN
58
16
NOTE
(12X8) MEMORY
48
52
56
1
(8 LINES)
54
(8 LINES)
20
INCREMENTER
MUX (8-1)
14
18
60
1
24
64
ENABLE
1
30
D/A CONV
8
LATCH
8
INCREMENTER
62
22
66
26
EXT. DEV.
E.G. SPKR.

Fig. 1

DIGITAL, SIMULTANEOUS, DISCRETE FREQUENCY GENERATOR

BACKGROUND OF THE INVENT:ON

1. Field of the Invention

The present invention relates to the field of digital frequency synthesizers and in particular, relates to musical tone generators and frequency synthesizers.

2. Prior Art

Frequency synthesizers are categorized as either analog or digital. In each category generators have been devised to produce one frequency at a time or multiple frequencies simultaneously.

Analog frequency synthesizers have been generally characterized by requiring a distinct electrical component for each discrete frequency. In other words, to create a number of frequencies an equal number of components such as resistors, inductors, switching circuitry or oscillators are required to simultaneously create the same number of frequencies. Complex switching circuits are devised to control switching between a smaller number of controlling components and a larger number of controlled oscillators, or tone generators. Large and complex circuits are the result.

The design or digital circuitry often parallels prior analog circuitry used for simultaneous generation of frequencies. Such digital circuits also incorporate a generally linear increase in component count with an increase in the number of frequencies generated. For example, a separate oscillator is required for each frequency, such as a flip-flop, phase-locked-loop or monostable oscillator. Electronic or mechanical switching between frequency determining components such as resistors or crystals is also used in order to control as many oscillators as frequencies which are required. A small set of fixed frequencies may be heterodyned to create a larger set of frequencies. In the heterodyning method, switching complexity increases as the number of simultaneous frequencies also increases. In addition, when heterodyning the set of fixed frequencies necessarily becomes even larger when the frequencies which are ultimately desired are not simply related. When digital counters are used as the basic element in frequency generators, the result is that the number of digital counters required equals the number of desired frequencies. The prior art uses a small number of separate oscillators to clock a number of counters to provide in turn a multiplicity of low frequency signals. Each desired frequency thus requires a separate counter. Shift registers have been used in the same manner as counters to produce a multiplicity of low frequency signals.

What is needed then is circuitry and a methodology for simultaneously producing a large number of frequencies without necessitating a corresponding increase in the number of separable elements required to generate the number of discrete frequencies desired.

BRIEF SUMMARY OF THE INVENTION

The present invention is an apparatus for simultaneously generating a multiple of frequencies comprising a means for generating a plurality of base frequencies, a note memory, a comparison means, and an incrementing means. The base frequency generating means is coupled to the note memory which is used for storing as many corresponding words as the number of the plurality of base frequencies. The comparison means is coupled to the note memory and addresses the memory and compares the lowest order of bits of each word in the note memory to the corresponding base frequency. The rate of comparison of the comparison means is greater than the highest base frequency. The incrementing means conditionally adds one to the corresponding word in note memory if the comparison generated by the comparison means indicates an inequality between the base frequency and the lowest order bit of the addressed word. By a combination of these elements, octaves of each of the base frequencies are generated for simultaneous output.

In another embodiment of the present invention the invention further comprises a note list memory for storing addresses and bit location codes of selected words in the note memory. A bit means is provided for selectively reading every address and bit location code in the note memory and for addressing a selected bit from the selected words in the note memory. An output means adds each of the selected bits and generates a sum output signal. By virtue of these additional elements, an arbitrary waveform may be generated from the base frequencies.

The following figures show one embodiment of the present invention whereby simultaneous multiple frequencies may be generated according to the present invention. Like elements are referenced by like numerals.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is the sole FIGURE which illustrates in simplified block form one circuit organization which can be utilized to achieve the objects of the invention.

DETAILED DESCRIPTION OF THE INVENTION

The present invention is a generator for simultaneously producing multiple frequencies and differs from the prior art in that there is no increase in the complexity or number of circuit elements as the number of simultaneous frequencies desired increases. The present invention maximizes the efficiency by which memory can be utilized to produce simultaneous frequencies by utilizing a single bit to produce each frequency. In the illustrated embodiment, a musical tone generator is described, although the same principles could be applied using ordinary skills in the art according to the present teachings in order to devise a generalized frequency synthesizer. In addition, the illustrated embodiment will show the generation of a squarewave tone. However, it is well understood that sinusoidal, triangular or any other non-rectangular waveforms can be easily generated based upon the rectangular waveforms using well-known waveform generation techniques including, but not limited to, Walsh functions.

According to the basic principle of the invention a plurality of base frequencies or pitches are generated from which a multiple number of octaves are constructed. In other words, the musical scale is generated in the highest octave and all lower octaves desired are derived therefrom. This can be readily accomplished in the present invention by noting that each higher order bit in a binary word changes as the number increases in unit steps at half the frequency as the next lower order bit. Thus, by adding one to the lowest order bit of a binary word, the various bits in the word form a representation of as many octaves as there are bits. Thus, a single binary word can represent a number of different octaves of a single tone or pitch. In the chromatic scale, twelve tones comprise the octave. These twelve tones are generated by a conventional top octave generator 10 (hereinafter TOG). A chromatic scale is described only for the purposes of illustration and in no way is it intended to limit the scope of the present invention. Clearly, many other scales or relationships between a set of base frequencies can be selected according to the application and objectives at hand. For example, the base frequencies may be numerically generated by a computer or may be produced by a fixed memory.

The twelve output lines of TOG 10 are coupled to a multiplexer 12. The output of multiplexer 12 is coupled to an exclusive-OR gate 14. A random access memory, or note memory 16 provides memory capacity for twelve words, each of eight bits in length. In the illustrated embodiment, eight bit words are chosen inasmuch as this is a convention within the industry and moreover, eight octaves are usually sufficient to provide a full dynamic range for a musical instrument. The lowest order output bit of note memory 16 is coupled as the other input to exclusive-OR gate 14. The output of OR gate 14 is coupled to an incrementer 18 whose output is coupled to the accessed word location from note memory 16.

Thus, the basis of the operation of the present invention can be understood by just these few elements. The selected pitch, or tone from TOG 10 is coupled through a multiplexer 12 to exclusive-OR gate 14. Gate 14 will present a 1 to incrementer 18 in the event that the least significant bit of the accessed word from note memory 16 is different than the value of the base frequency selected from TOG 10 during that clocked period and otherwise presents a 0 output if the least significant bit from the accessed word and the selected pitch from TOG 10 are the same. If the value stored in memory is different than that present on the selected pitch of TOG 10, the word in memory is incremented by 1, (the output of exclusive-OR gate 14) and stored back into note memory 16 at the same accessed address. By selecting the comparison or clock rate to be greater than the highest frequency of the plurality of base frequencies generated by TOG 10, one can always be assured that the contents of note memory 16 have been updated during a period of time so small that none of the waveforms generated by TOG 10 have changed. In the illustrated embodiment, it is sufficient to drive the circuitry of FIG. 1 by a conventional clock at a rate twice the highest frequency generated by TOG 10 in order to assure this result.

The output of note memory 16 is also coupled to multiplexer 20. Thus, the eight octaves represented by a pitch contained in a single word of memory is presented to the inputs of multiplexer 20. As will be described below in greater detail, the output of multiplexer 20 ultimately will be coupled to a latch 22 and a digital-to-analog converter 24 for conversion into an audio signal through the speaker 26 to produce the selected note. The note can be arbitrarily selected according to conventional principles well-known to the art. Pitch can be selected by addressing note memory 16 and the octave selected by controlling multiplexer 20 according to the present teachings.

Another aspect of the present invention can now be understood by reviewing the remaining elements within the circuitry of FIG. 1. The present invention is particularly adapted to a convenient method and means for presenting an arbitrary output waveform. In the illustrated embodiment, a note list word may be constructed on a 3-bit field comprising a bit location code representing the selected octave of any given pitch. Similarly, a 4-bit field comprises a pitch location code and is capable of representing any one of the twelve pitches within each corresponding octave. Thus, a 7-bit word, formed of a 3 and 4 bit field, is capable of indicating any one of the 96 different notes which the illustrated embodiment is capable of generating. By an expansion of these principles, any greater or lesser scale can also be represented without undue complication or proliferation of circuitry.

Note list memory 28 is a random-access memory capable of storing these 7-bit words. In the illustrated embodiment, in fact, note list memory 28 is a memory comprised of 256 bytes. Although note list memory 28 and note memory 16 have been shown and described as separate memories, it is clear that they may in fact be distinguishable portions of the same memory elements or organized in any other equivalent fashion. Note list memory 28 and note memory 16 have been shown and described herein as separate memories solely for the purposes of clarity of explanation and ease of understanding. In addition, note list memory 28 may be substantially larger than 256 words and in fact may be as large as practical to include as many different notes as may be required during any given time period to represent a complex output waveform.

The 4-bit pitch location field is coupled from note list memory 28 to multiplexer 12 to select the appropriate base frequency from TOG 10 as described above and to the address input of note memory 16 in order to simultaneously present the appropriate pitch word at the output of note memory 16. The 3-bit location code or octave field is simultaneously presented at the output of note list memory 28 and is coupled to multiplexer 20 wherein the appropriate octave of the selected pitch is coupled to an incrementer, or accumulator 30. A conventional counter 32 passes through the entire address domain or note list memory 28 to read the entire contents of memory 28 in order to call forth from memory 16 all the notes required for the simultaneous creation of the complex frequency. Each of these selected bits are accumulated in accumulator 30. When the entire contents of note list memory 28 have been read by counter 32, latch 22 is enabled by counter 32 and the contents of accumulator 30 is latched therein. Digital-to-analog converter 24 converts the digital signal stored at that moment in latch 22 into a corresponding analog voltage level.

The entire contents of note list memory 28 are read out at a rate higher than the highest desired output frequency in order to insure that the proper value of the selected octaves based upon the frequencies of TOG 10 are accumulated in accumulator 30. By the time that the last frequency output of TOG 10 changes value, note list memory 28 will be scanned at least twice again to select the various pitches and sub-octaves as specified in the note list contained within note list memory 28. If a pitch in note memory 16 is not contained in note-list memory 28, it will not be updated. However, when the pitch does appear in note-list memory 28, it will be updated. Inasmuch as only the oscillation rate of the bits in note memory 16 are significant and not the magnitude of the stored number, the time at which updating commences or ceases is irrelevant.

Information in note list 28 is controlled by an external device or computer 34 of any type well-known to the art. In the illustrated embodiment, the entire contents of note-list memory 28 can be changed between any given clock cycle by reading in a new list through multiplexer 38 under the address control of multiplexer 36. In other words, the address locations in note-list memory 28 are provided by external user device 34, such as through software control, by coupling the address through 2-to-1 multiplexer 36. Simultaneously therewith, the note memory addresses are read into the selected locations in note-list memory 28 through 2-to-1 multiplexer 38 on a data bus line 40 from user 34. Multiplexers 36 and 38 are controlled by a select line 42 again controlled by user 34 in order to appropriately select either data and addresses from user 34 or addresses from counter 32. Data bus 40 is also bidirectional to allow the contents of note list memory 28 to be read through multiplexer 38 for any purpose desired by user 34.

Although the present invention has been described in connection with the specifically illustrated embodiment as shown in FIG. 1, many other applications or alterations may be made in the present invention without departing from its spirit and scope. For example, additional circuitry may be added according to well-known design principles by following the present teachings to add amplitude information or to directly generate non-rectangular waveforms. The amplitude of any given note may increased in the illustrated embodiment by simply including that note several times in the note list memory. Alternatively, additional memory may be provided to store amplitude information which can be then used to multiply or amplify the digital analog data by any conventional means. In other words, an amplitude field could likewise be included within the words of the note list memory 28 in the same manner and the octave and pitch fields.

Although generation of the notes within note memory 16 have been shown by a combined use of exclusive-OR gate 14 and incrementer 18, many other alternative means may also be included such as reading the word into a register, adding one into the register and then rewriting the word back into the same memory location; or using note memory 16 in such a manner that each word location is an accumulator.

In addition, other fields within the words stored in note list memory 28 may be created and utilized in various applications. For example, a bit may be reserved to indicate if a certain word should be skipped. This would be of use where all the notes in note memory 16 were constantly updated but only those indicated by note-list memory 28 were to produce an audible note. One or more bits may also be reserved to indicate which of two or more channels of which the output should be directed.

Although the present invention has been described as a musical tone generator, it must be clearly understood that this application is merely a single preferred embodiment of the inventive concept which can be employed productively in many other situations. For example, the present invention may be used to analyze an arbitrary waveform by successively approximating the waveform by generating a series of iterations by a circuit of the type shown in FIG. 1. An external computer can be used to make the comparative analysis at each step of the iteration and to provide the appropriate feedback parameters to the circuitry of FIG. 1 for the next approximation. The speed of the circuitry of FIG. 1 is such that waveform analysis of this type can be easily accomplished.

Thus, what has been devised is a voice, or tone generator of heretofore unobtainable speed, flexibility and simplicity. In the prior art, a 256 voice generator required hundreds of integrated circuit packages whereas a prototype of the present invention was capable of functioning as a 256 voice generator with approximately 20 integrated circuits. Thus, the circuitry in the present invention is capable of applications, not only within the field of musical instrumentation but also within the fields of test and analysis instrumentation, communications, and many other fields as well.

I claim:

1. An apparatus for generating simultaneous multiple frequencies comprising:

base frequency means for generating a plurality of base frequencies;

note memory for storing as many corresponding words as the number of said plurality of base frequencies;

comparison means for comparing the lowest order bit of each said word in said note memory to said corresponding base frequencies, said comparing occurring at a higher rate than the highest base frequency; and

increment means for adding 1 to said corresponding word in said note memory if comparison by said comparison means indicates inequality between said base frequency and said lowest order bit;

wherein said comparison means includes an exclusive OR gate and said incrementing means includes an incrementer, one input of said exclusive OR gate being coupled to said base frequency means and the other input of said exclusive OR gate being coupled to the least significant bit output line from said note memory, the output of said exclusive OR gate being coupled to one input of said incrementer, the other inputs of said incrementer being coupled to the outputs of said note memory,

whereby the output of said exclusive OR gate is added to the addressed contents of said note memory to create a word representative of the pitch of the base frequency where each higher order bit is a lower octave of said base frequency, and

octaves of each said base frequencies are generated for simultaneous output.

2. The apparatus of claim 1 further comprising:

a note-list memory for storing addresses of selected words in said note memory and corresponding octave field codes;

note address means for successively reading every address and octave field in said note-list memory and for addressing a selected bit from said selected words in said note memory; and

output means for adding each said selected bit and generating a sum output signal,

whereby arbitrary waveforms may be generated from said base frequencies.

3. The apparatus of claim 2 wherein said base frequency means is a top octave generator for producing a chromatic series of pitches.

4. The apparatus of claim 2 wherein said output means comprises:

a multiplexer having its inputs coupled to said note memory;

an accumulator having its input coupled to said multiplexer;

“APPLE_PAT_4_445_414_05” 326 KB 2000-02-21 dpi: 300h x 300v pix: 1925h x 2903v

digital-to-analog conversion means coupled to said accumulator for selectively generating an analog signal from a digital input; and

control means coupled to said digital-to-analog means

for enabling said digital-to-analog means when said accumulator has reached a final value.

5. The apparatus of claim 4 wherein said control means is a counter used to address and read-out the entire contents of said note-list memory.

* * * * *

United States Patent [19]

Lapson et al.

[11] **Patent Number: 4,464,652**

[45] **Date of Patent: Aug. 7, 1984**

[54] **CURSOR CONTROL DEVICE FOR USE WITH DISPLAY SYSTEMS**

[75] Inventors: **William F. Lapson**, Cupertino; **William D. Atkinson**, Los Gatos, both cf Calif.

[73] Assignee: **Apple Computer, Inc.**, Cupertino, Calif.

[21] Appl. No.: **399,704**

[22] Filed: **Jul. 19, 1982**

[51] Int. Cl.³ **G09G 1/00**

[52] U.S. Cl. .... **340/710**; 340/709; 340/716; 74/471 XY

[58] **Field of Search** .... 340/710, 709, 809, 810, 340/870.28, 870.29, 711, 716; 250/231 SE; 74/198, 471 XY; 358/183

[56] **References Cited**

U.S. PATENT DOCUMENTS

3,395,589	8/1968	Gersten	74/198
3,541,541	11/1970	Engelbart	340/710
3,625,083	12/1971	Bosc	74/471 XY
3,835,464	9/1974	Rider	340/710
3,987,685	10/1976	Opocensky	340/710
4,245,244	1/1981	Lijewski et al.	358/183
4,310,839	1/1982	Schwerdt	340/709
4,369,439	1/1983	Broos	340/710
4,404,865	9/1983	Kim	74/471 XY

FOREIGN PATENT DOCUMENTS

1526428	9/1978	United Kingdom	340/710

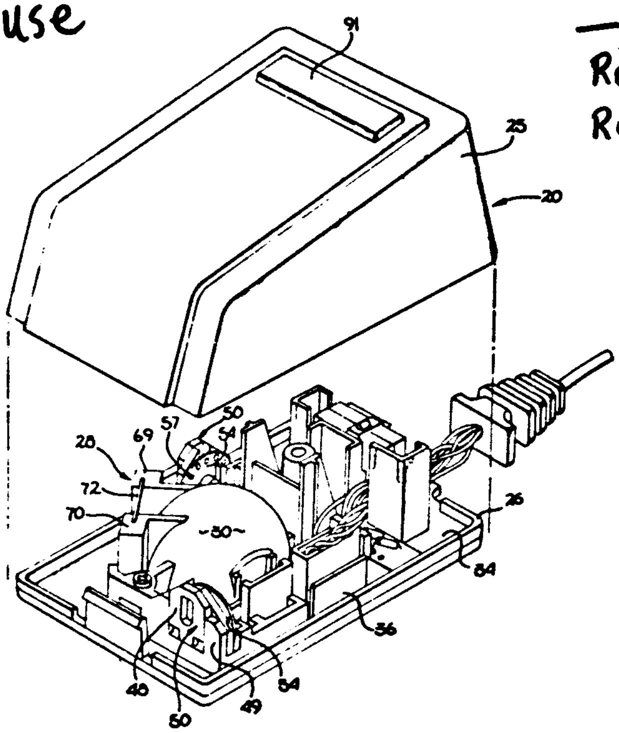
Primary Examiner—Gerald L. Brigance
Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman

[57] **ABSTRACT**

A cursor control device having particular application to a computer display system is disclosed. The cursor control includes a unitary frame, having a domed portion substantially surrounding and retaining a ball which is free to rotate. X-Y position indicating means are provided, such that rotation of the ball provides signals indicative of X-Y positions on the display system. The ball is free to "float" in the vertical direction within the dome, and thereby maintain good surface contact. X-Y positions are established by movement of the control device over a surface. A display system and method is disclosed for use in conjunction with the cursor control device, which permits a user to select command options simply by movement of the displayed cursor over a "pull-down" menu bar.

13 Claims, 15 Drawing Figures

Lisa mouse



Revisions

Re. 32,632 29 Mar 1988
Re. 32,633 29 Mar 1988

DAVID T. CRAIG

"APPLE_PAT_4_464_652_01" 182 KB 2000-02-21 dpi: 300h x 300v pix: 1973h x 2807v

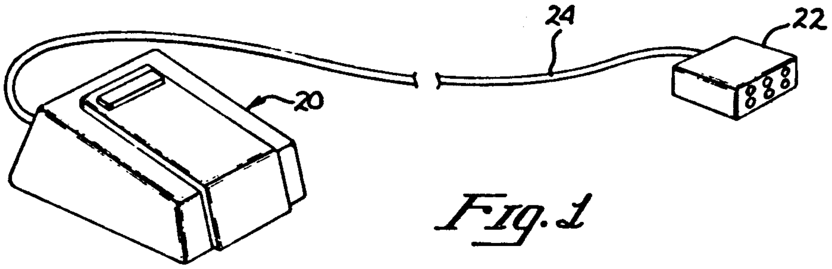


Fig. 1

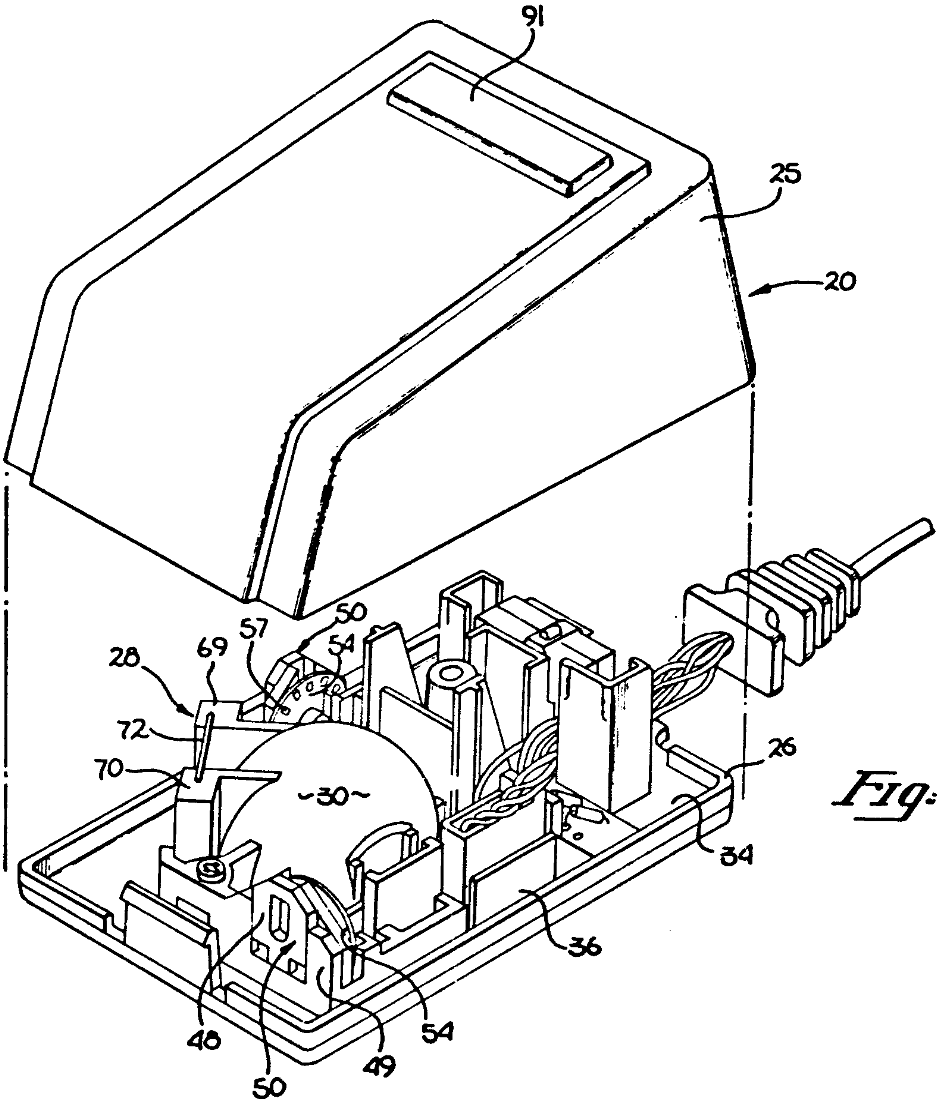


Fig. 2

“APPLE_PAT_4_464_652_02” 134 KB 2000-02-21 dpi: 300h x 300v pix: 1820h x 2942v

Fig. 3

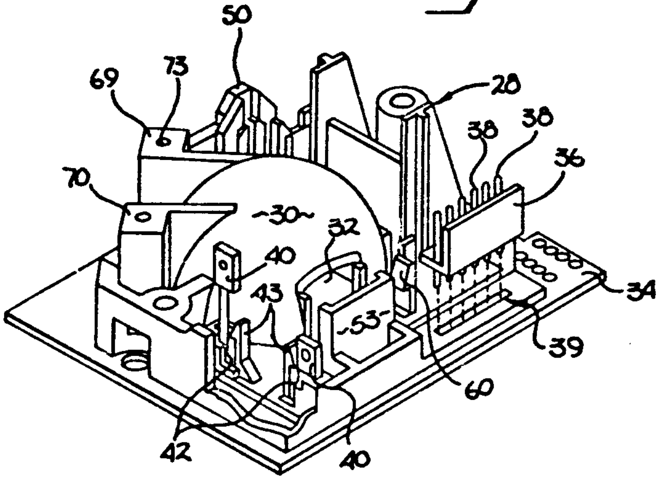
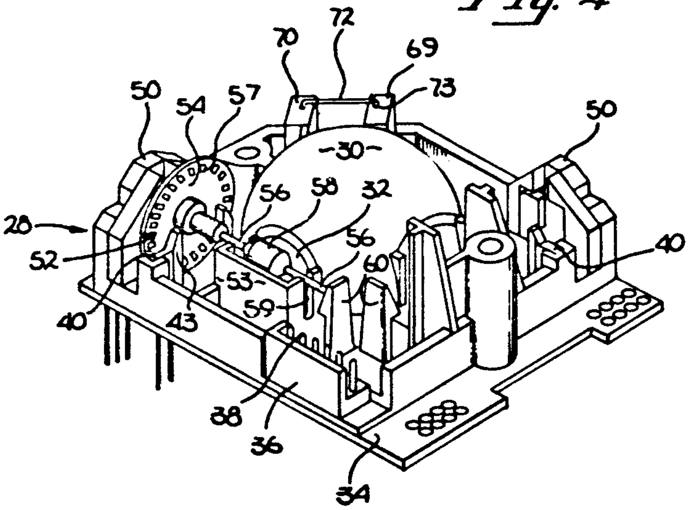


Fig. 4



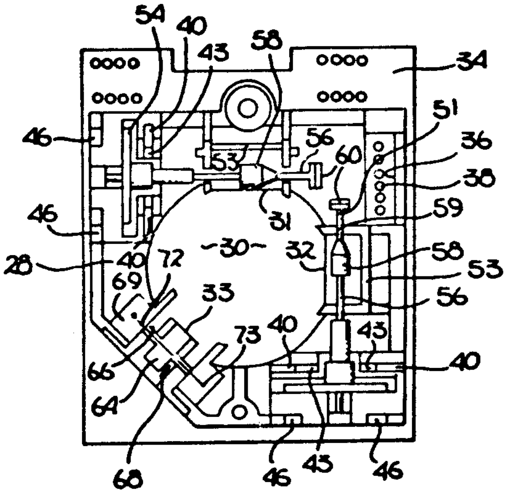


Fig. 5

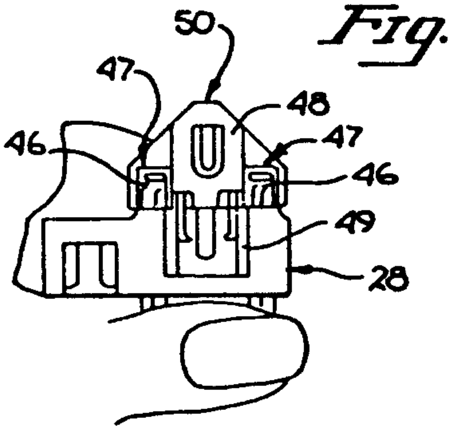


Fig. 6

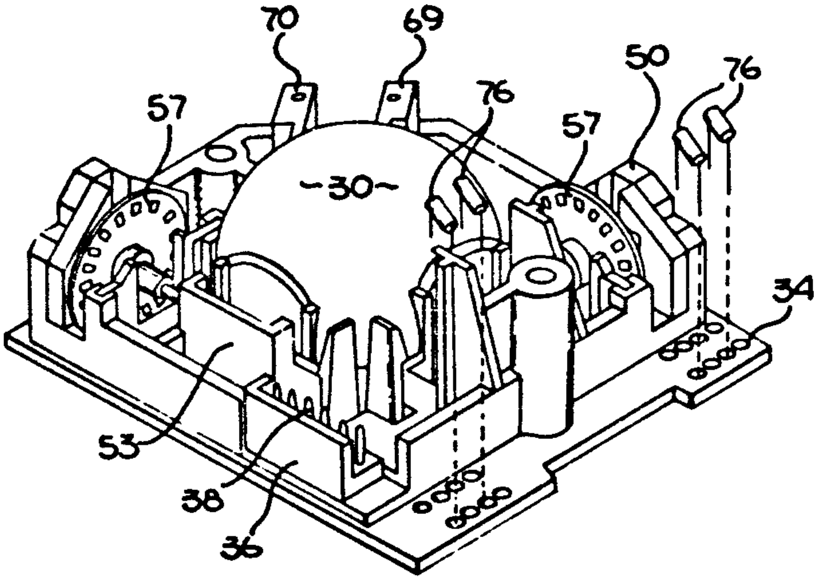


Fig. 7

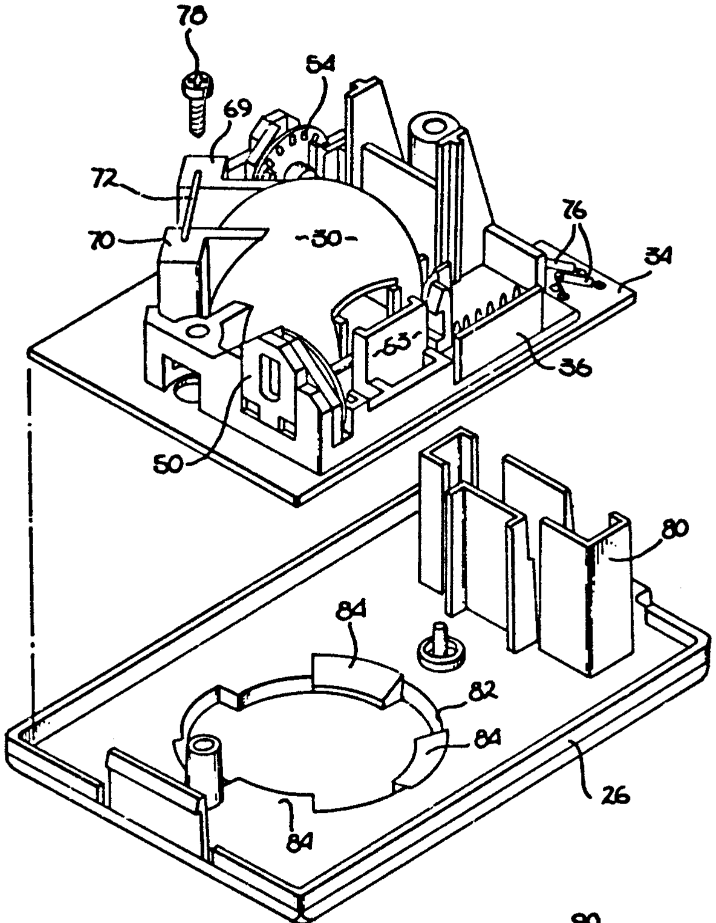

78
54
69
72
70
-30-
76
34
36
-53-
50
80
84
82
84
26
84

Fig. 8

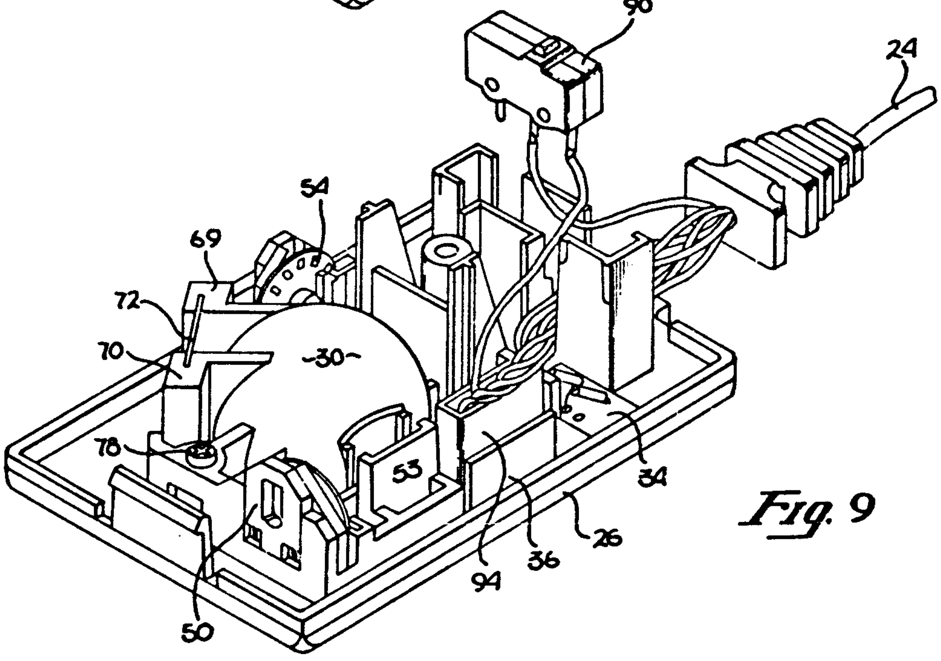

90
24
54
69
72
70
-30-
78
53
34
26
36
94
50

Fig. 9

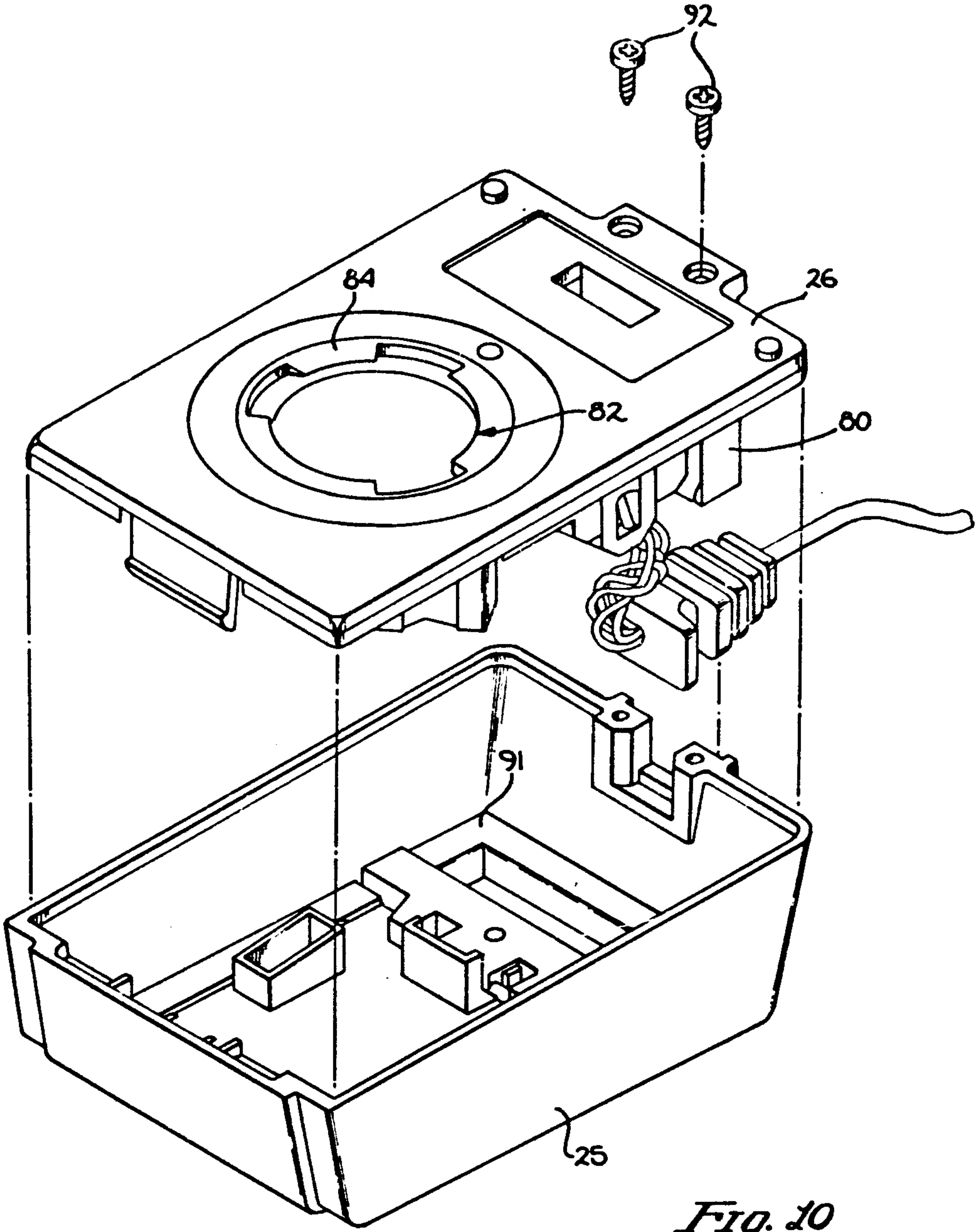


Fig. 10

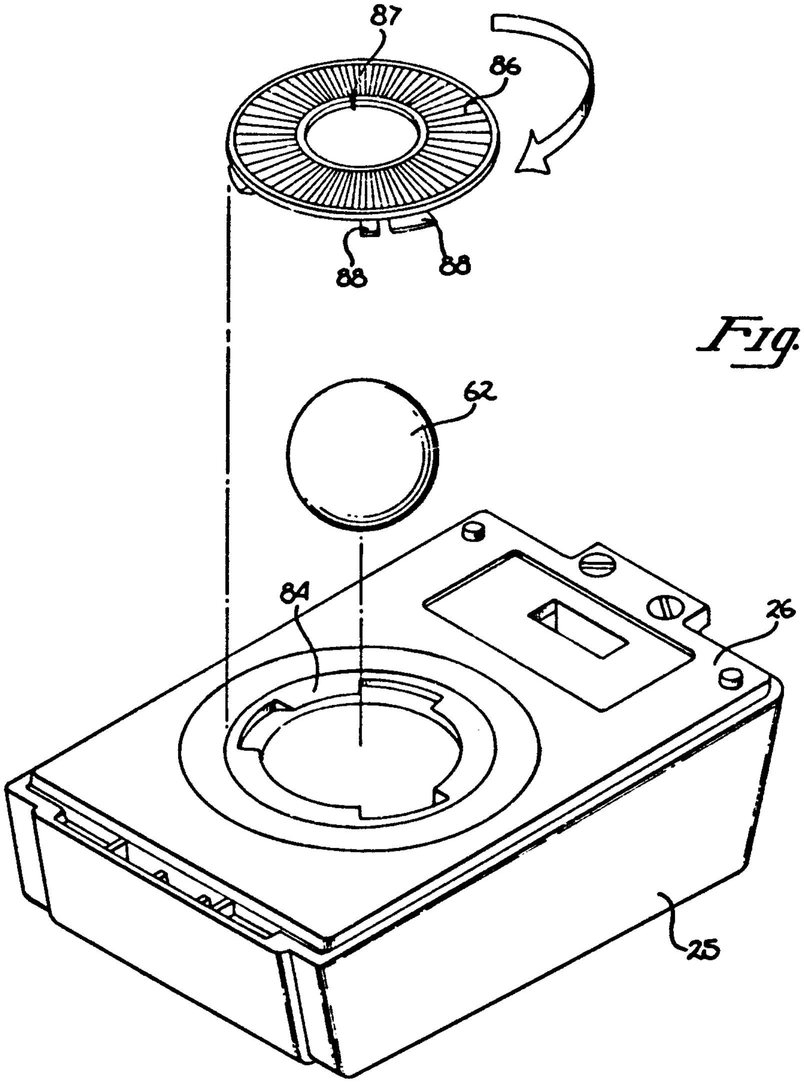


Fig. 11

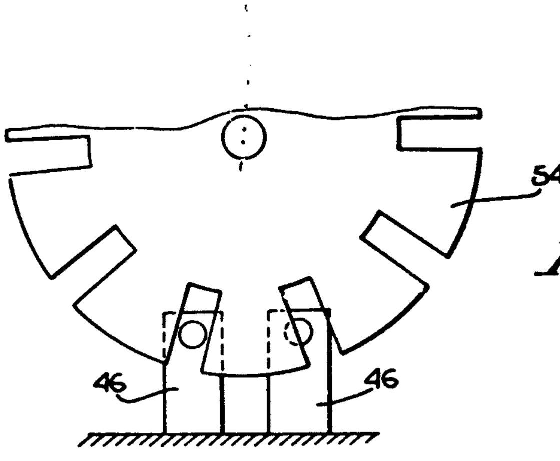
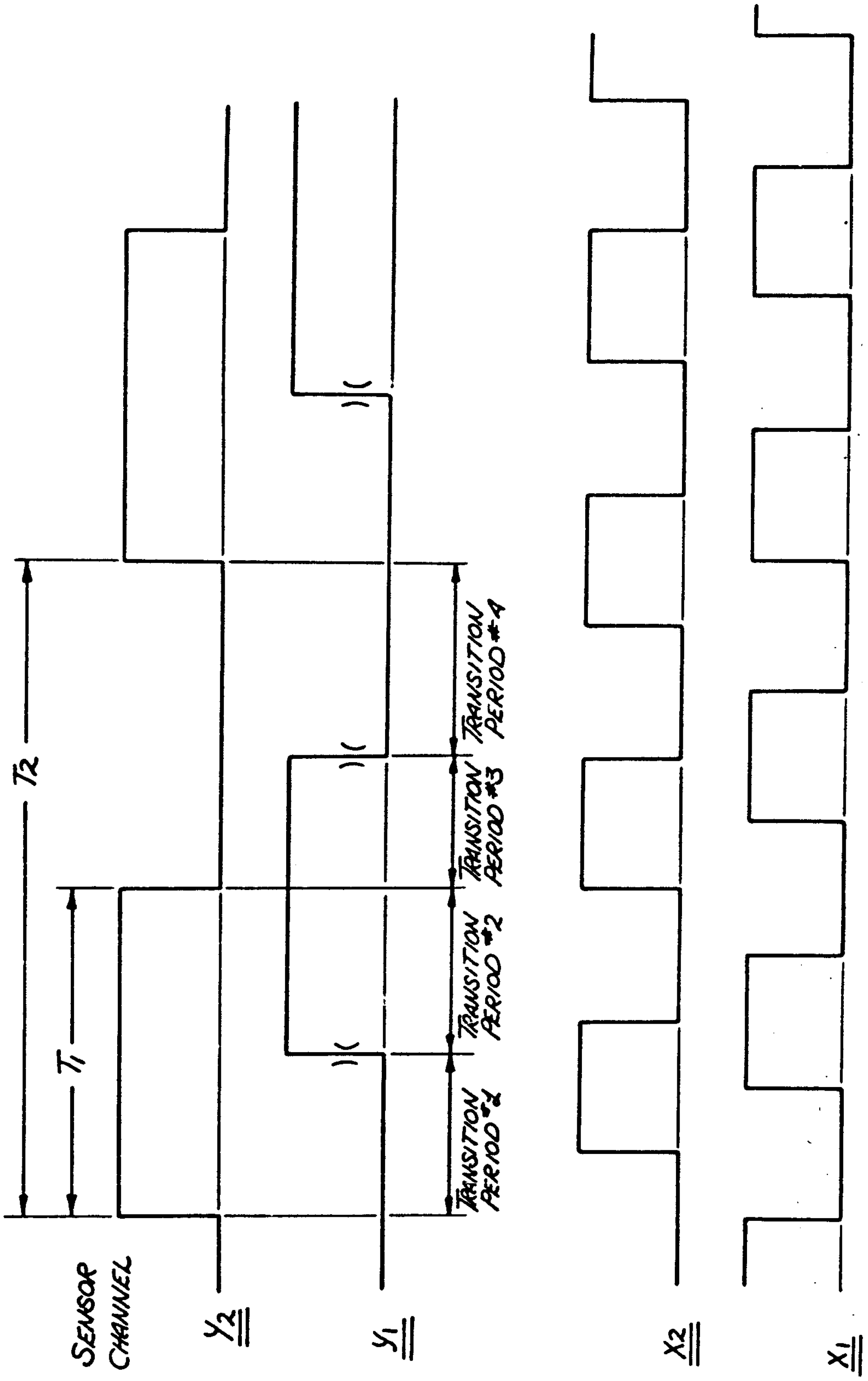


Fig. 12



OUTPUT SIGNAL TIMING

Fig. 13

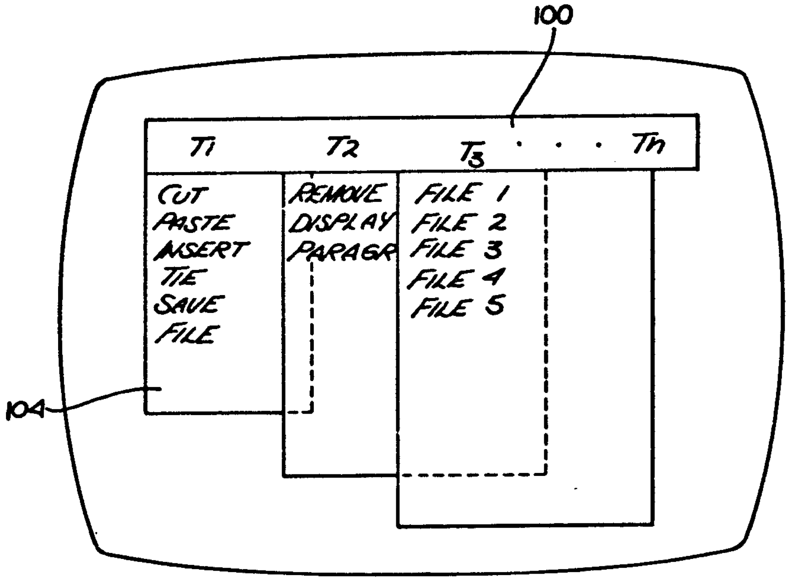

100
T1
T2
T3
. . .
Tn
CUT
PASTE
INSERT
TIE
SAVE
FILE
REMOVE
DISPLAY
PARAGR
FILE 1
FILE 2
FILE 3
FILE 4
FILE 5
104

Fig. 14

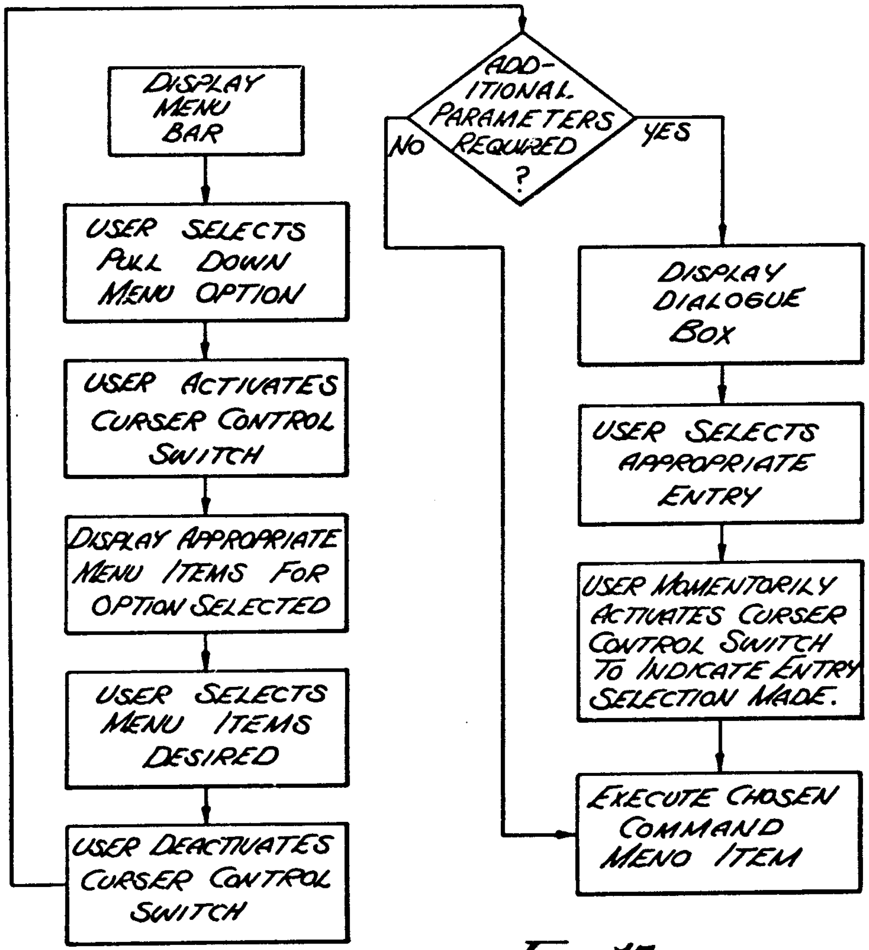

DISPLAY MENU BAR
USER SELECTS PULL DOWN MENU OPTION
USER ACTIVATES CURSER CONTROL SWITCH
DISPLAY APPROPRIATE MENU ITEMS FOR OPTION SELECTED
USER SELECTS MENU ITEMS DESIRED
USER DEACTIVATES CURSER CONTROL SWITCH
ADD-ITIONAL PARAMETERS REQUIRED ?
NO
YES
DISPLAY DIALOGUE BOX
USER SELECTS APPROPRIATE ENTRY
USER MOMENTORILY ACTIVATES CURSER CONTROL SWITCH TO INDICATE ENTRY SELECTION MADE.
EXECUTE CHOSEN COMMAND MENO ITEM

Fig. 15

CURSOR CONTROL DEVICE FOR USE WITH DISPLAY SYSTEMS

BACKGROUND OF THE INVENTION

1. Field

The present invention relates to the field of display systems, and more particularly to devices which can position a cursor over selected locations on a computer controlled display.

2. Art Background

In many computer controlled display systems, it is desirable to allow the user to control the position of a cursor or the like by means which are external from the main computer keyboard. For example, a user may be required to repetitively choose software options displayed on a cathode ray tube (CRT), or may desire to input data in a diagram format into the computer system. In such situations traditional keyboard input systems are not as effective as a cursor control device commonly referred to as a "mouse".

In a typical "mouse" system, a hand-held transducer provides positional movement signals to the display system. Traditionally, the movement of wheels within the cursor control device are coupled to potentiometers to provide signals indicative of an X-Y position on the display screen (see U.S. Pat. Nos. 3,541,541; 3,269,190; and 3,835,464). Other mouse systems utilize rotating balls on wheels which are in turn coupled to rotate apertures interrupting beams of light, thereby providing positional signals to the display system (see U.S. Pat. Nos. 3,892,963 and 3,541,521).

One common disadvantage of cursor control devices found in the prior art is their cost. Typically, prior art cursor controls include costly mechanical parts which require precise alignment for proper operation. Moreover, it is not uncommon for these devices to exhibit a loss in accuracy over time as the mechanism wears. As computer display capabilities have become more advanced in terms of user real-time graphic interation, cursor control devices have become a necessity in many computer systems. Accordingly, there exists a need to provide a cost effective, simple and highly reliable cursor control device for providing signals indicative of X-Y positions on a computer display system.

As will be disclosed below, the present invention provides an improved cursor control device which overcomes the disadvantages of the prior art by utilizing a unitary frame structure for accurate alignment of all elements and simple assembly, as well as photo-optics to provide the required positional signals. In addition, a display system and method is disclosed for use in association with the cursor control device which permits a user to select command options simply by movement of the cursor over a "pull-down" menu bar.

SUMMARY OF THE INVENTION

A cursor control device having particular application to computer display systems is disclosed. The cursor control includes a unitary frame having a domed portion which houses a ball which is free to rotate. Two encoder disc assemblies are provided, which include roller shafts disposed substantially 90 degrees relative to one another and in contact with the ball. Each roller shaft is coupled to an encoder disc having a plurality of slots disposed radially around the disc periphery. These slots interrupt light beams which are provided by photoemitters and directed at photo-detectors. Each slotted disc interrupts two light beams which are arranged such that when one beam is fully transmitted, the other is partially blocked. Beam interruptions produce signal pulses representing increments of motion, while the order in which the light beams are interrupted indicates the direction of motion, thereby resulting in an X-Y position on a display system. The ball is maintained in contact with the roller shafts by a spring biased idler wheel. The ball is free to "float" in the vertical direction within the dome, and thereby maintain good surface contact. Moreover, the ball may be easily removed for cleaning to insure that any build up of lint or the like does not prevent the ball from rotating smoothly. A switch is provided within the cursor control housing in order to signal the display system that a desired X-Y location on the display screen has been selected. In operation, a user may selectively position a cursor or the like on a display system by simply moving the cursor control device over a surface, such as a desk, until the desired cursor position is shown on the display device. A display system and method is disclosed for use in conjunction with the cursor control device, which permits user to select command options simply by movement of the displayed cursor over a "menu bar".

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a perspective view of the present invention.

FIG. 2 is a perspective of the present invention illustrating the cursor control device as it appears without the housing cover.

FIG. 3 is a perspective view of the unitary frame of the present invention coupled to the printed circuit board base, illustrating the placement of photo-detectors and the coupling connector.

FIG. 4 is a further perspective view of the unitary frame and circuit board of FIG. 3 illustrating the position of a roller shaft and encoder wheel.

FIG. 5 is a top view of the unitary frame and printed circuit board of the present invention.

FIG. 6 is a partial view of the unitary frame in FIG. 3, illustrating the insertion of a detector aperture.

FIG. 7 is a perspective view of the unitary frame of FIG. 3, illustrating the placement of resistors on the printed circuit board.

FIG. 8 is a perspective view of the coupling of the unitary frame cage and printed circuit board combination to the housing base of the present invention.

FIG. 9 is a perspective view illustrating the placement of the control switch within the housing base.

FIG. 10 is the perspective view of the final assembly of the present invention illustrating the coupling of the cover and base portions of the housing.

FIG. 11 is a perspective view illustrating the insertion or removal of the floating and rotating ball.

FIG. 12 is a diagrammatical illustration of the alignment of the photo-emitters in relation to each encoder disc.

FIG. 13 is a diagrammatical illustration of a sample quadrature output of the present invention indicative of X-Y locations on a display system.

FIG. 14 is a diagrammatical illustration of a "pull down" menu bar display.

FIG. 15 is a block diagram illustrating the sequence of steps utilized by the present invention to display options and associated commands on a "pull-down" menu bar display.

DETAILED DESCRIPTION OF THE INVENTION

A cursor control device having particular application for use in conjunction with a computer display system is disclosed. In the following description for purposes of explanation, specific numbers, materials and configurations are set forth in order to provide a thorough understanding of the present invention. However, it will be apparent to one skilled in the art that the present invention may be practiced without the specific details. In other instances, well known systems are shown in diagrammatical or block diagram form in order not to obscure the present invention unnecessarily.

Referring now to FIG. 1, the present invention includes a hand held cursor control unit 20 which is coupled to a plug 22 by means of a cable 24. As best illustrated in FIG. 2, cursor control unit 20 includes a cover 25 and a base 26 upon which the internal workings of the present invention are disposed. As will be apparent from the discussion which follows, cursor control unit 20 is designed with ease of assembly in mind, while providing very close tolerances and high X-Y position location accuracy.

With reference to FIGS. 3, 4 and 5, a premolded unitary frame 28 is provided which includes a domed housing 30 presently having three cut-out locations 31, 32 and 33. As illustrated, cut-outs 31 and 32 are disposed substantially at 90 degrees with respect to one another, with cut-out 33 being oriented generally symmetrically opposite the other cut-outs. In addition, frame 28 includes a plurality of bosses, slots and shaped stems of material which when pertinent will be discussed in this specification. In the presently preferred embodiment, the frame 28 is comprised of a plastic material (e.g. polycarbonate) which is impregnated with a lubricant (e.g. teflon). Thus, during operation and throughout its useful life, cursor control unit 20 does not require the addition of either wet or dry lubricants. Frame 28 is mounted on a printed circuit board 34 to facilitate electrical connection between the various electrical elements within the unit. Electrical connector header 36 is mounted as shown (see FIG. 3) to the unitary frame 28 such that connector pins 38 pass through a rectangular slot 39 through the frame to the circuit board below. As will be discussed, cable 24 is electrically coupled to the cursor control unit 20 through connector 36.

As illustrated in FIG. 3, photo-emitters 40 are inserted into slots 42 such that the emitter portion is facing away from the dome 30 (note that one emitter 40 is shown in FIG. 3 partially inserted). Upwardly extending clips 43 are snapped over portions of each emitter 40, as shown, to prevent them from being dislodged. Similarly, two photo-detectors 46 are inserted facing the emitters 40 into slots 47 in each of two detector apertures 50. As shown in FIG. 6, an outwardly extending portion 48 of each detector aperture 50 is aligned with guides 49 formed integrally with the frame 28, and the aperture is then snapped downward into place. Thus, each detector aperture 50 houses two detectors 46 which face two emitters 40, respectively. In the presently preferred embodiment, the emitter/detector combination operates within the infrared region. However, it will be appreciated that any suitable wavelength may be used in a particular application. In addition, presently, the detectors 46 incorporate integral Schmitt triggers to provide detector outputs which more closely approximate a digital signal.

Two encoder disc assemblies are provided to convert, as will be described, the movement of the cursor control unit 20 into signals indicative of X-Y locations defined on the display system. Each encoder assembly 52 includes an encoder disc 54 axially coupled to a roller shaft 56. In addition, each encoder disc 54 is provided with a plurality of radially disposed slots 57 which interrupt the light beams generated by the photo-emitters 40. A cylindrical contact member 58 surrounds each roller shaft 56 at each respective cut out location, as illustrated. Each encoder disc assembly 52 is mounted on the unitary frame 28 by inserting the encoder disc 54 between the detector aperture 50 and emitters 40 and snapping an end clip 60 over the opposite end of the roller shaft 56 (See FIGS. 4, 5 and 7), thereby allowing rotation of the roller shaft and encoder disc with a minimum of friction. As illustrated, each shaft 56 is slipped into and carried by a "U" shaped guide 59 formed from upwardly extending alignment bosses 53 to maintain each roller shaft 56 in proper orientation. End 51 of the shaft 56 is carried for rotation within a hollow portion of the detector aperture 50, such that encoder disc 54 is disposed in close proximity to the aperture 50. The present invention's use of integral lubrication within the frame material, permits each shaft 56 to freely rotate about its longitudinal axis.

As a result of the above described configuration, the radially disposed slots 57 of each encoder disc interrupt two light beams from photo-emitters 40. The position of the emitter/detector combination and encoder disc is such that when one beam is fully transmitted, the other is partially blocked by a slit on the encoder disc. As will be discussed, in operation a ball 62 is disposed within the dome 30 of the frame, and retained such that it is maintained in contact with both cylindrical contact members 58. The rotation of the ball 62 within the dome 30 in turn causes the rotation of each roller shaft 56 and its respective encoder disc. As will be discussed, the beam interruptions from the rotation of each encoder disc 54 produce signal pulses representing increments of motion, while the order in which the light beams are interrupted indicates the direction of motion of the cursor control unit.

Ball 62 is retained against the cylindrical contact members 58 by an idler wheel 64 mounted for rotation on a fixed shaft 66, as best shown in FIG. 5. The idler wheel 64 and shaft 66 are inserted within a slot 68 formed by rectangular bosses 69 and 70 extending upwardly from the frame's base. Wheel 64 extends through cut-out 33 into the interior of the dome 30. The legs of a staple shaped idler spring 72 are inserted through passages 73 passing perpendicular to the horizontal plane of the frame 28 and circuit board 34, thereby retaining the shaft 66 within the slot 68.

Referring now to FIG. 7, resistors 76, which are required by the specific electronics of the emitter/detector combination of the present invention, are inserted into the printed circuit board 34. The resistors 76 and associated leads from the connector 36, photo-emitters 40, and photo-detectors 46 are then electrically connected and soldered in place as is conventionally done in the art.

With reference now to FIGS. 8, 9 and 10, the assembled frame 28 and circuit assembly is mounted on the base 26 by means of a screw 78. As illustrated, base 26 includes an upwardly extending switch retaining portion 80 and a generally circular cut-out orifice 82. As best shown in FIGS. 8, 10 and 11, circular orifice 82 is disposed substantially below the opening of dome 30, and includes outwardly extending locking ridges 84 which are designed to accommodate a lock cap 86 (See FIG. 11), such that ball 62 may be retained within the dome 30. Lock cap 86 includes outwardly extending tabs 88 arranged to interleaf with ridges 84. In operation, a user desiring to insert or remove ball 62 from the cursor control unit 20, may unlock and remove the lock cap 86 from the orifice 82 by simply rotating the cap such that the tabs 88 and ridges 84 no longer interleaf.

As illustrated, lock cap 86 generally has a toroidal form having a central orifice 87 of smaller diameter than cutout orifice 82. It will be apparent, that once ball 62 is inserted and retained by lock cap 86, Thus, ball 62 contacts the surface below the cursor control unit 20 and rotates in response to the movement of the unit on the surface.

As shown in FIG. 9, cable 24 is coupled to cursor control 20 through a female connector 94 which is inserted over pins 38. A switch 90 is coupled to the cable 24 through electrical connector 36, and is inserted within the retaining portion 80. A switch cap 91 forms part of the cover 25 (see FIG. 1), and is disposed above switch 90 such that the depression of the switch cap 91 forces switch 90 to electrically close, and thereby signal the computer display system that an appropriate X-Y location has been selected. As shown in FIG. 10, base 26 and cover 25 are coupled by securing both sections to one another using screws 92. Once the cover and base have been joined, ball 62 is inserted and lock cap 86 is attached as discussed above to retain the ball within the dome portion 30.

With reference to FIGS. 12 and 13, a sample quadrature output of the cursor control unit 20 is illustrated. As previously described, photo-detectors 46 are disposed such that if one detector is fully exposed by a slot of the encoder disc 54, the other detector is only partially exposed. Thus, in addition to the increments of motion of the cursor control over a surface, the direction of motion may also be determined. Assume for sake of example that the cursor control 20 is moved. As illustrated in FIG. 13, a substantially digital output signal is generated by each photo-emitter/detector combination associated with each encoder assembly. In the example shown, cursor control 20 would provide a regularly spaced output from the X channel detectors if the control 20 is moved over a surface at a constant speed along the X-axis. Similarly, if there is little movement of the control unit along the Y axis, little change will occur on the Y channels inasmuch as the Y encoder disk is not being rotated significantly (see FIG. 13). The computer display system is provided with appropriate software or hardware, for example edge detectors, to detect signal state transitions. Thus, the signals from each pair of channels may be decoded such that the X-Y direction of motion may be determined for the particular order of transition changes from each channel along an axis. Inasmuch as the particular circuitry and software used for decoding the various signals and positioning the cursor or the like on a display system will be apparent to one skilled in the art, the details of such will not be recited herein.

Referring now to FIGS. 14 and 15, a display system and method for use in conjunction with the cursor control device 20 will be described. As previously discussed, control 20 is coupled to a display system which is controlled by a computer or other equivalent circuitry. Appropriate programming of the computer is provided such that a "menu" bar 100 comprising a variety of command options indicated by titles (for example, $T_1, T_2, T_3 \ldots T_n$), is displayed across the CRT screen or the like as shown in FIG. 14. If a particular title (for example T_1) is selected, one or more sub-command items 104 are displayed by the computer system below the primary menu title. As illustrated, the sub-command items appear to the user to be "pulled down" from the main menu bar 100. The user then selects a desired item for execution by the computer by appropriate movement of a cursor control, as will be described. Although the list of items 104 are shown for illustration below menu title options T_1, T_2, and T_3, in the present embodiment only one menu option may be pulled down and displayed at a time.

The sequence of operations executed by the computer system to permit the user to select a particular menu title and subcommand item is shown in FIG. 15. The computer initially displays menu bar 100 on the display system as shown in FIG. 14. A user desiring to select a particular title moves cursor control unit 20 over a surface, thereby rotating ball 62 within dome 30 and sending signals indicative of X-Y locations to the display system for corresponding movement of a cursor or the like on the display screen. Once the cursor is positioned over (or in proximity with) the chosen menu title selection, the user depresses switch cap 91 on cursor control 20, thereby activating switch 90, and signaling the computer system that the particular title has been selected. The computer display system then either executes the menu title if it is an immediate command, or displays a set of sub-command items for user selection. If items are displayed, the user continues to depress switch cap 91, and once again moves the cursor control over the surface until the displayed cursor lies over or in proximity with the item to be executed. The user then removes pressure from the switch cap 91 thereby deactivating switch 90, and indicating to the computer which item is to be executed.

The computer system then determines if further parameters are required to be specified by the user. If no further data is required, the computer executes the item indicated by the cursor position on the display screen. However, if parameters must be specified by the user prior to execution a "dialogue box" is defined on the display system which displays the various data selections which are required. For example, a user may be required to select page formats, specify numerical values, etc. In the present embodiment, a user inputs the desired data selections by positioning the cursor over the selection, in for example a multiple choice format, and momentarily activates the switch 90 on the cursor control unit. Once the required selections are made, the computer proceeds to execute the chosen menu item.

Accordingly, it is possible for a user to select and execute a variety of commands without the necessity of inputting characters on a keyboard, as is commonly required in the art. Rather, the present invention permits fast entry and execution of commands, such as for example in a word processing system or the like, wherein large blocks of text or other data may be manipulated or operated upon simply by movement of the cursor control 20 over a surface and the appropriate depression of switch 90.

Thus, an improved cursor control and display system has been described. The present invention permits a user to select desired menu titles on a menu bar by movement of a cursor control over a surface. Sub-command items may be specified for execution by the computer control display system in the same manner, such that the operator need not enter command characters on a keyboard or the like in order to access and execute most system functions.

Although the present invention has been described with reference to FIGS. 1-15 and with emphasis on a "pull down" type display system, it should be understood that the figures are for illustration only and should not be taken as limitations upon the invention. It is contemplated that many changes and modifications may be made, by one of ordinary skill in the art, to the materials and arrangements of the elements of the invention without department from the spirit and scope of the invention as disclosed above.

What is claimed is:

1. A device for providing signals indicative of X-Y locations on a display system or the like, comprising:

a housing including a base having an opening for the passage of a rotatable ball;

a unitary frame disposed on said base including:

a domed portion integrally formed with said frame substantially surrounding and retaining said rotatable ball;

said domed portion having first and second cutouts through said dome disposed substantially at 90 degrees with respect to one another, and a third cut-out disposed at an angle with respect to said first and second cut-outs;

X-Y position indicating means passing through said first and second cut-outs, for converting the rotation of said ball into signals indicative of X-Y positions on said display system;

biasing means passing through said third cut-out, for biasing said ball against said X-Y position indicating means;

means for removing said ball from said domed portion through said opening in said base, such that said ball and the interior or said dome may be serviced, said means for removing comprising:

outwardly extending lock ridges integrally formed with said opening in said base;

a lock cap having a second opening of smaller diameter then said base opening to permit only a portion of said ball to pass therethrough and contact said surface;

said lock cap further including outwardly extending lock tabs to interleaf with said lock ridges, such that rotation of said cap interleafs with said tabs and ridges thereby locking said cap onto said base;

whereby movement of said device over a surface such that a portion of said ball is maintained in contact with said surface results in X-Y positions defined on said display system.

2. The device as defined by claim 1, wherein said biasing means comprises a wheel carried by a shaft, said shaft being biased such that said wheel is maintained in contact with said ball.

3. The device as defined by claim 2, wherein said third cut-out is disposed generally at 45 degrees with respect to said first and second cut-outs.

4. The device as defined by claim 3, wherein said X-Y position indicating means includes a roller shaft coupled to an encoder disc having a plurality of radially disposed slots, said disc being disposed between a photoemitter and photo-detector.

5. The device as defined by claim 4, wherein said photo-detector is disposed within a detector aperture, said aperture being retained on said unitary frame to form an integral unit.

6. The device as defined by claim 5, further including a circuit board disposed between said frame and said base.

7. The device as defined by claim 6, further including a switch coupled to said circuit board to specify selected X-Y positions on said display system.

8. The device as defined by claim 7, said device being coupled to a computer controlled display system wherein menu commands are displayed and selected by a user through movement of said device.

9. A computer controlled display system having a display wherein a plurality of command options are displayed along a menu bar and sub-command items corresponding to each option are displayed once said option has been selected, comprising:

first display means coupled to said computer for generating and displaying said menu bar comprising said plurality of command options;

cursor control means coupled to said display system for selectively positioning a cursor on said display, said cursor control means including a cursor control device for movement over a surface, the movement of said cursor control device over said surface by a user resulting in a corresponding movement of said cursor on said display;

signal generation means including a switch having a first and second position coupled to said display system for signalling said computer of an option choice once said cursor is positioned over a first predetermined area on said display corresponding to an option to be selected, said user placing said switch in said second position while moving said cursor control device over said surface such that said cursor is over said first predetermined area;

second display means coupled to said computer for generating and displaying said sub-command items corresponding to said selected option;

said switch being placed in said first position by said user once said user has positioned said cursor over a second predetermined area corresponding to a sub-command item to be selected;

whereby an option and a sub-command item is selected and executed by said computer.

10. The display system of claim 9 wherein said cursor control device comprises:

a housing including a base having an opening for the passage of a rotatable ball;

a unitary frame disposed on said base including:

a domed portion integrally formed with said frame substantially surrounding and retaining said rotatable ball;

said domed portion having first and second cutouts through said dome disposed substantially at 90 degrees with respect to one another, and a third cut-out disposed at an angle with respect to said first and second cut-outs;

X-Y position indicating means passing through said first and second cut-outs, for converting the rotation of said ball into signals indicative of X-Y positions on said display system;

biasing means passing through said third cut-out, for biasing said ball against said X-Y position indicating means;

"APPLE_PAT_4_464_652_13" 309 KB 2000-02-21 dpi: 300h x 300v pix: 1919h x 2897v

means for removing said ball from said domed portion through said opening in said base, such that said ball and the interior of said dome may be serviced, said means for removing said ball comprising:

outwardly extending lock ridges integrally formed with said opening in said base;

a lock cap having a second opening of smaller diameter then said base opening to permit only a portion of said ball to pass therethrough and contact said surface;

said lock cap further including outwardly extending lock tabs to interleaf with said lock ridges, such that rotation of said cap interleafs with said tabs and ridges thereby locking said cap onto said base;

whereby said option and sub-command item may be selected by movement of said cursor control means over a surface such that a portion of said ball is in contact with said surface.

11. In a computer controlled display system having a display wherein a plurality of command options are displayed along a menu bar and sub-command items corresponding to each option are displayed once said option has been selected, a method for selecting an option and an item, comprising the steps of:

(a) generating and displaying said menu bar comprising said plurality of command options;

(b) positioning a cursor on said display using a cursor control device for movement over a surface, the movement of said cursor control device over said surface by a user resulting in a corresponding movement of said cursor on said display;

(c) signalling said computer of an option choice once said cursor is positioned over a first predetermined area on said display corresponding to an option to be selected, said user signalling said computer by placing a switch coupled to said display system in a second position while moving said cursor control device over said surface such that said cursor is over said first predetermined area;

(d) generating and displaying said sub-command items corresponding to said selected option;

(e) positioning said cursor over a second predetermined area corresponding to a sub-command item to be selected, said switch being maintained in said second position until said cursor is positioned over said second predetermined area;

(f) placing said switch in a first position once said user has positioned said cursor over said second predetermined area;

whereby an option and an item associated with said option is selected.

12. The method as defined by claim 11, wherein said switch is disposed on said cursor control device.

13. The method as defined by claim 12, where said computer displays said sub-command items generally below said optiion on said menu bar.

* * * * *

United States Patent [19]

Atkinson

[11] E **Patent Number: Re. 32,632**

[45] **Reissued Date of Patent: Mar. 29, 1988**

[54] **DISPLAY SYSTEM**

[75] Inventor: **William D. Atkinson,** Los Gatos, Calif.

[73] Assignee: **Apple Computer, Inc.,** Cupertino, Calif.

[21] Appl. No.: **811,372**

[22] Filed: **Dec. 20, 1985**

Related U.S. Patent Documents

Reissue of:
[64] Patent No.: **4,464,652** ✓
Issued: **Aug. 7, 1984**
Appl. No.: **399,704**
Filed: **Jul. 19, 1982**

U.S. Applications:
[62] Division of Ser. No. 399,704, Jul. 19, 1982, Pat. No. 4,464,652.

[51] **Int. Cl.⁴** **G09G 1/16**
[52] **U.S. Cl.** **340/709;** 340/710; 340/706; 340/721
[58] **Field of Search** .... 340/706, 709, 710, 711, 340/712, 809, 810, 716, 870.28, 870.29; 178/18, 19; 74/471 XY; 358/183

[56] **References Cited**

U.S. PATENT DOCUMENTS

3,395,589	8/1968	Gersten	74/198
3,541,541	11/1970	Englebart	340/710
3,625,083	12/1971	Bose	74/471 XY
3,835,464	9/1974	Kider	340/710
3,987,685	10/1976	Opocensky	340/710
4,232,311	11/1980	Agneta	340/709
4,245,244	1/1981	Lijewski et al.	358/183
4,310,839	1/1982	Schwerdt	340/709
4,369,439	1/1983	Broos	340/710
4,404,865	9/1983	Kim	74/471 XY
4,451,895	5/1984	Sliwkowski	340/707

FOREIGN PATENT DOCUMENTS

1526428	9/1978	United Kingdom	340/710

OTHER PUBLICATIONS

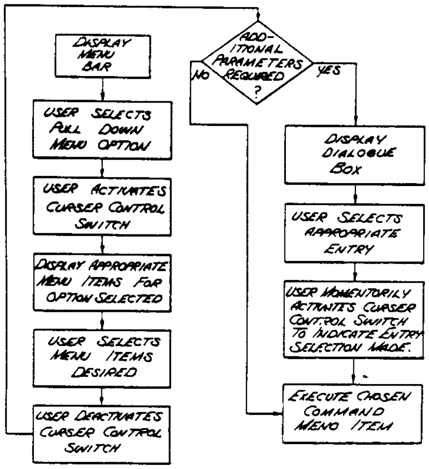
"The Smalltalk Environment", BYTE, Aug. 1981, p. 90, Larry Teslor.
"A Display Oriented Programmer's Assistant", *Int. J. Man-Machine Studies,* 1979, Teitleman.
"A Tour Through Cedar", *IEEE Software,* Apr., 1984, Teitleman.
"Xerox's 'Star'", *Seybold Report,* vol. 10, No. 16, Apr. 27, 1981.
"Star Graphics: An Object Oriented Implementation", *Computer Graphics,* Jul., 1982, Lipkie et al.

Primary Examiner—Gerald L. Brigance
Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman

[57] **ABSTRACT**

A cursor control device having particular application to a computer display system is disclosed. The cursor control includes a unitary frame, having a domed portion substantially surrounding and retaining a ball which is free to rotate. X-Y position indicating means are provided, such that rotation of the ball provides signals indicative of X-Y positions on the display system. The ball is free to "float" in the vertical direction within the dome, and thereby maintain good surface contact. X-Y positions are established by movement of the control device over a surface. A display system and method is disclosed for use in conjunction with the cursor control device, which permits a user to select command options simply by movement of the displayed cursor over a "pull-down" menu bar.

4 Claims, 15 Drawing Figures


DISPLAY MENU BAR
USER SELECTS PULL DOWN MENU OPTION
USER ACTIVATES CURSER CONTROL SWITCH
DISPLAY APPROPRIATE MENU ITEMS FOR OPTION SELECTED
USER SELECTS MENU ITEMS DESIRED
USER DEACTIVATES CURSER CONTROL SWITCH
ADDITIONAL PARAMETERS REQUIRED ?
NO
YES
DISPLAY DIALOGUE BOX
USER SELECTS APPROPRIATE ENTRY
USER MOMENTORILY ACTIVATES CURSER CONTROL SWITCH TO INDICATE ENTRY SELECTION MADE.
EXECUTE CHOSEN COMMAND MENO ITEM

Lisa

Mouse

Pull-down menus

"APPLE_PAT_4_464_652_A_01" 190 KB 2000-02-23 dpi: 300h x 300v pix: 1948h x 2970v

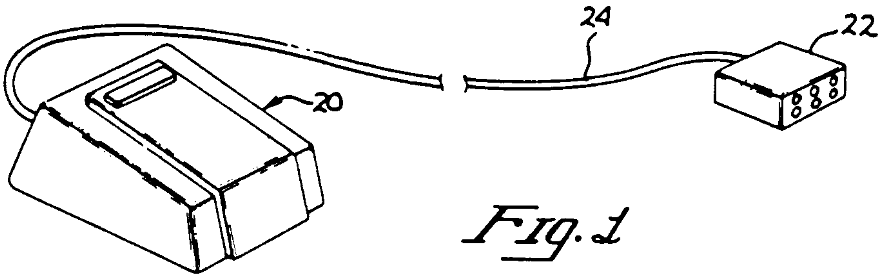


Fig. 1

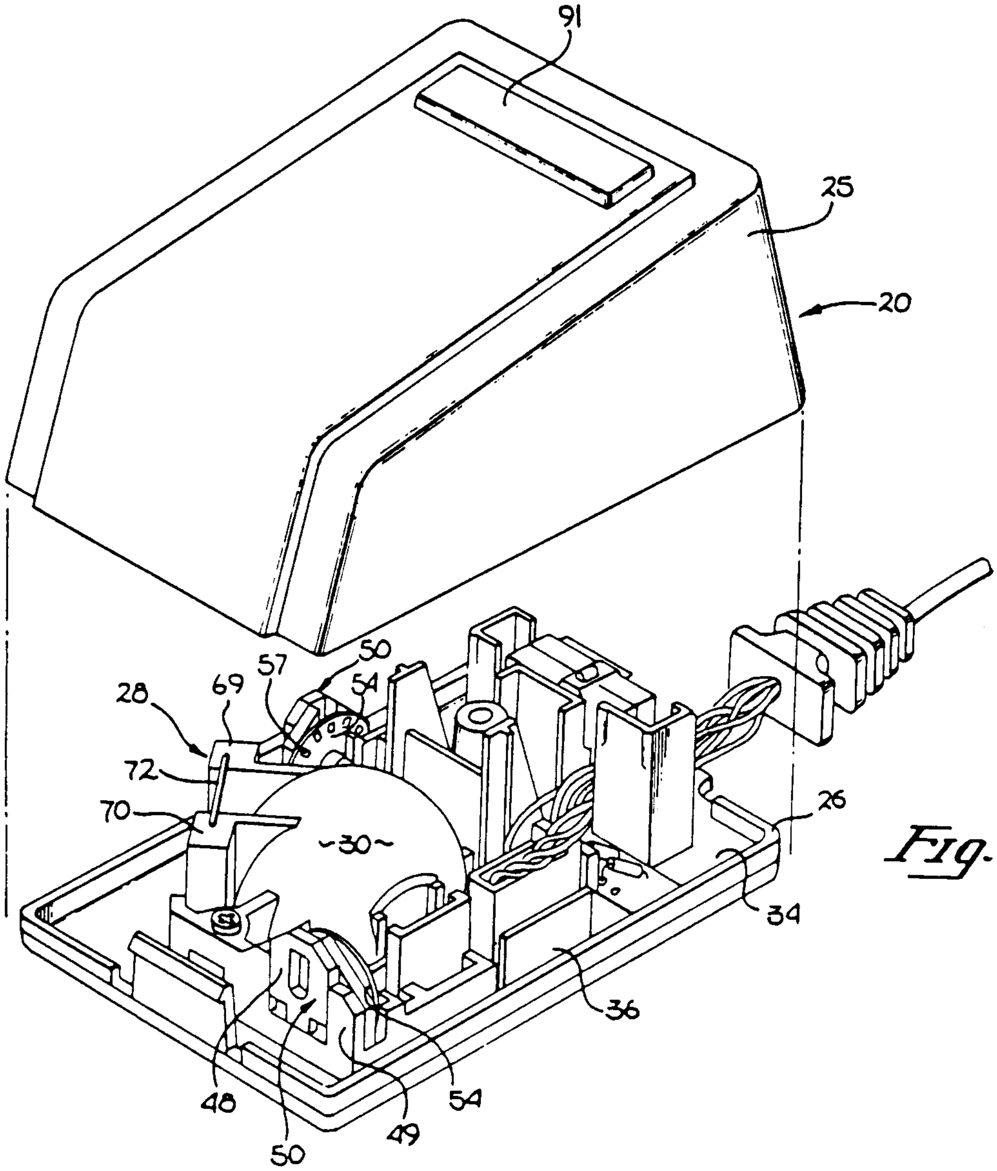


Fig. 2

Fig. 3

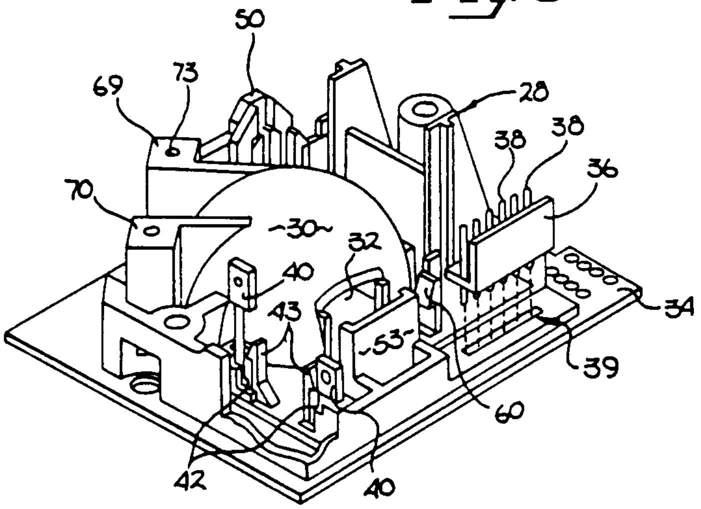
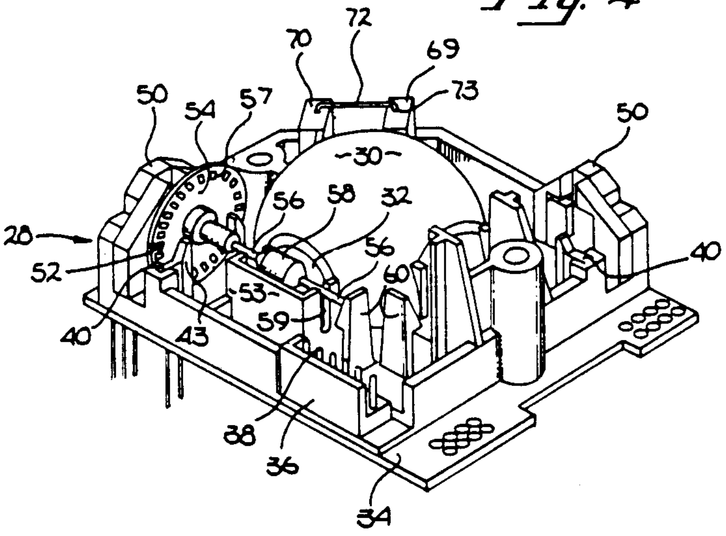


Fig. 4



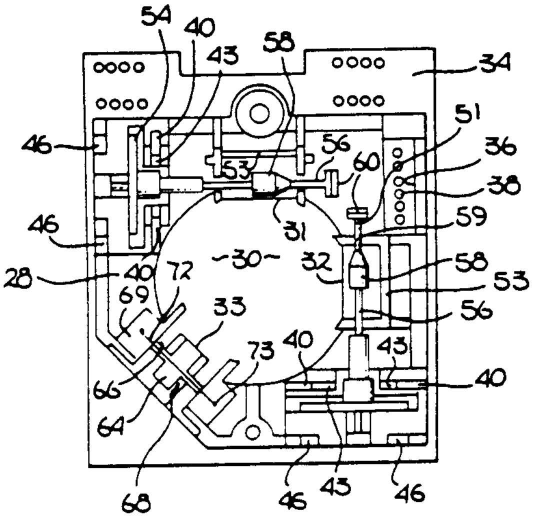


Fig. 5

"APPLE_PAT_4_464_652_A_03" 141 KB 2000-02-23 dpi: 300h x 300v pix: 1879h x 2700v

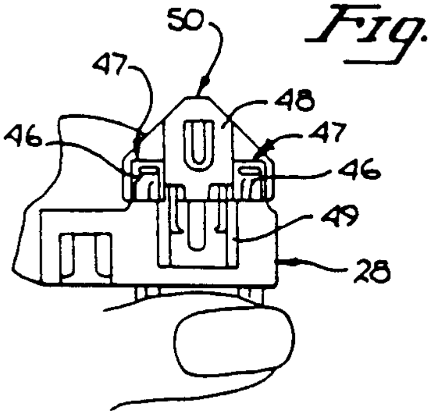


Fig. 6

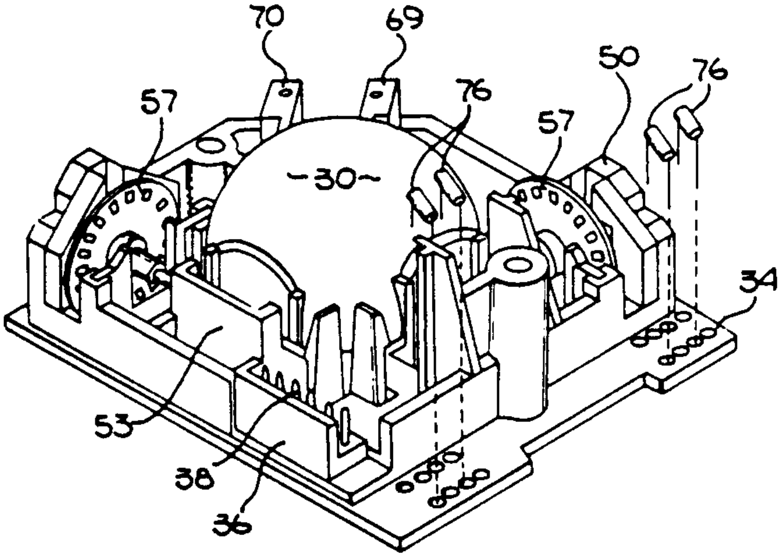


Fig. 7

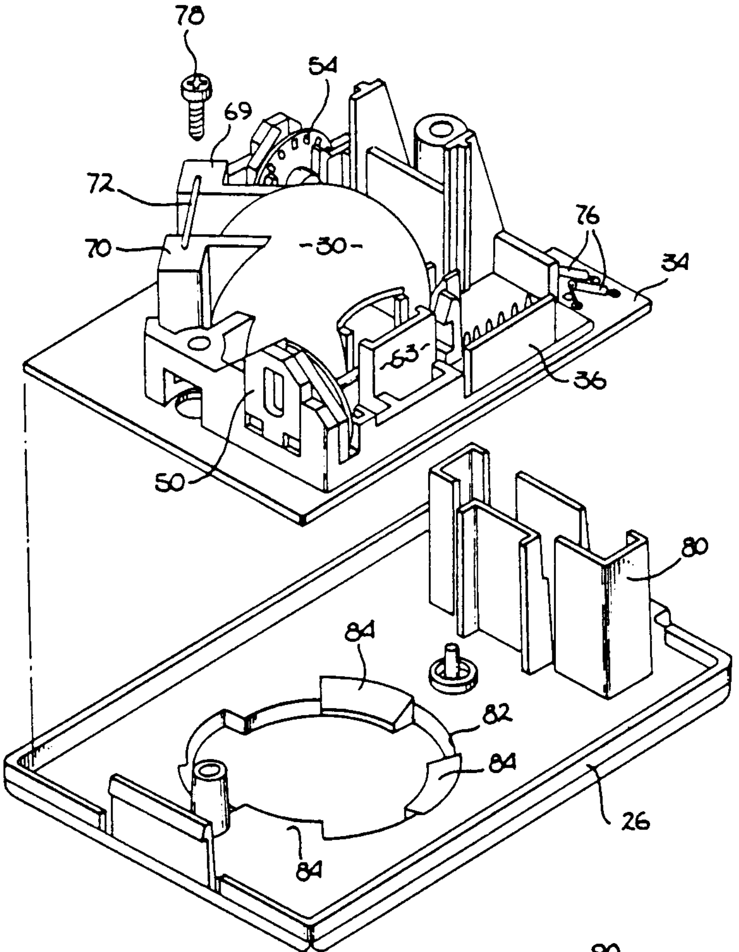

78
54
69
72
70
-30-
76
34
63
36
50
80
84
82
84
26
84

Fig. 8

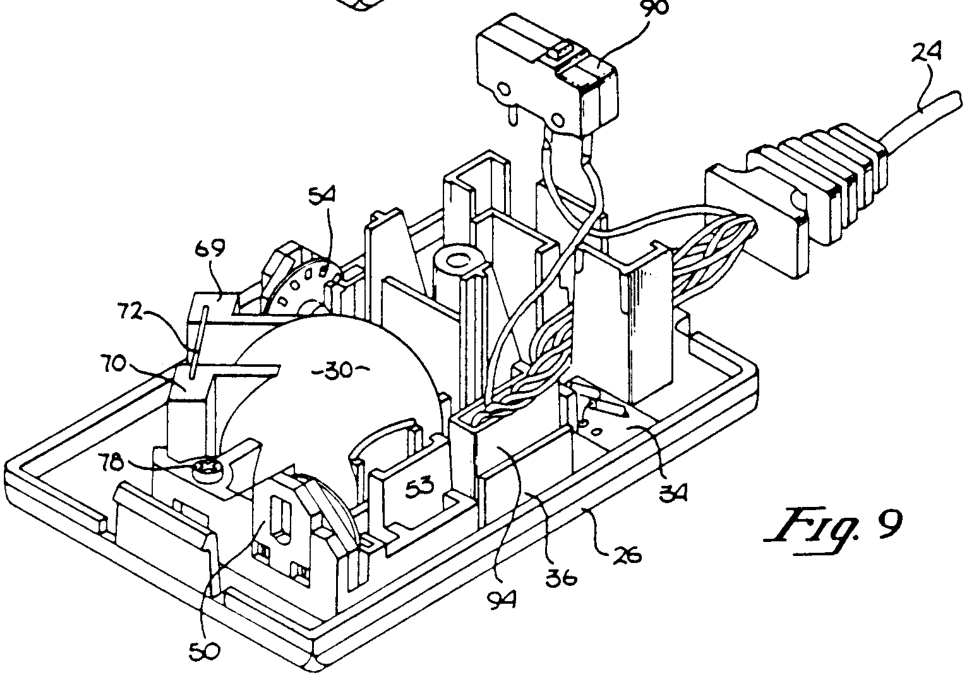

90
24
54
69
72
70
-30-
78
53
34
26
36
94
50

Fig. 9

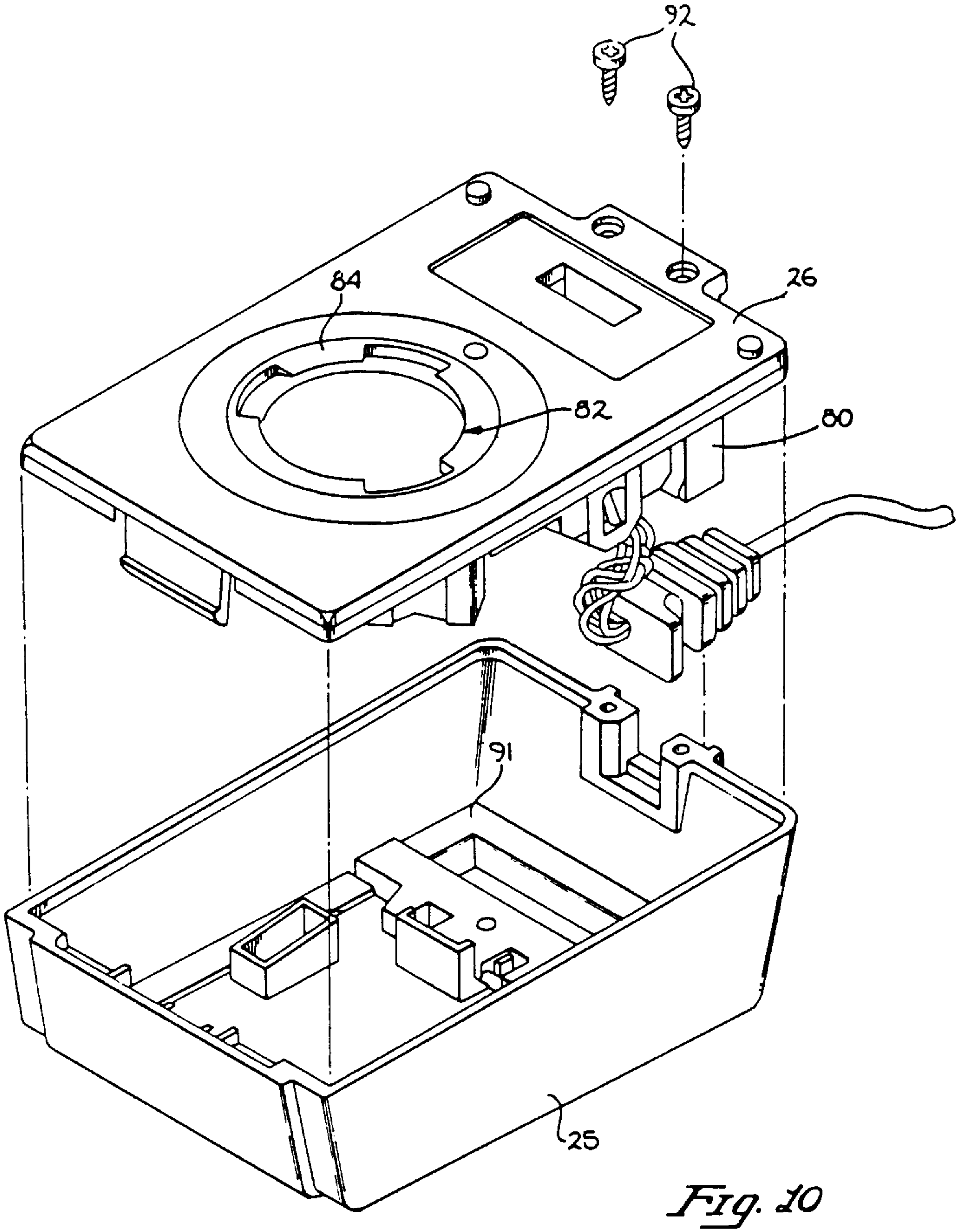


Fig. 10

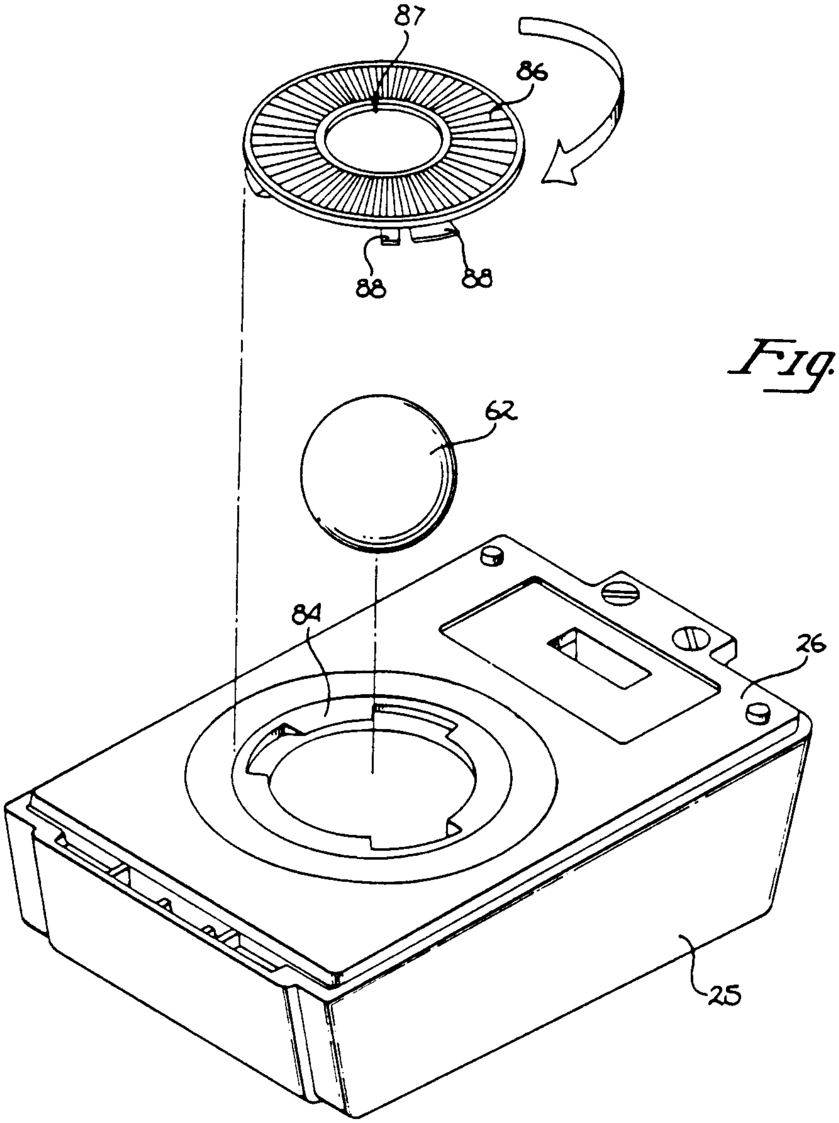


Fig. 11

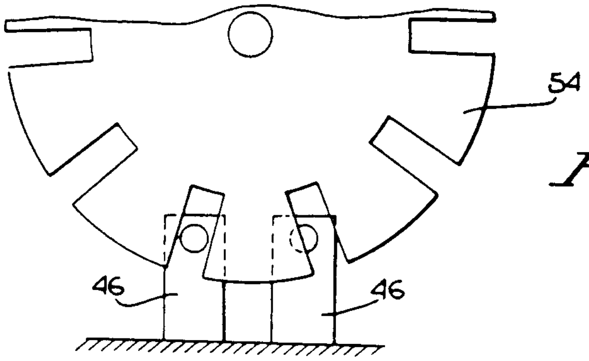


Fig. 12

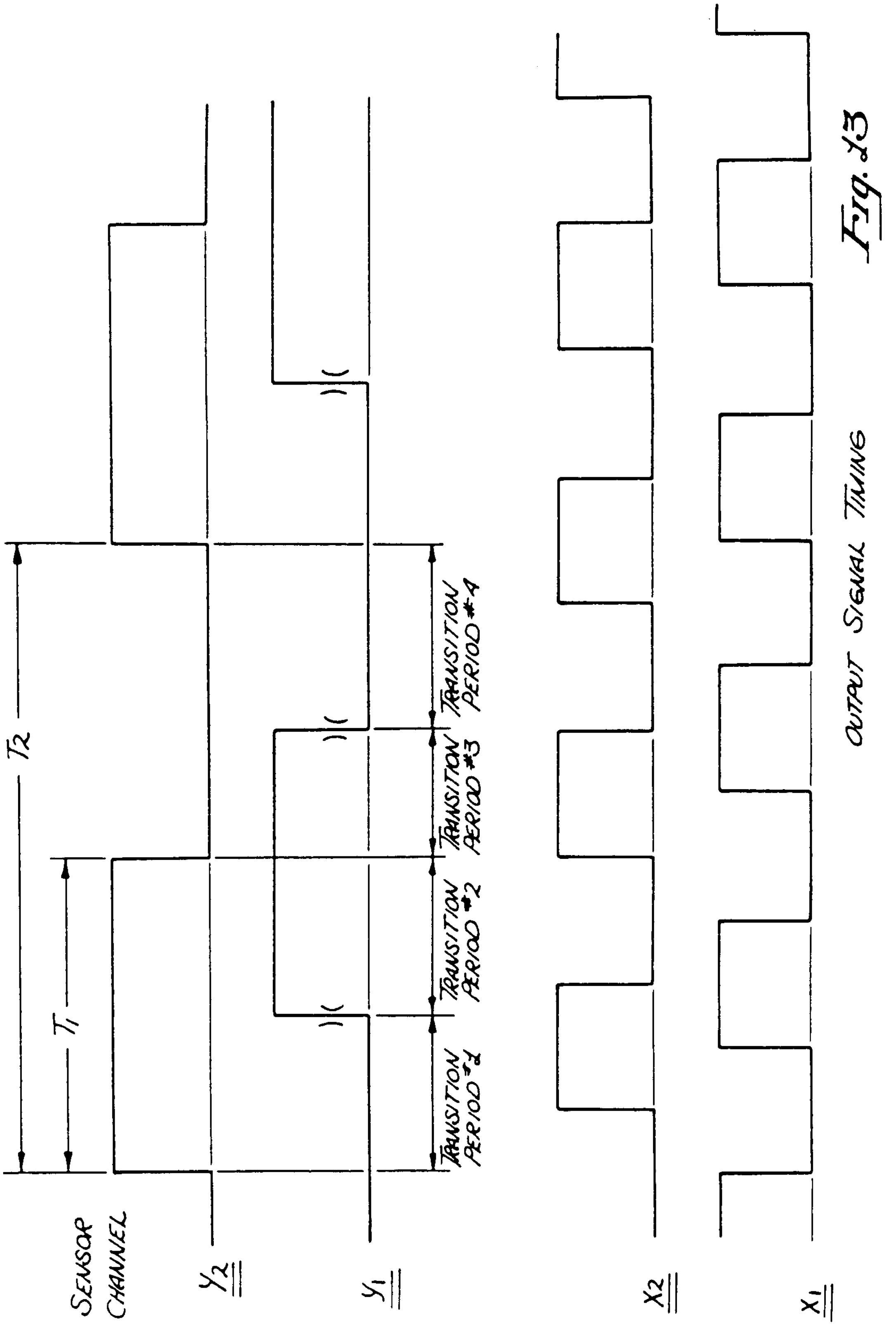

SENSOR CHANNEL
Y2
Y1
X2
X1
T2
T1
TRANSITION PERIOD #1
TRANSITION PERIOD #2
TRANSITION PERIOD #3
TRANSITION PERIOD #4
OUTPUT SIGNAL TIMING

Fig. 13

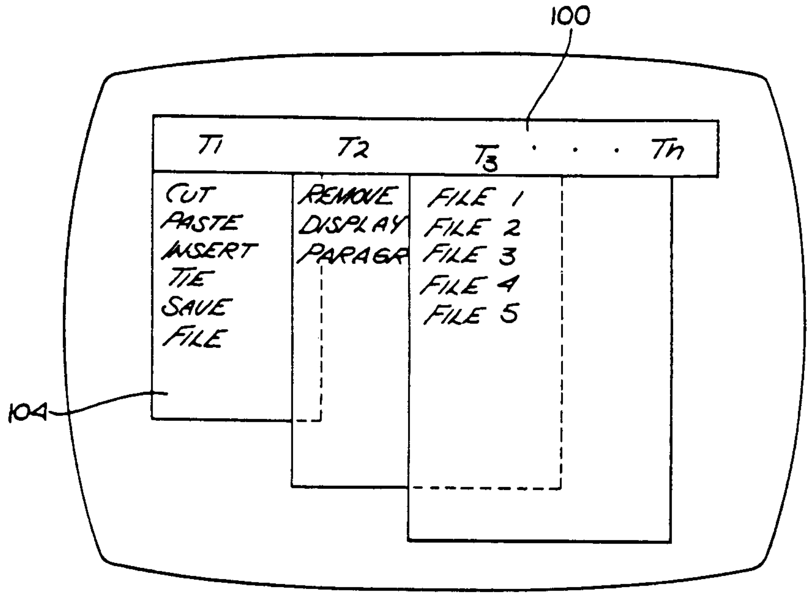

100
T1
T2
T3
. . .
Tn
CUT
PASTE
INSERT
TIE
SAVE
FILE
REMOVE
DISPLAY
PARAGR
FILE 1
FILE 2
FILE 3
FILE 4
FILE 5
104

Fig. 14

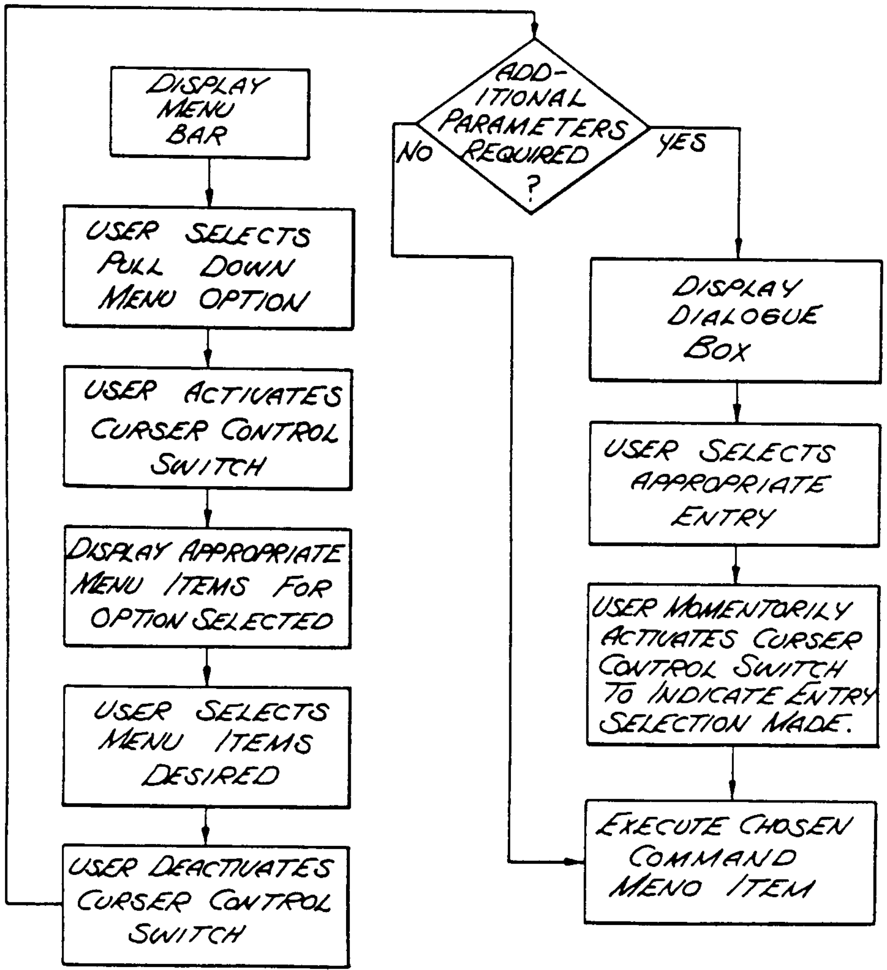

DISPLAY MENU BAR
USER SELECTS PULL DOWN MENU OPTION
USER ACTIVATES CURSER CONTROL SWITCH
DISPLAY APPROPRIATE MENU ITEMS FOR OPTION SELECTED
USER SELECTS MENU ITEMS DESIRED
USER DEACTIVATES CURSER CONTROL SWITCH
ADDITIONAL PARAMETERS REQUIRED ?
NO
YES
DISPLAY DIALOGUE BOX
USER SELECTS APPROPRIATE ENTRY
USER MOMENTORILY ACTIVATES CURSER CONTROL SWITCH TO INDICATE ENTRY SELECTION MADE.
EXECUTE CHOSEN COMMAND MENO ITEM

Fig. 15

DISPLAY SYSTEM

Matter enclosed in heavy brackets [ ] appears in the original patent but forms no part of this reissue specification; matter printed in italics indicates the additions made by reissue.

This application is a divisional reissue of application Ser. No. 06/399704, filed Jul. 19, 1982, now U.S. Pat. No. 4,464,652.

BACKGROUND OF THE INVENTION

1. Field

The present invention relates to the field of display systems, and more particularly to devices which can position a cursor over selected locations on a computer controlled display.

2. Art Background

In many computer controlled display systems, it is desirable to allow the user to control the position of a cursor or the like by means which are external from the main computer keyboard. For example, a user may be required to repetitively choose software options displayed on a cathode ray tube (CRT), or may desire to input data in a diagram format into the computer system. In such situations traditional keyboard input systems are not as effective as a cursor control device commonly referred to as a "mouse".

In a typical "mouse" system, a hand-held transducer provides positional movement signals to the display system. Traditionally, the movement of wheels within the cursor control device are coupled to potentiometers to provide signals indicative of an X-Y position on the display screen (see U.S. Pat. Nos. 3,541,541; 3,269,190; and 3,835,464). Other mouse systems utilize rotating balls on wheels which are in turn coupled to rotate apertures interrupting beams of light, thereby providing positional signals to the display system (see U.S. Pat. Nos. 3,892,963 and 3,541,521).

One common disadvantage of cursor control devices found in the prior art is their cost. Typically, prior art cursor controls include costly mechanical parts which require precise alignment for proper operation. Moreover, it is not uncommon for these devices to exhibit a loss in accuracy over time as the mechanism wears. As computer display capabilities have become more advanced in terms of user real-time graphic interaction, cursor control devices have become a necessity in many computer systems. Accordingly, there exists a need to provide a cost effective, simple and highly reliable cursor control device for providing signals indicative of X-Y positions on a computer display system.

As will be disclosed below, the present invention provides an improved cursor control device which overcomes the disadvantages of the prior art by utilizing a unitary frame structure for accurate alignment of all elements and simple assembly, as well as photo-optics to provide the required positional signals. In addition, a display system and method is disclosed for use in association with the cursor control device which permits a user to select command options simply by movement of the cursor over a "pull-down" menu bar.

SUMMARY OF THE INVENTION

A cursor control device having particular application to computer display systems is disclosed. The cursor control includes a unitary frame having a domed portion which houses a ball which is free to rotate. Two encoder disc assemblies are provided, which include roller shafts disposed substantially 90 degrees relative to one another and in contact with the ball. Each roller shaft is coupled to an encoder disc having a plurality of slots disposed radially around the disc periphery. These slots interrupt light beams which are provided by photo-emitters and directed at photo-detectors. Each slotted disc interrupts two light beams which are arranged such that when one beam is fully transmitted, the other is partially blocked. Beam interruptions produce signal pulses representing increments of motion, while the order in which the light beams are interrupted indicates the direction of motion, thereby resulting in an X-Y position on a display system. The ball is maintained in contact with the roller shafts by a spring biased idler wheel. The ball is free to "float" in the vertical direction within the dome, and thereby maintain good surface contact. Moreover, the ball may be easily removed for cleaning to insure that any build up of lint or the like does not prevent the ball from rotating smoothly. A switch is provided within the cursor control housing in order to signal the display system that a desired X-Y location on the display screen has been selected. In operation, a user may selectively position a cursor or the like on a display system by simply moving the cursor control device over a surface, such as a desk, until the desired cursor position is shown on the display device. A display system and method is disclosed for use in conjunction with the cursor control device, which permits user to select command options simply by movement of the displayed cursor over a "menu bar".

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a perspective view of the present invention.

FIG. 2 is a perspective of the present invention illustrating the cursor control device as it appears without the housing cover.

FIG. 3 is a perspective view of the unitary frame of the present invention coupled to the printed circuit board base, illustrating the placement of photo-detectors and the coupling connector.

FIG. 4 is a further perspective view of the unitary frame and circuit boad of FIG. 3 illustrating the position of a roller shaft and encoder wheel.

FIG. 5 is a top view of the unitary frame and printed circuit board of the present invention.

FIG. 6 is a partial view of the unitary frame in FIG. 3, illustrating the insertion of a detector aperture.

FIG. 7 is a perspective view of the unitary frame of FIG. 3, illustrating the placement of resistors on the printed circuit board.

FIG. 8 is a perspective view of the coupling of the unitary frame cage and printed circuit board combination to the housing base of the present invention.

FIG. 9 is a perspective view illustrating the placement of the control switch within the housing base.

FIG. 10 is the perspective view of the final assembly of the present invention illustrating the coupling of the cover and base portions of the housing.

FIG. 11 is a perspective view illustrating the insertion or removal of the floating and rotating ball.

FIG. 12 is a diagrammatical illustration of the alignment of the photo-emitters in relation to each encoder disc.

FIG. 13 is a diagrammatical illustration of a sample quadrature output of the present invention indicative of X-Y locations on display system.

FIG. 14 is a diagrammatical illustration of a "pull down" menu bar display.

FIG. 15 is a block diagram illustrating the sequence of steps utilized by the present invention to display options and associated commands on a "pull-down" menu bar display.

DETAILED DESCRIPTION OF THE INVENTION

A cursor control device having particular application for use in conjunction with a computer display system is disclosed. In the following description for purposes of explanation, specific numbers, materials and configurations are set forth in order to provide a thorough understanding of the present invention. However, it will be apparent to one skilled in the art that the present invention may be practiced without the specific details. In other instances, well known systems are shown in diagrammatical or block diagram form in order not to obscure the present invention unnecessarily.

Referring now to FIG. 1, the present invention includes a hand held cursor control unit **20** which is coupled to a plug **22** by means of a cable **24**. As best illustrated in FIG. 2, cursor control unit **20** includes a cover **25** and a base **26** upon which the internal workings of the present invention are disposed. As will be apparent from the discussion which follows, cursor control unit **20** is designed with ease of assembly in mind, while providing very close tolerances and high X-Y position location accuracy.

With reference to FIGS. 3, 4 and 5, a premolded unitary frame **28** is provided which includes a domed housing **30** presently having three cut-out locations **31**, **32** and **33**. As illustrated, cut-outs **31** and **32** are disposed substantially at 90 degrees with respect to one another, with cut-out **33** being oriented generally symmetrically opposite the other cut-outs. In addition, frame **28** includes a plurality of bosses, slots and shaped stems of material which when pertinent will be discussed in this specification. In the presently preferred embodiment, the frame **28** is comprised of a plastic material (e.g. polycarbonate) which is impregnated with a lubricant (e.g. teflon). Thus, during operation and throughout its useful life, cursor control unit **20** does not require the addition of either wet or dry lubricants. Frame **28** is mounted on a printed circuit board **34** to facilitate electrical connection between the various electrical elements within the unit. Electrical connector header **36** is mounted as shown (see FIG. 3) to the unitary frame **28** such that connector pins **38** pass through a rectangular slot **39** through the frame to the circuit board below. As will be discussed, cable **24** is electrically coupled to the cursor control unit **20** through connector **36**.

As illustrated in FIG. 3, photo-emitters **40** are inserted into slots **42** such that the emitter portion is facing away from the dome **30** (note that one emitter **40** is shown in FIG. 3 partially inserted). Upwardly extending clips **43** are snapped over portions of each emitter **40**, as shown, to prevent them from being dislodged. Similarly, two photo-detectors **46** are inserted facing the emitters **40** into slots **47** in each of two detector apertures **50**. As shown in FIG. 6, an outwardly extending portion **48** of each detector aperture **50** is aligned with guides **49** formed integrally with the frame **28**, and the aperture is then snapped downward into place. Thus, each detector aperture **50** houses two detectors **46** which face two emitters **40**, respectively. In the presently preferred embodiment, the emitter/detector combination operates within the infrared region. However, it will be appreciated that any suitable wavelength may be used in a particular application. In addition, presently, the detectors **46** incorporate integral Schmitt triggers to provide detector outputs which more closely approximate a digital signal.

Two encoder disc assemblies are provided to convert, as will be described, the movement of the cursor control unit **20** into signals indicative of X-Y locations defined on the display system. Each encoder assembly **52** includes an encoder disc **54** axially coupled to a roller shaft **56**. In addition, each encoder disc **54** is provided with a plurality of radially disposed slots **57** which interrupt the light beams generated by the photo-emitters **40**. A cylindrical contact member **58** surrounds each roller shaft **56** at each respective cut out location, as illustrated. Each encoder disc assembly **52** is mounted on the unitary frame **28** by inserting the encoder disc **54** between the detector aperture **50** and emitters **40** and snapping an end clip **60** over the opposite end of the roller shaft **56** (see FIGS. 4, 5 and 7), thereby allowing rotation of the roller shaft and encoder disc with a minimum of friction. As illustrated, each shaft **56** is slipped into and carried by a "U" shaped guide **59** formed from upwardly extending alignment bosses **53** to maintain each roller shaft **56** in proper orientation. End **51** of the shaft **56** is carried for rotation within a hollow portion of the detector aperture **50**, such that encoder disk **54** is disposed in close proximity to the aperture **50**. The present invention's use of integral lubrication within the frame material, permits each shaft **56** to freely rotate about its longitudinal axis.

As a result of the above described configuration, the radially disposed slots **57** of each encoder disc interrupt two light beams from photo-emitters **40**. The position of the emitter/detector combination and encoder disc is such that when one beam is fully transmitted, the other is partially blocked by a slit on the encoder disc. As will be discussed, in operation a ball **62** is disposed within the dome **30** of the frame, and retained such that it is maintained in contact with both cylindrical contact members **58**. The rotation of the ball **62** within the dome **30** in turn causes the rotation of each roller shaft **56** and its respective encoder disc. As will be discussed, the beam interruptions from the rotation of each encoder disc **54** produce signal pulses representing increments of motion, while the order in which the light beams are interrupted indicates the direction of motion of the cursor control unit.

Ball **62** is retained against the cylindrical contact members **58** by an idler wheel for rotation on a fixed shaft **66**, as best shown in FIG. 5. The idler wheel **64** and shaft **66** are inserted within a slot **68** formed by rectangular bosses **69** and **70** extending upwardly from the frame's base. Wheel **64** extends through cut-out **33** into the interior of the dome **30**. The legs of a staple shaped idler spring **72** are inserted through passages **73** passing perpendicular to the horizontal plane of the frame **28** and circuit board **34**, thereby retaining the shaft **66** within the slot **68**.

Referring now to FIG. 7, resistors **76**, which are required by the specific electronics of the emitter/detector combination of the present invention, are inserted into the printed circuit board **34**. The resistors **76** and associated leads from the connector **36**, photo-emitters

40, and photo-detectors 46 are then electrically connected and soldered in place as is conventionally done in the art.

With reference now to FIGS. 8, 9 and 10, the assembled frame 28 and circuit assembly is mounted on the base 26 by means of a screw 78. As illustrated, base 26 includes an upwardly extending switch retaining portion 80 and a generally circular cut-out orifice 82. As best shown in FIGS. 8, 10 and 11, circular orifice 82 is disposed substantially below the opening of dome 30, and includes outwardly extending locking ridges 84 which are designed to accommodate a lock cap 86 (See FIG. 11), such that ball 62 may be retained within the dome 30. Lock cap 86 includes outwardly extending tabs 88 arranged to interleaf with ridges 84. In operation, a user desiring to insert or remove ball 62 from the cursor control unit 20, may unlock and remove the lock cap 86 from the orifice 82 by simply rotating the cap such that the tabs 88 and ridges 84 no longer interleaf.

As illustrated, lock cap 86 generally has a toroidal form having a central orifice 87 of smaller diameter than cutout orifice 82. It will be apparent, that once ball 62 is inserted and retained by lock cap 86, Thus, ball 62 contacts the surface below the cursor control unit 20 and rotates in response to the movement of the unit on the surface.

As shown in FIG. 9, cable 24 is coupled to cursor control 20 through a female connector 94 which is inserted over pins 38. A switch 90 is coupled to the cable 24 through electrical connector 36, and is inserted within the retaining portion 80. A switch cap 91 forms part of the cover 25 (see FIG. 1), and is disposed above switch 90 such that the depression of the switch cap 91 forces switch 90 to electrically close, and thereby signal the computer display system that an appropriate X-Y location has been selected. As shown in FIG. 10, base 26 and cover 25 are coupled by securing both sections to one another using screws 92. Once the cover and base have been joined, ball 62 is inserted and lock cap 86 is attached as discussed above to retain the ball within the dome portion 30.

With reference to FIGS. 12 and 13, a sample quadrature output of the cursor control unit 20 is illustrated. As previously described, photo-detectors 46 are disposed such that if one detector is fully exposed by a slot of the encoder disc 54, the other detector is only partially exposed. Thus, in addition to the increments of motion of the cursor control over a surface, the direction of motion may also be determined. Assume for sake of example that the cursor control 20 is moved. As illustrated in FIG. 13, a substantially digital output signal is generated by each photo-emitter/detector combination associated with each encoder assembly. In the example shown, cursor control 20 would provide a regularly spaced output from the X channel detectors if the control 20 is moved over a surface at a constant speed along the X-axis. Similarly, if there is little movement of the control unit along the Y axis, little change will occur on the Y channels inasmuch as the Y encoder disk is not being rotated significantly (see FIG. 13). The computer display system is provided with appropriate software or hardware, for example edge detectors, to detect signal state transitions. Thus, the signals from each pair of channels may be decoded such that the X-Y direction of motion may be determined for the particular order of transition changes from each channel along an axis. Inasmuch as the particular circiutry and software used for decoding the various signals and positioning the cursor or the like on a display system will be apparent to one skilled in the art, the details of such will not be recited herein.

Referring now to FIGS. 14 and 15, a display system and method for use in conjunction with the cursor control device 20 will be described. As previously discussed, control 20 is coupled to a display system which is controlled by a computer or other equivalent circuitry. Appropriate programming of the computer is provided such that a "menu" bar 100 comprising a variety of command options indicated by titles (for example, T_1, T_2, T_3. . . T_n), is displayed across the CRT screen or the like as shown in FIG. 14. If a particular title (for example T_1) is selected, one or more sub-command items 104 are displayed by the computer system below the primary menu title. As illustrated, the sub-command items appear to the user to be "pulled down" from the main menu bar 100. The user then selects a desired item for execution by the computer by appropriate movement of a cursor control, as will be described. Although the list of items 104 are shown for illustration below menu title options T_1, T_2, and T_3, in the present embodiment only one menu option may be pulled down and displayed at a time.

The sequence of operations executed by the computer system to permit the user to select a particular menu title and sub-command item is shown in FIG. 15. The computer initially displays menu bar 100 on the display system as shown in FIG. 14. A user desiring to select a particular title moves cursor control unit 20 over a surface, thereby rotating ball 62 within dome 30 and sending signals indicative of X-Y locations to the display system for corresponding movement of a cursor or the like on the display screen. Once the cursor is positioned over (or in proximity with) the chosen menu title selection, the user depresses switch cap 91 on cursor control 20, thereby activating switch 90, and signaling the computer system that the particular title has been selected. The computer display system then either executes the menu title if it is an immediate command, or displays a set of sub-command items for user selection. If items are displayed, the user continues to depress switch cap 91, and once again moves the cursor control over the surface until the displayed cursor lies over or in proximity with the item to be executed. The user then removes pressure from the switch cap 91 thereby deactivating switch 90, and indicating to the computer which item is to be executed.

The computer system then determines if further parameters are required to be specified by the user. If no further data is required, the computer executes the item indicated by the cursor position on the display screen. However, if parameters must be specified by the user prior to execution a "dialogue box" is defined on the display system which displays the various data selections which are required. For example, a user may be required to select page formats, specify numerical values, etc. In the present embodiment, a user inputs the desired data selections by positioning the cursor over the selection, in for example a multiple choice format, and momentarily activates the switch 90 on the cursor control unit. Once the required selections are made, the computer proceeds to execute the chosen menu item.

Accordingly, it is possible for a user to select and execute a variety of commands without the necessity of inputting characters on a keyboard, as is commonly required in the art. Rather, the present invention permits fast entry and execution of commands, such as for example in a word processing system or the like, wherein large blocks of text or other data may be manipulated or operated upon simply by movement of the cursor control **20** over a surface and the appropriate depression of switch **90**.

Thus, an improved cursor control and display system has been described. The present invention permits a user to select desired menu titles on a menu bar by movement of a cursor control over a surface. Sub-command items may be specified for execution by the computer control display system in the same manner, such that the operator need not enter command characters on a keyboard or the like in order to access and execute most system functions.

Although the present invention has been described with reference to FIGS. **1–15** and with emphasis on a "pull down" type display system, it should be understood that the figures are for illustration only and should not be taken as limitations upon the invention. It is contemplated that many changes and modifications may be made, by one of ordinary skill in the art, to the materials and arrangements of the elements of the invention without department from the spirit and scope of the invention as disclosed above.

What is claimed is:

[**1**. A device for providing signals indicative of X-Y locations on a display system or the like, comprising:

a housing including a base having an opening for the passage of a rotatable ball;

a unitary frame disposed on said base including:

a domed portion integrally formed with said frame substantially surrounding and retaining said rotatable ball;

said domed portion having first and second cut-outs through said dome disposed substantially at 90 degrees with respect to one another, and a third cut-out disposed at an angle with respect to said first and second cut-outs;

X-Y position indicating means passing through said first and second cut-outs, for converting the rotation of said ball into signals indicative of X-Y positions on said display system;

biasing means passing through said third cut-out, for biasing said ball against said X-Y position indicating means;

means for removing said ball from said domed portion through said opening in said base, such that said ball and the interior or said dome may be serviced, said means for removing comprising:

outwardly extending lock ridges integrally formed with said opening in said base;

a lock cap having a second opening of smaller diameter then said base opening to permit only a portion of said ball to pass therethrough and contact said surface;

said lock cap further including outwardly extending lock tabs to interleaf with said lock ridges, such that rotation of said cap interleafs with said tabs and ridges thereby locking said cap onto said base;

whereby movement of said device over a surface such that a portion of said ball is maintained in contact with said surface results in X-Y positions defined on said display system.]

[**2**. The device as defined by claim **1**, wherein said biasing means comprises a wheel carried by a shaft, said shaft being biased such that said wheel is maintained in contact with said ball.]

[**3**. The device as defined by claim **2**, wherein said third cut-out is disposed generally at **45** degrees with respect to said first and second cut-outs.]

[**4**. The device as defined by claim **3**, wherein said X-Y position indicating means includes a roller shaft coupled to an encoder disc having a plurality of radially disposed slots, said disc being disposed between a photo-emitter and photo-detector.]

[**5**. The device as defined by claim **4**, wherein said photo-detector is diposed within a detector aperture, said aperture being retained on said unitary frame to form an integral unit.]

[**6**. The device as defined by claim **5**, further including a circuit board disposed between said frame and said base.]

[**7**. The device as defined by claim **6**, further including a switch coupled to said circuit board to specify selected X-Y positions on said display system.]

[**8**. The device as defined by claim **7**, said device being coupled to a computer controlled display system wherein menu commands are displayed and selected by a user through movement of said device.]

9. A computer controlled display system having a display wherein a plurality of command options are displayed along a menu bar and sub-command items corresponding to each option are displayed once said option has been selected, comprising:

first display means coupled to said computer for generating and displaying said menu bar comprising said plurality of command options;

cursor control means coupled to said display system for selectively positioning a cursor on said display, said cursor control means including a cursor control device for movement over a surface, the movement of said cursor control device over said surface by a user resulting in a corresponding movement of said cursor on said display;

signal generation means including a switch having a first and second position coupled to said display system for signalling said computer of an option choice once said cursor is positioned over a first predetermined area on said display corresponding to an option to be selected, said user placing said switch in said second position while moving said cursor control device over said surface such that said cursor is over said first predetermined area;

second display means coupled to said computer for generating and displaying said sub-command items corresponding to said selected option;

said switch being placed in said first position by said user once said user has positioned said cursor over a second predetermined area corresponding to a sub-command item to be selected;

whereby an option and a sub-command item is selected and executed by said computer.

[**10**. The display system of claim **9** wherein said cursor control device comprises:

a housing including a base having an opening for the passage of a rotatable ball;

a unitary frame disposed on said base including:

a domed portion integrally formed with said frame substantially surrounding and retaining said rotatable ball;

said domed portion having first and second cut-outs through said dome disposed substantially at 90 degrees with respect to one another, and a third cut-out disposed at an angle with respect to said first and second cut-outs;

X-Y position indicating means passing through said first and second cut-outs, for converting the rotation of said ball into signals indicative of X-Y positions on said display system;

biasing means passing through said third cut-out, for biasing said ball against said X-Y position indicating means;

means for removing said ball from said domed portion through said opening in said base, such that said ball and the interior of said dome may be serviced, said means for removing said ball comprising:

outwardly extending lock ridges integrally formed with said opening in said base;

a lock cap having a second opening of smaller diameter then said base opening to permit only a portion of said ball to pass therethrough and contact said surface;

said lock cap further including outwardly extending lock tabs to interleaf with said lock ridges, such that rotation of said cap interleafs with said tabs and ridges thereby locking said cap onto said base;

whereby said option and sub-command item may be selected by movement of said cursor control means over a surface such that a portion of said ball is in contact with said surface.]

11. In a computer controlled display system having a display wherein a plurality of command options are displayed along a menu bar and sub-command items corresponding to each option are displayed once said option has been selected, a method for selecting an option and an item, comprising the steps of:

(a) generating and displaying said menu bar comprising said plurality of command options;

(b) positioning a cursor on said display using a cursor control device for movement over a surface, the movement of said cursor control device over said surface by a user resulting in a corresponding movement of said cursor on said display;

(c) signalling said computer of an option choice once said cursor is positioned over a first predetermined area on said display corresponding to an option to be selected, said user signalling said computer by placing a switch coupled to said display system in a second position while moving said cursor control device over said surface such that said cursor is over said first predetermined area;

(d) generating and displaying said sub-command items corresponding to said selected option;

(e) positioning said cursor over a second predetermined area corresponding to a sub-command item to be selected, said switch being maintained in said second position until said cursor is positioned over said second predetermined area;

(f) placing said switch in a first position once said user has positioned said cursor over said second predetermined area;

whereby an option and an item associated with said option is selected.

12. The method as defined by claim **11**, wherein said switch is disposed on said cursor control device.

13. The method as defined by claim **12**, where said computer displays said sub-command items generally below said optiion on said menu bar.

* * * * *

United States Patent [19]

Hovey et al.

[11] E **Patent Number: Re. 32,633**

[45] **Reissued Date of Patent: Mar. 29, 1988**

[54] CURSOR CONTROL DEVICE

[75] Inventors: **Dean Hovey**, Los Altos; **James Sachs**, Menlo Park; **James Yurchenco**, Palo Alto; **William Lapson**, Cupertino, all of Calif.

[73] Assignee: **Apple Computer, Inc.**, Cupertino, Calif.

[21] Appl. No.: **811,635**

[22] Filed: **Dec. 20, 1985**

Related U.S. Patent Documents

Reissue of:

[64]	Patent No.:	**4,464,652** ✓
	Issued:	**Aug. 7, 1984**
	Appl. No.:	**399,704**
	Filed:	**Jul. 19, 1982**

U.S. Applications:

[62] Division of Ser. No. 399,704, Jul. 19, 1982, Pat. No. 4,464,652.

[51] **Int. Cl.⁴** **G09G 1/16**

[52] **U.S. Cl.** **340/710**; 340/706; 340/709; 178/18

[58] **Field of Search** .... 340/706, 711, 712, 709, 340/710, 809, 810, 716, 870.28, 870.29; 178/18, 19; 74/471 XY; 388/183

[56] **References Cited**

U.S. PATENT DOCUMENTS

3,395,589	8/1968	Gersten	74/198
3,541,541	11/1970	Englebart	340/710
3,625,083	12/1971	Bosc	74/471 XY
3,835,464	9/1974	Rider	340/710
3,987,685	10/1976	Opocensky	340/710
4,245,244	1/1981	Lijewski et al.	358/183
4,310,839	1/1982	Schwerdt	340/709
4,369,439	1/1983	Broos	340/710
4,404,865	9/1983	Kim	74/471 XY

FOREIGN PATENT DOCUMENTS

1526428	9/1978	United Kingdom	340/710

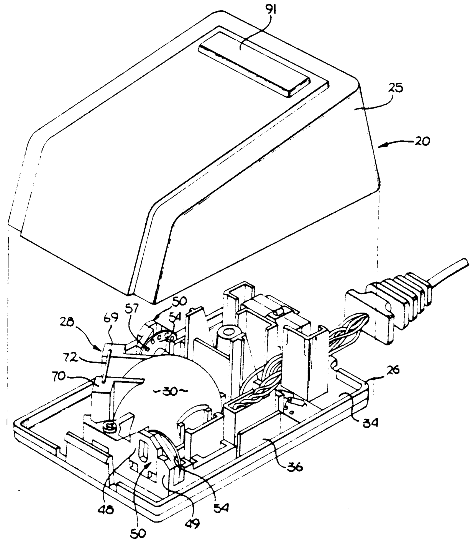
Primary Examiner—Gerald L. Brigance
Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman

[57] **ABSTRACT**

A cursor control device having particular application to a computer display system is disclosed. The cursor control includes a unitary frame, having a domed portion substantially surrounding and retaining a ball which is free to rotate. X-Y position indicating means are provided, such that rotation of the ball provides signals indicative of X-Y positions on the display system. The ball is free to "float" in the vertical direction within the dome, and thereby maintain good surface contact. X-Y positions are established by movement of the control device over a surface. A display system and method is disclosed for use in conjunction with the cursor control device, which permits a user to select command options simply by movement of the displayed cursor over a "pull-down" menu bar.

8 Claims, 15 Drawing Figures

Lisa mouse



Mouse itself

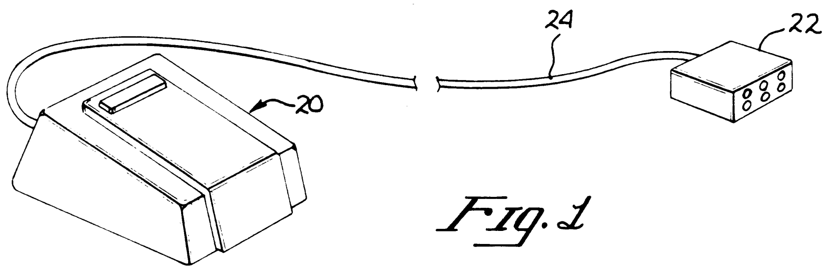


Fig. 1

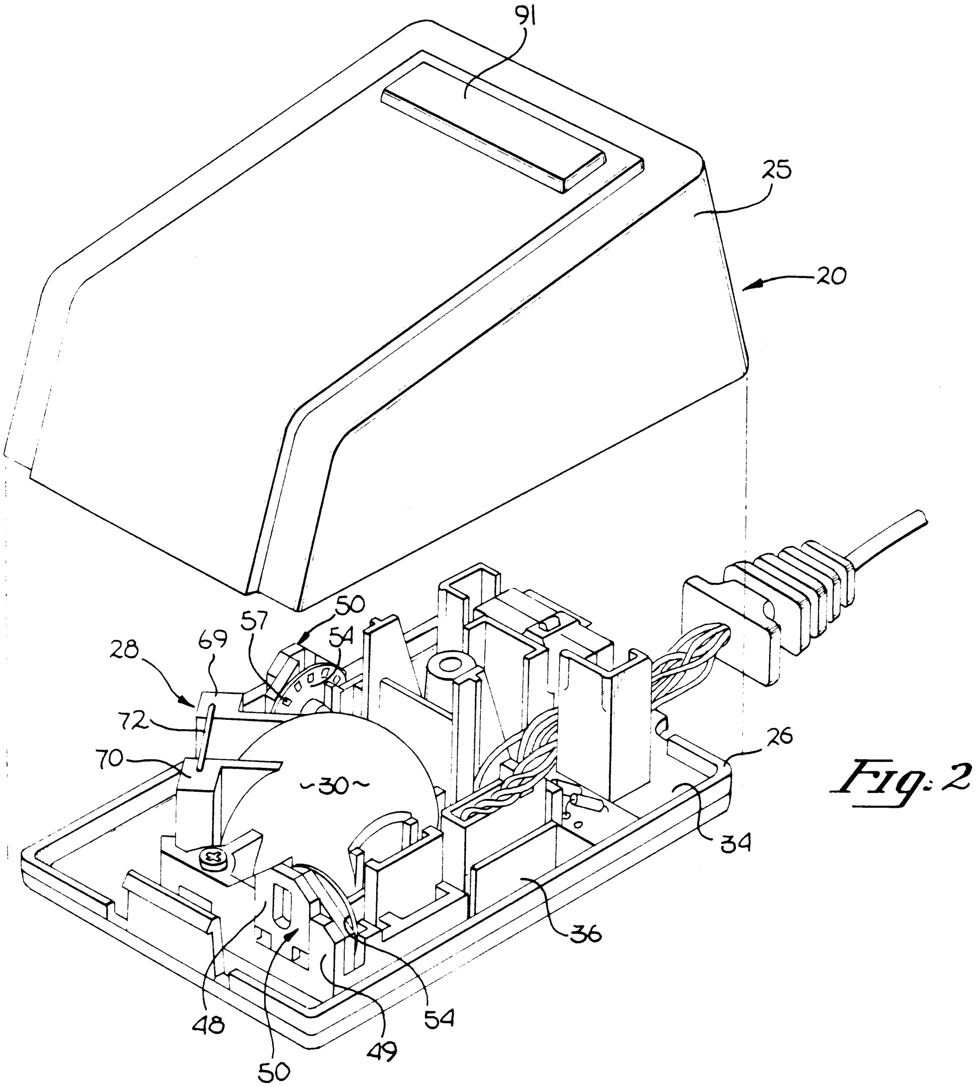


Fig. 2

Fig. 3

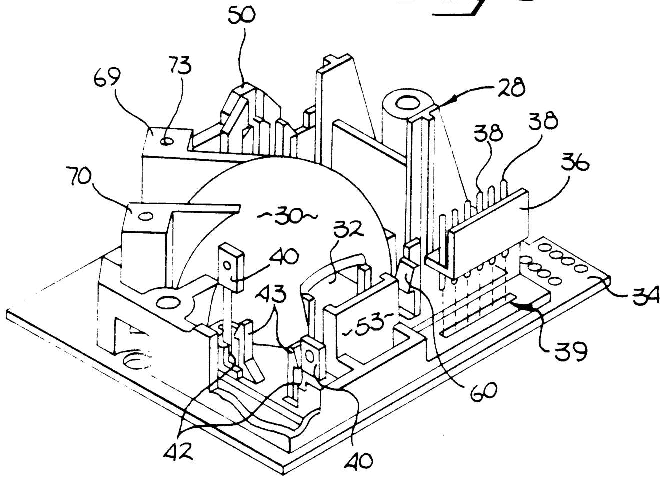
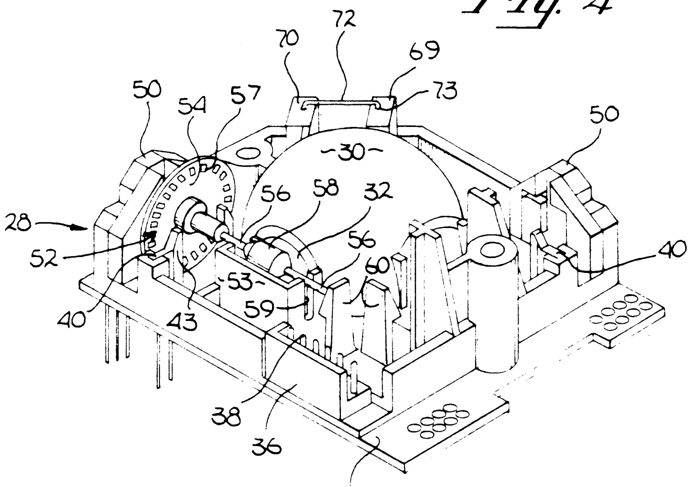


Fig. 4



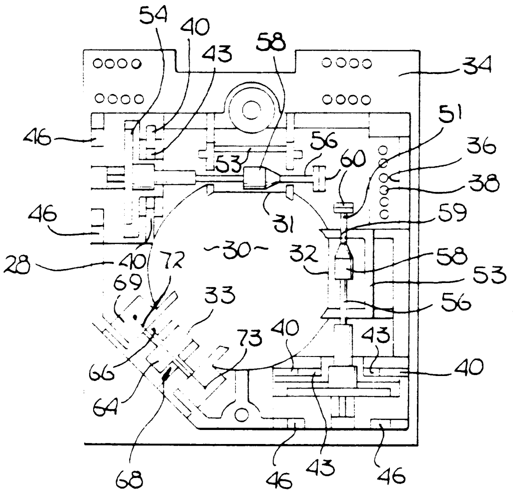
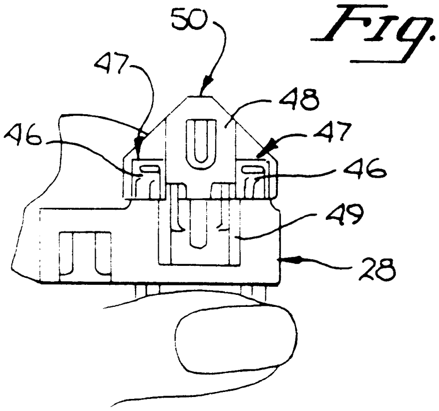


Fig. 5

Fig. 6



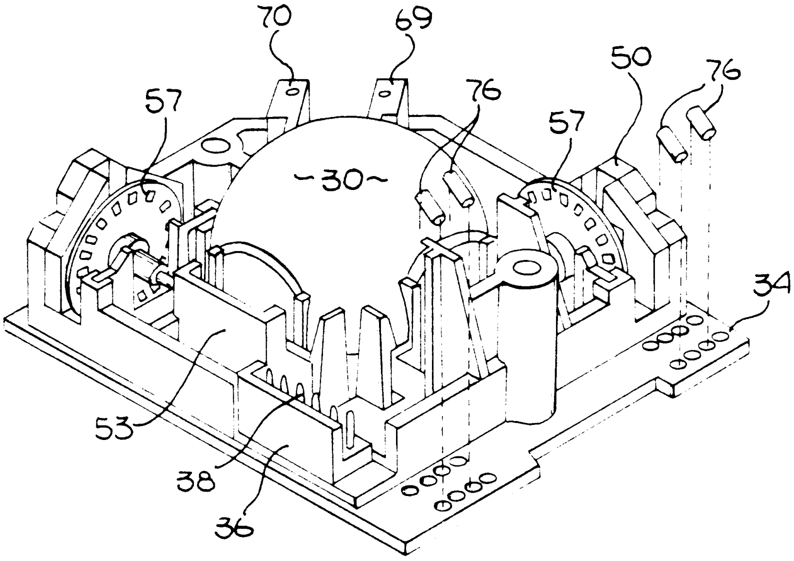


Fig. 7

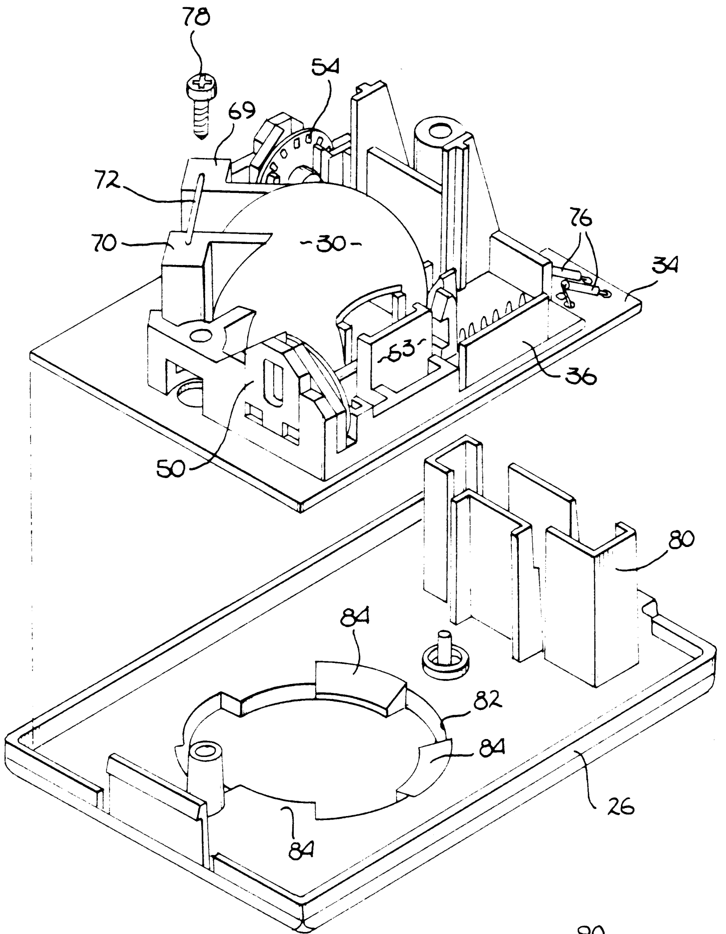

78
54
69
72
70
-30-
76
34
-53-
36
50
80
84
82
84
26
84

Fig. 8

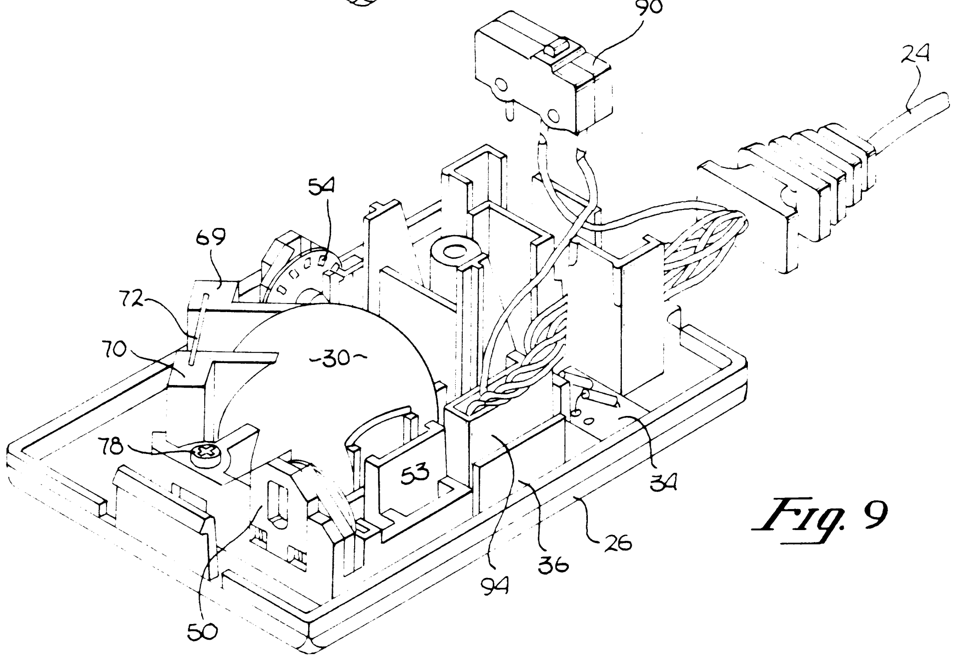

90
24
54
69
72
70
-30-
78
53
34
26
36
94
50

Fig. 9

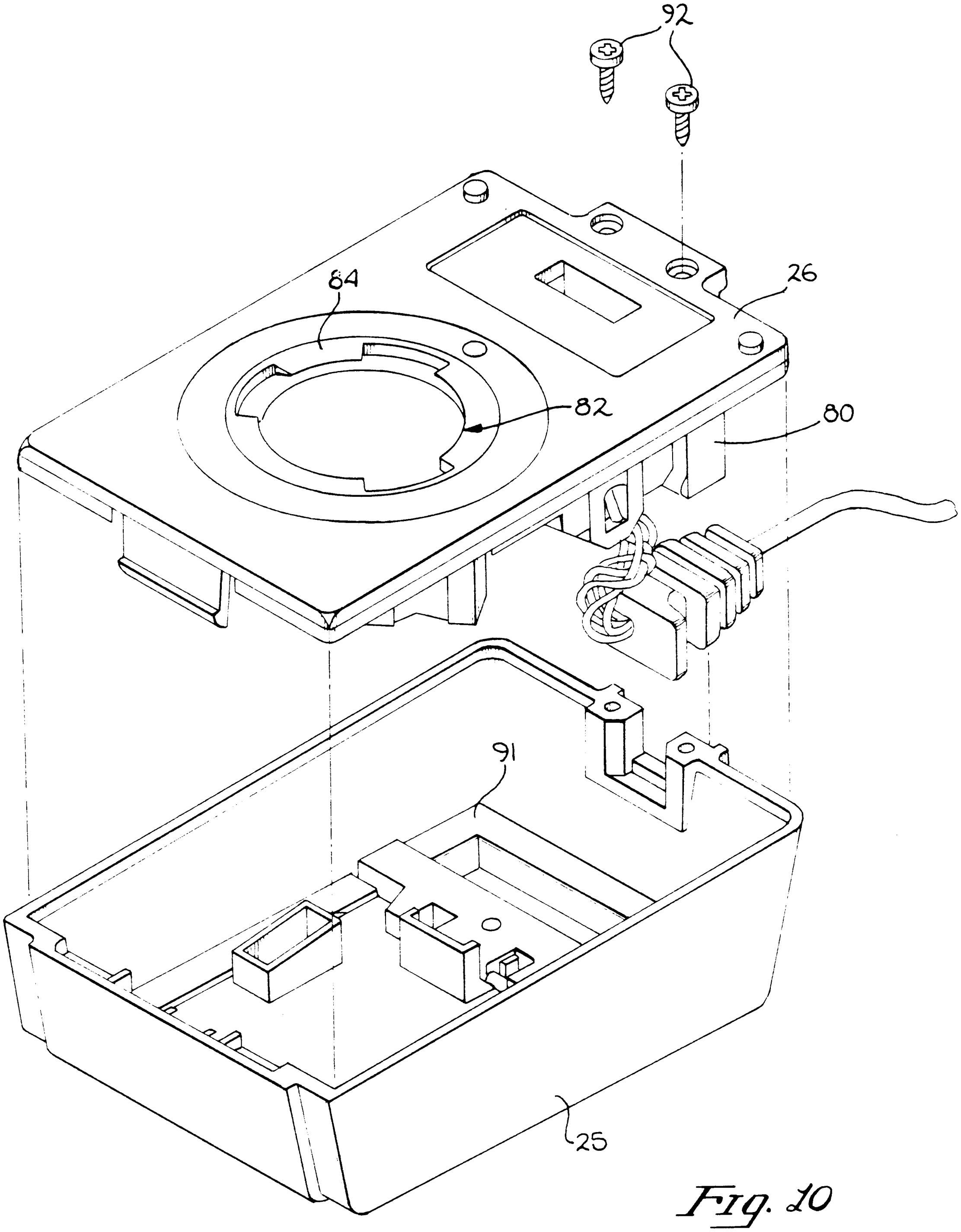


Fig. 10

“APPLE_PAT_4_464_652_B_06” 256 KB 2000-02-23 dpi: 600h x 600v pix: 3684h x 4907v

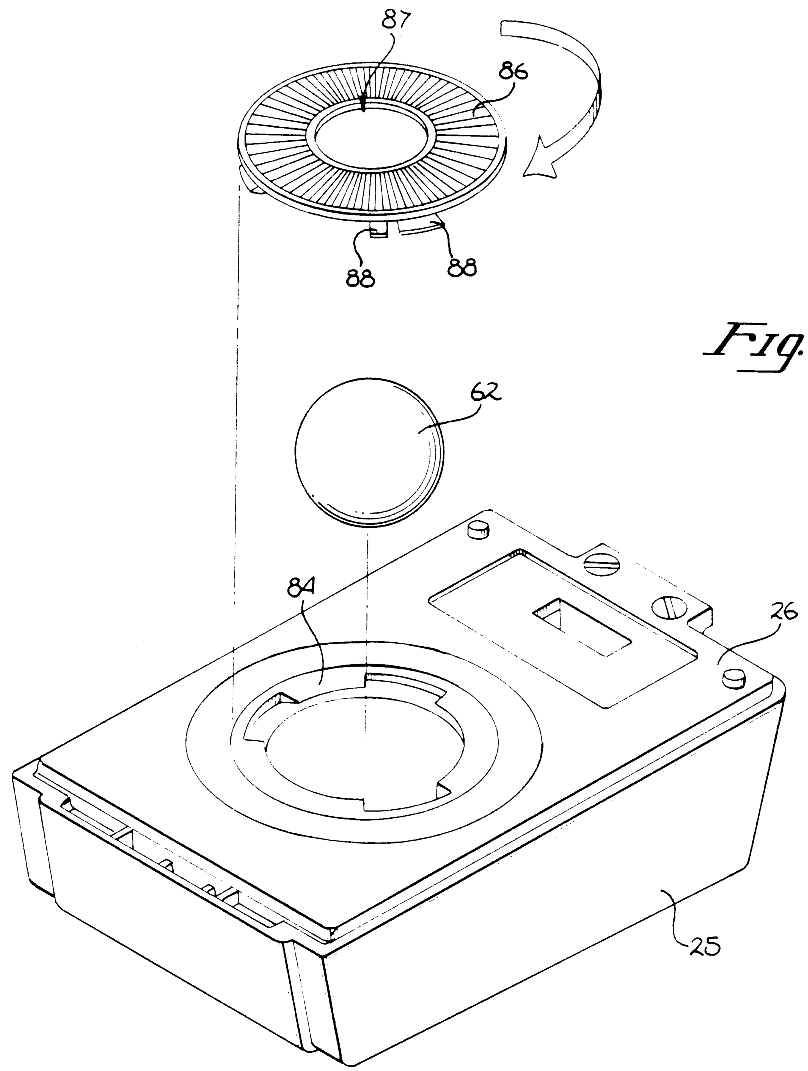


Fig. 11

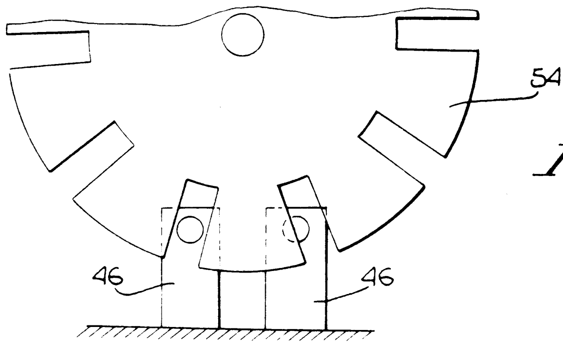


Fig. 12

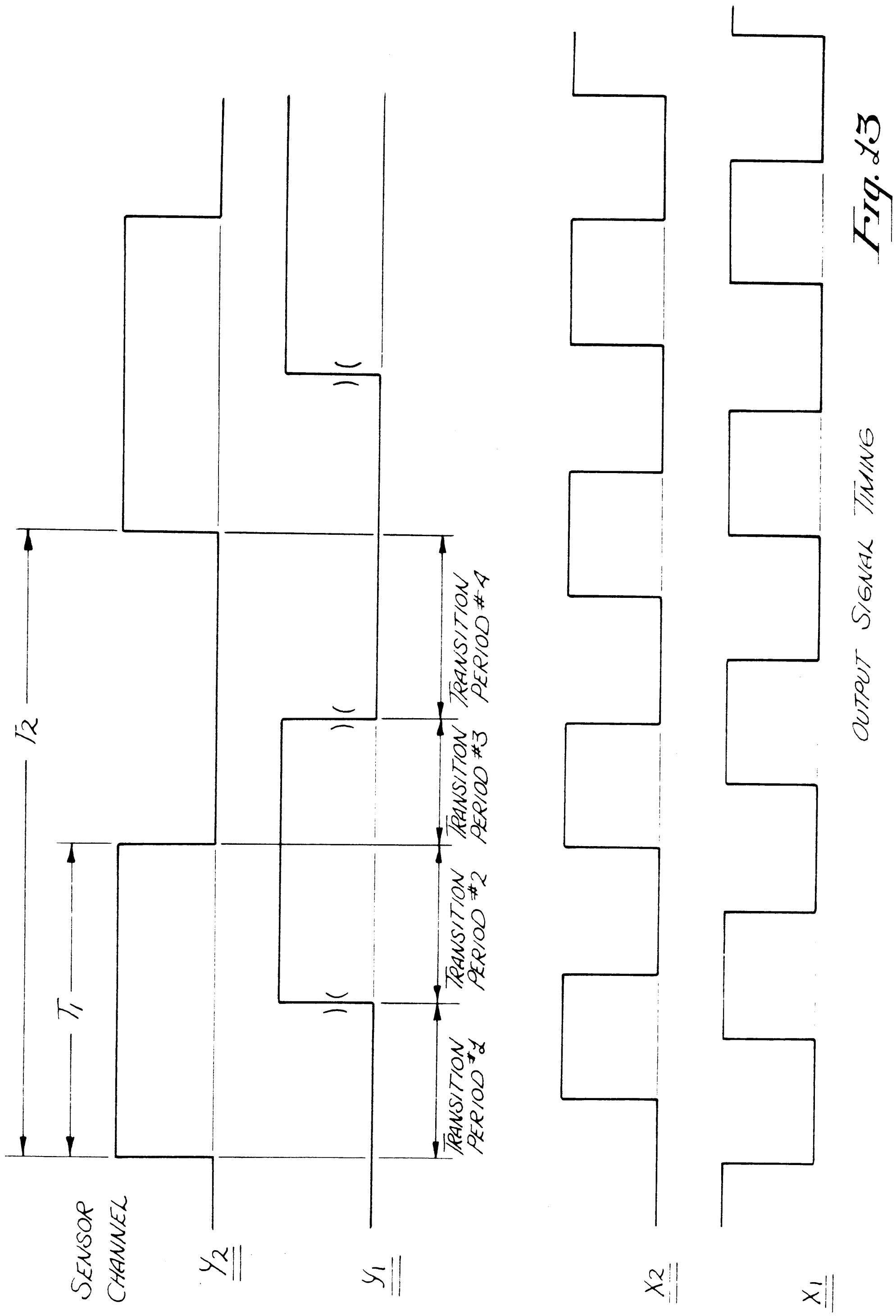

SENSOR CHANNEL
Y_2
Y_1
X_2
X_1
T_1
T_2
TRANSITION PERIOD #1
TRANSITION PERIOD #2
TRANSITION PERIOD #3
TRANSITION PERIOD #4
OUTPUT SIGNAL TIMING

Fig. 13

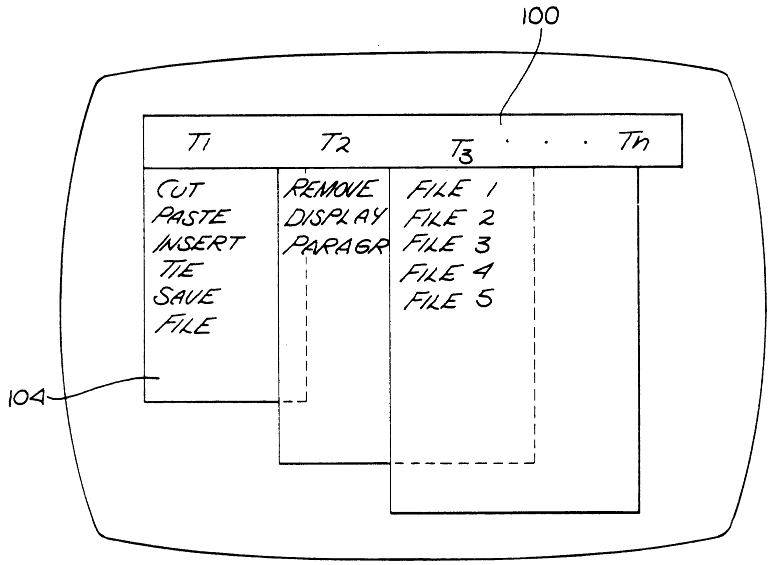

100
T1
T2
T3
Tn
CUT
PASTE
INSERT
TIE
SAVE
FILE
REMOVE
DISPLAY
PARAGR
FILE 1
FILE 2
FILE 3
FILE 4
FILE 5
104

Fig. 14

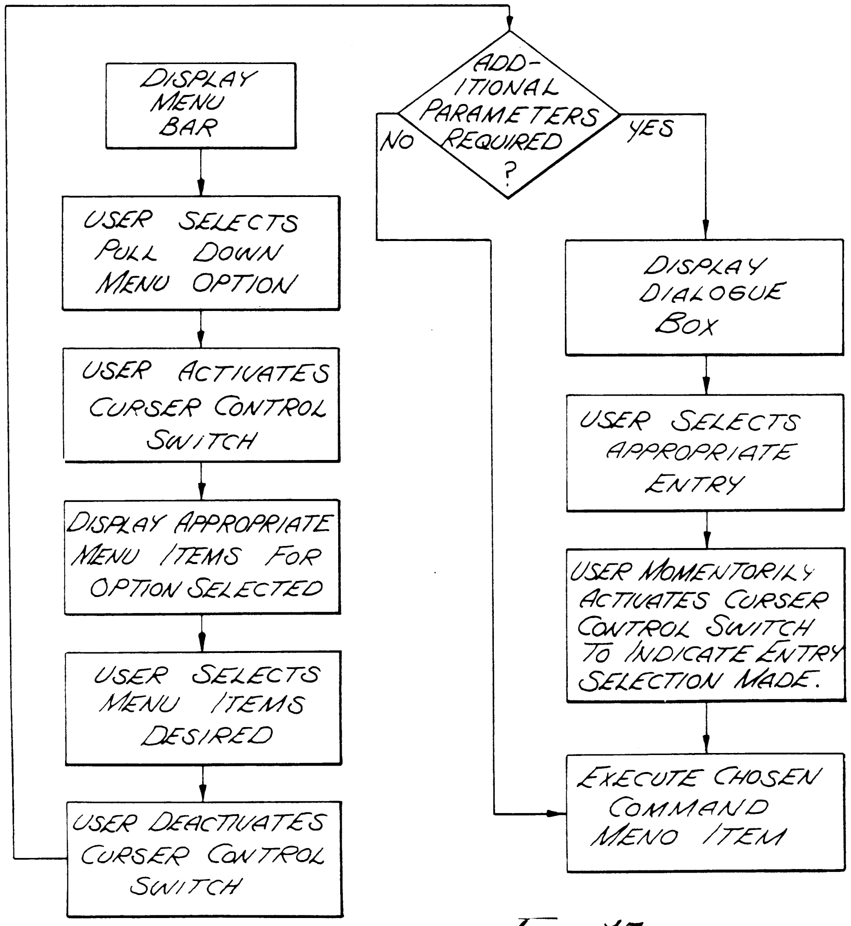

DISPLAY MENU BAR
USER SELECTS PULL DOWN MENU OPTION
USER ACTIVATES CURSER CONTROL SWITCH
DISPLAY APPROPRIATE MENU ITEMS FOR OPTION SELECTED
USER SELECTS MENU ITEMS DESIRED
USER DEACTIVATES CURSER CONTROL SWITCH
ADD-ITIONAL PARAMETERS REQUIRED ?
NO
YES
DISPLAY DIALOGUE BOX
USER SELECTS APPROPRIATE ENTRY
USER MOMENTORILY ACTIVATES CURSER CONTROL SWITCH TO INDICATE ENTRY SELECTION MADE.
EXECUTE CHOSEN COMMAND MENO ITEM

Fig. 15

CURSOR CONTROL DEVICE

Matter enclosed in heavy brackets [ ] appears in the original patent but forms no part of this reissue specification; matter printed in italics indicates the additions made by reissue.

This application is a divisional reissue of application Ser. No. 06/399704, filed Jul. 19, 1982, now U.S. Pat. No. 4,464,652.

BACKGROUND OF THE INVENTION

1. Field

The present invention relates to the field of display systems, and more particularly to devices which can position a cursor over selected locations on a computer controlled display.

2. Art Background

In many computer controlled display systems, it is desirable to allow the user to control the position of a cursor or the like by means which are external from the main computer keyboard. For example, a user may be required to repetitively choose software options displayed on a cathode ray tube (CRT), or may desire to input data in a diagram format into the computer system. In such situations traditional keyboard input systems are not as effective as a cursor control device commonly referred to as a "mouse".

In a typical "mouse" system, a hand-held transducer provides positional movement signals to the display system. Traditionally, the movement of wheels within the cursor control device are coupled to potentiometers to provide signals indicative of an X-Y position on the display screen (see U.S. Pat. Nos. 3,541,541; 3,269,190; and 3,835,464). Other mouse systems utilize rotating balls on wheels which are in turn coupled to rotate apertures interrupting beams of light, thereby providing positional signals to the display system (see U.S. Pat. Nos. 3,892,963 and 3,541,521).

One common disadvantage of cursor control devices found in the prior art is their cost. Typically, prior art cursor controls include costly mechanical parts which require precise alignment for proper operation. Moreover, it is not uncommon for these devices to exhibit a loss in accuracy over time as the mechanism wears. As computer display capabilities have become more advanced in terms of user real-time graphic interaction, cursor control devices have become a necessity in many computer systems. Accordingly, there exists a need to provide a cost effective, simple and highly reliable cursor control device for providing signals indicative of X-Y positions on a computer display system.

As will be disclosed below, the present invention provides an improved cursor control device which overcomes the disadvantages of the prior art by utilizing a unitary frame structure for accurate alignment of all elements and simple assembly, as well as photo-optics to provide the required positional signals. In addition, a display system and method is disclosed for use in association with the cursor control device which permits a user to select command options simply by movement of the cursor over a "pull-down" menu bar.

SUMMARY OF THE INVENTION

A cursor control device having particular application to computer display systems is disclosed. The cursor control includes a unitary frame having a domed portion which houses a ball which is free to rotate. Two encoder disc assemblies are provided, which include roller shafts disposed substantially 90 degrees relative to one another and in contact with the ball. Each roller shaft is coupled to an encoder disc having a plurality of slots disposed radially around the disc periphery. These slots interrupt light beams which are provided by photo-emitters and directed at photo-detectors. Each slotted disc interrupts two light beams which are arranged such that when one beam is fully transmitted, the other is partially blocked. Beam interruptions produce signal pulses representing increments of motion, while the order in which the light beams are interrupted indicates the direction of motion, thereby resulting in an X-Y position on a display system. The ball is maintained in contact with the roller shafts by a spring biased idler wheel. The ball is free to "float" in the vertical direction within the dome, and thereby maintain good surface contact. Moreover, the ball may be easily removed for cleaning to insure that any build up of lint or the like does not prevent the ball from rotating smoothly. A switch is provided within the cursor control housing in order to signal the display system that a desired X-Y location on the display screen has been selected. In operation, a user may selectively position a cursor or the like on a display system by simply moving the cursor control device over a surface, such as a desk, until the desired cursor position is shown on the display device. A display system and method is disclosed for use in conjunction with the cursor control device, which permits user to select command options simply by movement of the displayed cursor over a "menu bar".

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a perspective view of the present invention.

FIG. **2** is a perspective of the present invention illustrating the cursor control device as it appears without the housing cover.

FIG. **3** is a perspective view of the unitary frame of the present invention coupled to the printed circuit board base, illustrating the placement of photo-detectors and the coupling connector.

FIG. **4** is a further perspective view of the unitary frame and circuit boad of FIG. **3** illustrating the position of a roller shaft and encoder wheel.

FIG. **5** is a top view of the unitary frame and printed circuit board of the present invention.

FIG. **6** is a partial view of the unitary frame in FIG. **3**, illustrating the insertion of a detector aperture.

FIG. **7** is a perspective view of the unitary frame of FIG. **3**, illustrating the placement of resistors on the printed circuit board.

FIG. **8** is a perspective view of the coupling of the unitary frame cage and printed circuit board combination to the housing base of the present invention.

FIG. **9** is a perspective view illustrating the placement of the control switch within the housing base.

FIG. **10** is the perspective view of the final assembly of the present invention illustrating the coupling of the cover and base portions of the housing.

FIG. **11** is a perspective view illustrating the insertion or removal of the floating and rotating ball.

FIG. **12** is a diagrammatical illustration of the alignment of the photo-emitters in relation to each encoder disc.

FIG. **13** is a diagrammatical illustration of a sample quadrature output of the present invention indicative of X-Y locations on display system.

FIG. **14** is a diagrammatical illustration of a "pull down" menu bar display.

FIG. **15** is a block diagram illustrating the sequence of steps utilized by the present invention to display options and associated commands on a "pull-down" menu bar display.

DETAILED DESCRIPTION OF THE INVENTION

A cursor control device having particular application for use in conjunction with a computer display system is disclosed. In the following description for purposes of explanation, specific numbers, materials and configurations are set forth in order to provide a thorough understanding of the present invention. However, it will be apparent to one skilled in the art that the present invention may be practiced without the specific details. In other instances, well known systems are shown in diagrammatical or block diagram form in order not to obscure the present invention unnecessarily.

Referring now to FIG. **1**, the present invention includes a hand held cursor control unit **20** which is coupled to a plug **22** by means of a cable **24**. As best illustrated in FIG. **2**, cursor control unit **20** includes a cover **25** and a base **26** upon which the internal workings of the present invention are disposed. As will be apparent from the discussion which follows, cursor control unit **20** is designed with ease of assembly in mind, while providing very close tolerances and high X-Y position location accuracy.

With reference to FIGS. **3**, **4** and **5**, a premolded unitary frame **28** is provided which includes a domed housing **30** presently having three cut-out locations **31**, **32** and **33**. As illustrated, cut-outs **31** and **32** are disposed substantially at 90 degrees with respect to one another, with cut-out **33** being oriented generally symmetrically opposite the other cut-outs. In addition, frame **28** includes a plurality of bosses, slots and shaped stems of material which when pertinent will be discussed in this specification. In the presently preferred embodiment, the frame **28** is comprised of a plastic material (e.g. polycarbonate) which is impregnated with a lubricant (e.g. teflon). Thus, during operation and throughout its useful life, cursor control unit **20** does not require the addition of either wet or dry lubricants. Frame **28** is mounted on a printed circuit board **34** to facilitate electrical connection between the various electrical elements within the unit. Electrical connector header **36** is mounted as shown (see FIG. **3**) to the unitary frame **28** such that connector pins **38** pass through a rectangular slot **39** through the frame to the circuit board below. As will be discussed, cable **24** is electrically coupled to the cursor control unit **20** through connector **36**.

As illustrated in FIG. **3**, photo-emitters **40** are inserted into slots **42** such that the emitter portion is facing away from the dome **30** (note that one emitter **40** is shown in FIG. **3** partially inserted). Upwardly extending clips **43** are snapped over portions of each emitter **40**, as shown, to prevent them from being dislodged. Similarly, two photo-detectors **46** are inserted facing the emitters **40** into slots **47** in each of two detector apertures **50**. As shown in FIG. **6**, an outwardly extending portion **48** of each detector aperture **50** is aligned with guides **49** formed integrally with the frame **28**, and the aperture is then snapped downward into place.

Thus, each detector aperture **50** houses two detectors **46** which face two emitters **40**, respectively. In the presently preferred embodiment, the emitter/detector combination operates within the infrared region. However, it will be appreciated that any suitable wavelength may be used in a particular application. In addition, presently, the detectors **46** incorporate integral Schmitt triggers to provide detector outputs which more closely approximate a digital signal.

Two encoder disc assemblies are provided to convert, as will be described, the movement of the cursor control unit **20** into signals indicative of X-Y locations defined on the display system. Each encoder assembly **52** includes an encoder disc **54** axially coupled to a roller shaft **56**. In addition, each encoder disc **54** is provided with a plurality of radially disposed slots **57** which interrupt the light beams generated by the photoemitters **40**. A cylindrical contact member **58** surrounds each roller shaft **56** at each respective cut out location, as illustrated. Each encoder disc assembly **52** is mounted on the unitary frame **28** by inserting the encoder disc **54** between the detector aperture **50** and emitters **40** and snapping an end clip **60** over the opposite end of the roller shaft **56** (see FIGS. **4**, **5** and **7**), thereby allowing rotation of the roller shaft and encoder disc with a minimum of friction. As illustrated, each shaft **56** is slipped into and carried by a "U" shaped guide **59** formed from upwardly extending alignment bosses **53** to maintain each roller shaft **56** in proper orientation. End **51** of the shaft **56** is carried for rotation within a hollow portion of the detector aperture **50**, such that encoder disk **54** is disposed in close proximity to the aperture **50**. The present invention's use of integral lubrication within the frame material, permits each shaft **56** to freely rotate about its longitudinal axis.

As a result of the above described configuration, the radially disposed slots **57** of each encoder disc interrupt two light beams from photo-emitters **40**. The position of the emitter/detector combination and encoder disc is such that when one beam is fully transmitted, the other is partially blocked by a slit on the encoder disc. As will be discussed, in operation a ball **62** is disposed within the dome **30** of the frame, and retained such that it is maintained in contact with both cylindrical contact members **58**. The rotation of the ball **62** within the dome **30** in turn causes the rotation of each roller shaft **56** and its respective encoder disc. As will be discussed, the beam interruptions from the rotation of each encoder disc **54** produce signal pulses representing increments of motion, while the order in which the light beams are interrupted indicates the direction of motion of the cursor control unit.

Ball **62** is retained against the cylindrical contact members **58** by an idler wheel for rotation on a fixed shaft **66**, as best shown in FIG. **5**. The idler wheel **64** and shaft **66** are inserted within a slot **68** formed by rectangular bosses **69** and **70** extending upwardly from the frame's base. Wheel **64** extends through cut-out **33** into the interior of the dome **30**. The legs of a staple shaped idler spring **72** are inserted through passages **73** passing perpendicular to the horizontal plane of the frame **28** and circuit board **34**, thereby retaining the shaft **66** within the slot **68**.

Referring now to FIG. **7**, resistors **76**, which are required by the specific electronics of the emitter/detector combination of the present invention, are inserted into the printed circuit board **34**. The resistors **76** and associated leads from the connector **36**, photo-emitters

40, and photo-detectors 46 are then electrically connected and soldered in place as is conventionally done in the art.

With reference now to FIGS. 8, 9 and 10, the assembled frame 28 and circuit assembly is mounted on the base 26 by means of a screw 78. As illustrated, base 26 includes an upwardly extending switch retaining portion 80 and a generally circular cut-out orifice 82. As best shown in FIGS. 8, 10 and 11, circular orifice 82 is disposed substantially below the opening of dome 30, and includes outwardly extending locking ridges 84 which are designed to accommodate a lock cap 86 (See FIG. 11), such that ball 62 may be retained within the dome 30. Lock cap 86 includes outwardly extending tabs 88 arranged to interleaf with ridges 84. In operation, a user desiring to insert or remove ball 62 from the cursor control unit 20, may unlock and remove the lock cap 86 from the orifice 82 by simply rotating the cap such that the tabs 88 and ridges 84 no longer interleaf.

As illustrated, lock cap 86 generally has a toroidal form having a central orifice 87 of smaller diameter than cutout orifice 82. It will be apparent, that once ball 62 is inserted and retained by lock cap 86, Thus, ball 62 contacts the surface below the cursor control unit 20 and rotates in response to the movement of the unit on the surface.

As shown in FIG. 9, cable 24 is coupled to cursor control 20 through a female connector 94 which is inserted over pins 38. A switch 90 is coupled to the cable 24 through electrical connector 36, and is inserted within the retaining portion 80. A switch cap 91 forms part of the cover 25 (see FIG. 1), and is disposed above switch 90 such that the depression of the switch cap 91 forces switch 90 to electrically close, and thereby signal the computer display system that an appropriate X-Y location has been selected. As shown in FIG. 10, base 26 and cover 25 are coupled by securing both sections to one another using screws 92. Once the cover and base have been joined, ball 62 is inserted and lock cap 86 is attached as discussed above to retain the ball within the dome portion 30.

With reference to FIGS. 12 and 13, a sample quadrature output of the cursor control unit 20 is illustrated. As previously described, photo-detectors 46 are disposed such that if one detector is fully exposed by a slot of the encoder disc 54, the other detector is only partially exposed. Thus, in addition to the increments of motion of the cursor control over a surface, the direction of motion may also be determined. Assume for sake of example that the cursor control 20 is moved. As illustrated in FIG. 13, a substantially digital output signal is generated by each photo-emitter/detector combination associated with each encoder assembly. In the example shown, cursor control 20 would provide a regularly spaced output from the X channel detectors if the control 20 is moved over a surface at a constant speed along the X-axis. Similarly, if there is little movement of the control unit along the Y axis, little change will occur on the Y channels inasmuch as the Y encoder disk is not being rotated significantly (see FIG. 13). The computer display system is provided with appropriate software or hardware, for example edge detectors, to detect signal state transitions. Thus, the signals from each pair of channels may be decoded such that the X-Y direction of motion may be determined for the particular order of transition changes from each channel along an axis. Inasmuch as the particular circiutry and software used for decoding the various signals and positioning the cursor or the like on a display system will be apparent to one skilled in the art, the details of such will not be recited herein.

Referring now to FIGS. 14 and 15, a display system and method for use in conjunction with the cursor control device 20 will be described. As previously discussed, control 20 is coupled to a display system which is controlled by a computer or other equivalent circuitry. Appropriate programming of the computer is provided such that a "menu" bar 100 comprising a variety of command options indicated by titles (for example, T_1, T_2, T_3. . . T_n), is displayed across the CRT screen or the like as shown in FIG. 14. If a particular title (for example T_1) is selected, one or more sub-command items 104 are displayed by the computer system below the primary menu title. As illustrated, the sub-command items appear to the user to be "pulled down" from the main menu bar 100. The user then selects a desired item for execution by the computer by appropriate movement of a cursor control, as will be described. Although the list of items 104 are shown for illustration below menu title options T_1, T_2, and T_3, in the present embodiment only one menu option may be pulled down and displayed at a time.

The sequence of operations executed by the computer system to permit the user to select a particular menu title and sub-command item is shown in FIG. 15. The computer initially displays menu bar 100 on the display system as shown in FIG. 14. A user desiring to select a particular title moves cursor control unit 20 over a surface, thereby rotating ball 62 within dome 30 and sending signals indicative of X-Y locations to the display system for corresponding movement of a cursor or the like on the display screen. Once the cursor is positioned over (or in proximity with) the chosen menu title selection, the user depresses switch cap 91 on cursor control 20, thereby activating switch 90, and signaling the computer system that the particular title has been selected. The computer display system then either executes the menu title if it is an immediate command, or displays a set of sub-command items for user selection. If items are displayed, the user continues to depress switch cap 91, and once again moves the cursor control over the surface until the displayed cursor lies over or in proximity with the item to be executed. The user then removes pressure from the switch cap 91 thereby deactivating switch 90, and indicating to the computer which item is to be executed.

The computer system then determines if further parameters are required to be specified by the user. If no further data is required, the computer executes the item indicated by the cursor position on the display screen. However, if parameters must be specified by the user prior to execution a "dialogue box" is defined on the display system which displays the various data selections which are required. For example, a user may be required to select page formats, specify numerical values, etc. In the present embodiment, a user inputs the desired data selections by positioning the cursor over the selection, in for example a multiple choice format, and momentarily activates the switch 90 on the cursor control unit. Once the required selections are made, the computer proceeds to execute the chosen menu item.

Accordingly, it is possible for a user to select and execute a variety of commands without the necessity of inputting characters on a keyboard, as is commonly required in the art. Rather, the present invention permits fast entry and execution of commands, such as for example in a word processing system or the like, wherein large blocks of text or other data may be manipulated or operated upon simply by movement of the cursor control **20** over a surface and the appropriate depression of switch **90**.

Thus, an improved cursor control and display system has been described. The present invention permits a user to select desired menu titles on a menu bar by movement of a cursor control over a surface. Sub-command items may be specified for execution by the computer control display system in the same manner, such that the operator need not enter command characters on a keyboard or the like in order to access and execute most system functions.

Although the present invention has been described with reference to FIGS. **1-15** and with emphasis on a "pull down" type display system, it should be understood that the figures are for illustration only and should not be taken as limitations upon the invention. It is contemplated that many changes and modifications may be made, by one of ordinary skill in the art, to the materials and arrangements of the elements of the invention without department from the spirit and scope of the invention as disclosed above.

What is claimed is:

1. A device for providing signals indicative of X-Y locations on a display system or the like, comprising:

a housing including a base having an opening for the passage of a rotatable ball;

a unitary frame disposed on saide base including:

a domed portion integrally formed with said frame substantially surrounding and retaining said rotatable ball;

said domed portion having first and second cut-outs through said dome disposed substantially at 90 degrees with respect to one another, and a third cut-out disposed at an angle with respect to said first and second cut-outs;

X-Y position indicating means passing through said first and second cut-outs, for converting the rotation of said ball into signals indicative of X-Y positions on said display system;

biasing means passing through said third cut-out, for biasing said ball against said X-Y position indicating means;

means for removing said ball from said domed portion through said opening in said base, such that said ball and the interior of said dome may be serviced, said means for removing comprising:

outwardly extending lock ridges integrally formed with said opening in said base;

a lock cap having a second opening of smaller diameter then said base opening to permit only a portion of said ball to pass therethrough and contact said surface;

said lock cap further including outwardly extending lock tabs to interleaf with said lock ridges, such that rotation of said cap interleafs with said tabs and ridges thereby locking said cap onto said base;

whereby movement of said device over a surface such that a portion of said ball is maintained in contact with said surface results in X-Y positions defined on said display system.

2. The device as defined by claim **1**, wherein said biasing means comprises a wheel carried by a shaft, and shaft being biased such that said wheel is maintained in contact with said ball.

3. The device as defined in claim **2**, wherein said third cut-out is disposed generally at 45 degrees with respect to said first and second cut-outs.

4. The device as defined by claim **3**, wherein said X-Y position indicating means includes a roller shaft coupled to an encoder disc having a plurality of radially disposed slots, said disc being disposed between a photo-emitter and photo-detector.

5. The device as defined in claim **4**, wherein said photo-detector is diposed within a detector aperture, said aperture being retained on said unitary frame to form an integral unit.

6. The device as defined by claim **5**, further including a circuit board disposed between said frame and said base.

7. The device as defined by claim **6**, further including a switch coupled to said circuit board to specify selected X-Y positions on said display system.

8. The device as defined by claim **7**, said device being coupled to a computer controlled display system wherein menu commands are displayed and selected by a user through movement of said device.

[**9**. A computer controlled display system having a display wherein a plurality of command options are displayed along a menu bar and sub-command items corresponding to each option are displayed once said option has been selected, comprising:

first display means coupled to said computer for generating and displaying said menu bar comprising said plurality of command options;

cursor control means coupled to said display system for selectively positioning a cursor on said display, said cursor control means including a cursor control device for movement over a surface, the movement of said cursor control device over said surface by a user resulting in a corresponding movement of said cursor on said display;

signal generation means including a switch having a first and second position coupled to said display system for signalling said computer of an option choice once said cursor is positioned over a first predetermined area on said display corresponding to an option to be selected, said user placing said switch in said second position while moving said cursor control device over said surface such that said cursor is over said first predetermined area;

second display means coupled to said computer for generating and displaying said sub-command items corresponding to said selected option;

said switch being placed in said first position by said user once said user has positioned said cursor over a second predetermined area corresponding to a sub-command item to be selected;

whereby an option and a sub-command item is selected and executed by said computer.]

[**10**. The display system of claim **9** wherein said cursor control device comprises:

a housing including a base having an opening for the passage of a rotatable ball;

a unitary frame disposed on said base including:

a domed portion integrally formed with said frame substantially surrounding and retaining said rotatable ball;

said domed portion having first and second cut-outs through said dome disposed substantially at 90 degrees with respect to one another, and a third cut-out disposed at an angle with respect to said first and second cut-outs;

X-Y position indicating means passing through said first and second cut-outs, for converting the rotation of said ball into signals indicative of X-Y positions on said display system;

biasing means passing through said third cut-out, for biasing said ball against said X-Y position indicating means;

means for removing said ball from said domed portion through said opening in said base, such that said ball and the interior of said dome may be serviced, said means for removing said ball comprising:

outwardly extending lock ridges integrally formed with said opening in said base;

a lock cap having a second opening of smaller diameter then said base opening to permit only a portion of said ball to pass therethrough and contact said surface;

said lock cap further including outwardly extending lock tabs to interleaf with said lock ridges, such that rotation of said cap interleafs with said tabs and ridges thereby locking said cap onto said base;

whereby said option and sub-command item may be selected by movement of said cursor control means over a surface such that a portion of said ball is in contact with said surface.]

[**11**. In a computer controlled display system having a display wherein a plurality of command options are displayed along a menu bar and sub-command items corresponding to each option are displayed once said option has been selected, a method for selecting an option and an item, comprising the steps of:

(a) generating and displaying said menu bar comprising said plurality of command options;

(b) positioning a cursor on said display using a cursor control device for movement over a surface, the movement of said cursor control device over said surface by a user resulting in a corresponding movement of said cursor on said display;

(c) signalling said computer of an option choice once said cursor is positioned over a first predetermined area on said display corresponding to an option to be selected, said user signalling said computer by placing a switch coupled to said display system in a second position while moving said cursor control device over said surface such that said cursor is over said first predetermined area;

(d) generating and displaying said sub-command items corresponding to said selected option;

(e) positioning said cursor over a second predetermined area corresponding to a sub-command item to be selected, said switch being maintained in said second position until said cursor is positioned over said second predetermined area;

(f) placing said switch in a first position once said user has positioned said cursor over said second predetermined area;

whereby an option and an item associated with said option is selected.]

[**12**. The method as defined by claim **11**, wherein said switch is disposed on said cursor control device.]

[**13**. The method as defined by claim **12**, where said computer displays said sub-command items generally below said option on said menu bar.]

* * * * *

United States Patent [19]

Jordan et al.

[11] **Patent Number: 4,466,033**

[45] **Date of Patent: Aug. 14, 1984**

[54] **DISK DRIVE WITH AUTOMATIC DISC CLAMPING AND EJECTING**

[75] Inventors: **Richard Jordan,** Los Altos; **William Bull,** Sunnyvale; **Robert L. Ciardella,** Saratoga; **Robert Taggart,** Portola Valley; **Frederick R. Holt,** Cupertino, all of Calif.

[73] Assignee: **Apple Computer, Inc.,** Cupertino, Calif.

[21] Appl. No.: **351,652**

[22] Filed: **Feb. 24, 1982**

[51] **Int. Cl.³** **G11B 17/02; G11B 5/016**
[52] **U.S. Cl.** **360/99**
[58] **Field of Search** 360/97, 98, 99, 106

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,193,102	3/1980	Beuch	360/99
4,359,762	11/1982	Stollorz	360/98

FOREIGN PATENT DOCUMENTS

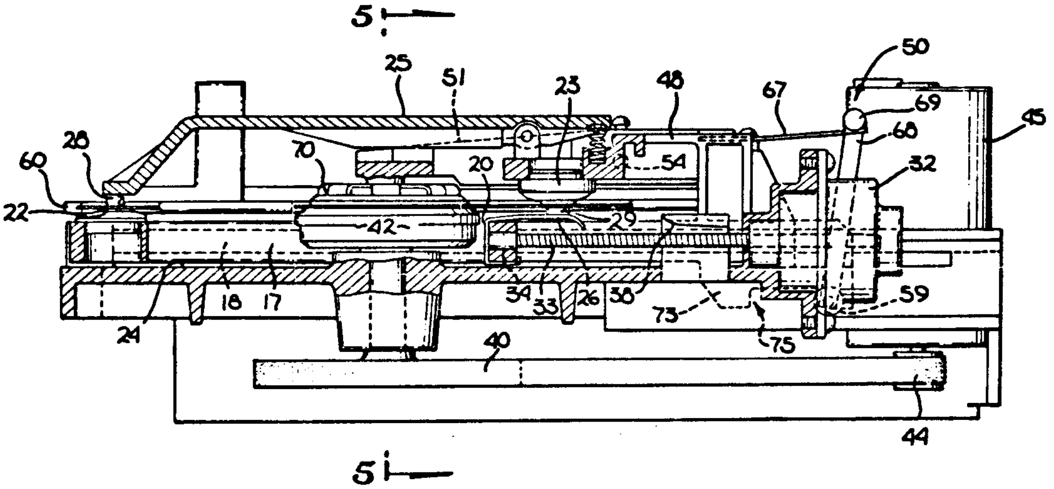
0010172	4/1980	European Pat. Off. .
0042061	12/1981	European Pat. Off. .
1466809	3/1977	United Kingdom .
2016794	8/1982	United Kingdom .

Primary Examiner—Bernard Konick
Assistant Examiner—Paul Stefanski
Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman

[57] **ABSTRACT**

A floppy disk drive with automatic disc clamping and ejection is disclosed. The movement of the linear actuator is used to provide both the clamping and ejection, without other drive means. A pair of magnetic heads, positioned on opposing sides of a drive wheel, are fixed to a carriage, one engages the upper surface of the disc, the other the lower surface. Neither head moves relative to the other. The drive components are easily assembled providing a relatively inexpensive, yet reliable drive.

9 Claims, 12 Drawing Figures



Lisa "Twiggy" drive

(5.25" disks, 860K)

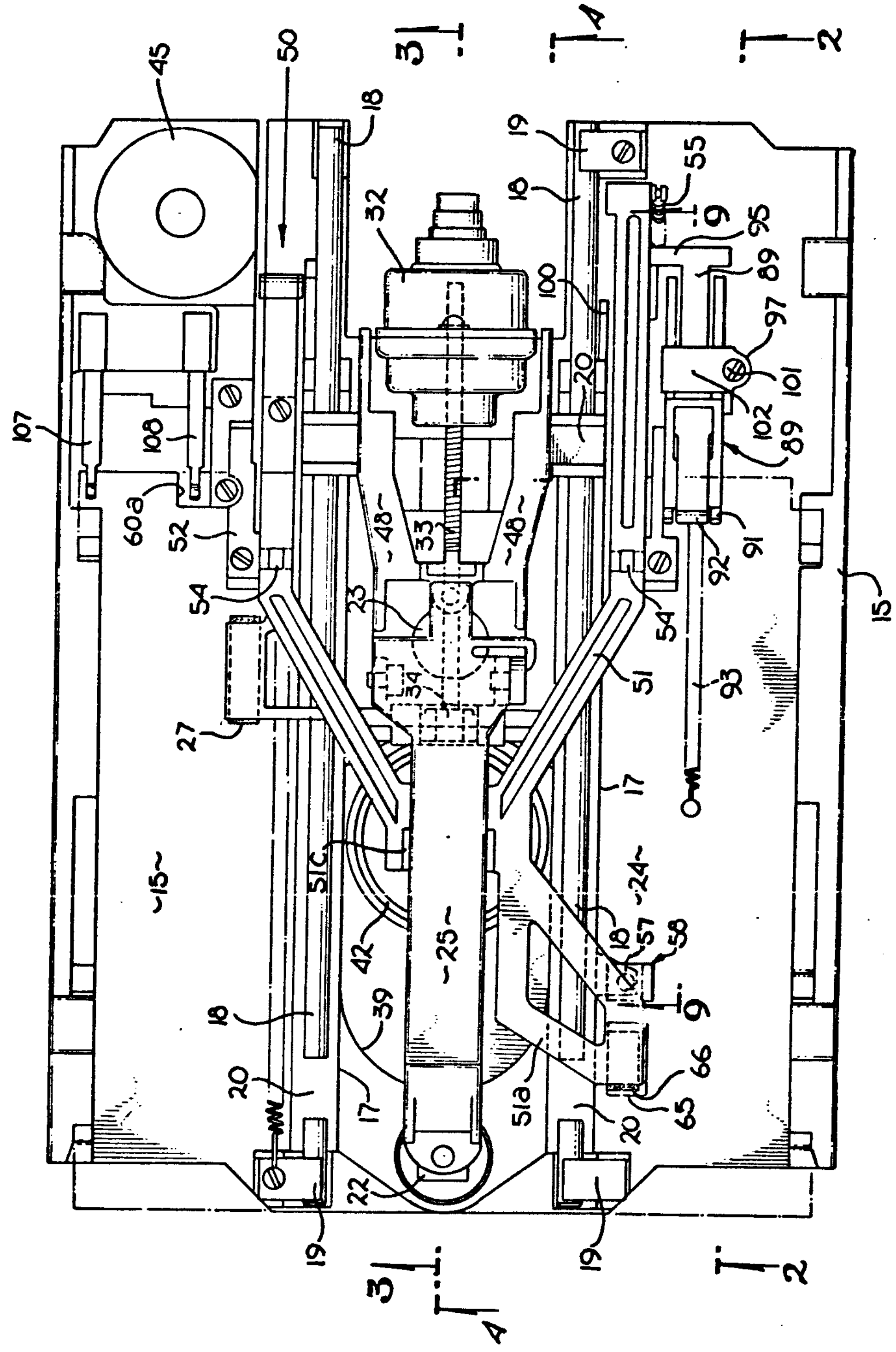

45
50
3
4
2
18
19
55
9
95
89
32
100
97
20
101
102
107
108
60a
52
48
33
92
91
54
23
15
34
51
93
27
17
51c
24
15
42
25
18
57
58
39
66
65
51a
22
20
19

Fig. 1

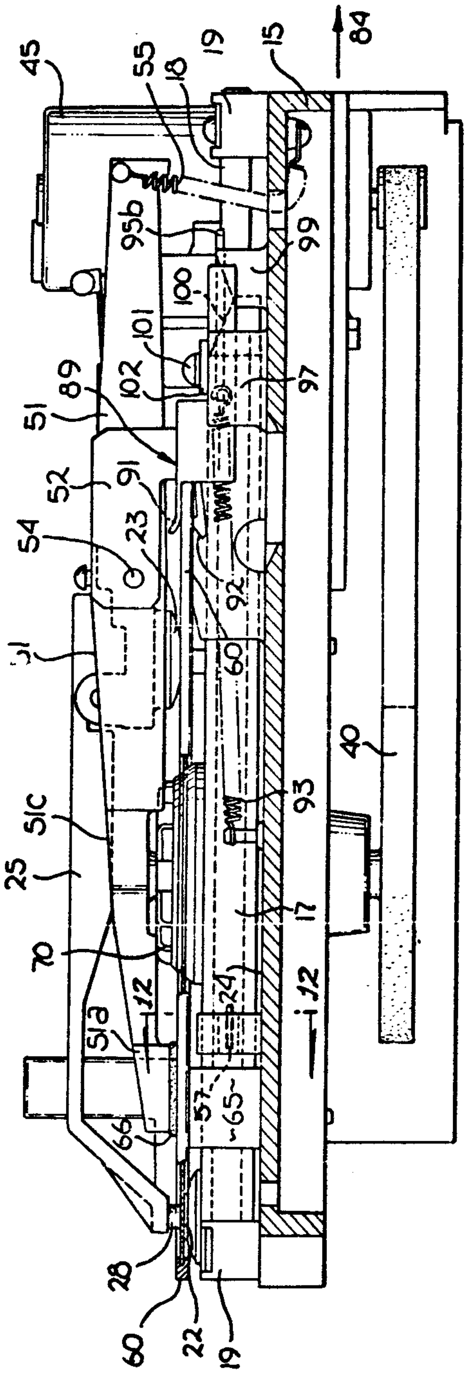
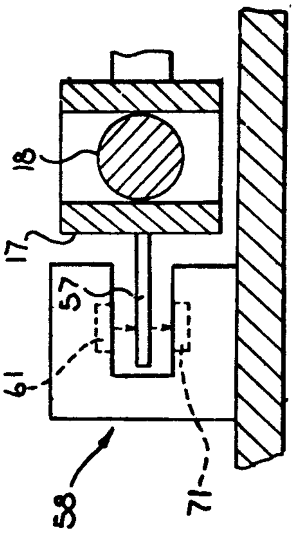

45
55
18
19
15
84
99
95b
100
101
102
97
89
51
52
91
54
23
92
60
31
40
93
51c
25
17
70
12
12
24
51a
57
65
66
28
60
22
19

Fig. 2

Fig. 12


18
17
57
61
58
71

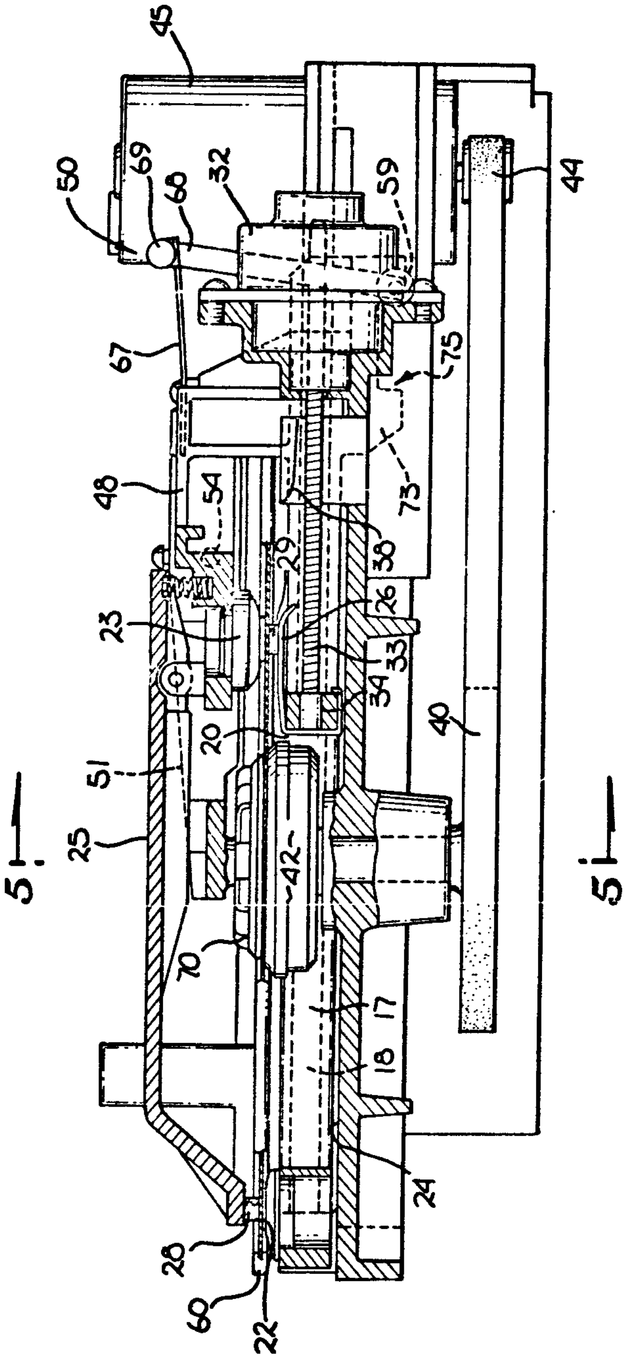

45
50
69
68
32
59
44
67
75
48
54
73
38
29
23
26
33
34
20
40
51
25
5
5
42
70
17
18
24
28
60
22

Fig. 3

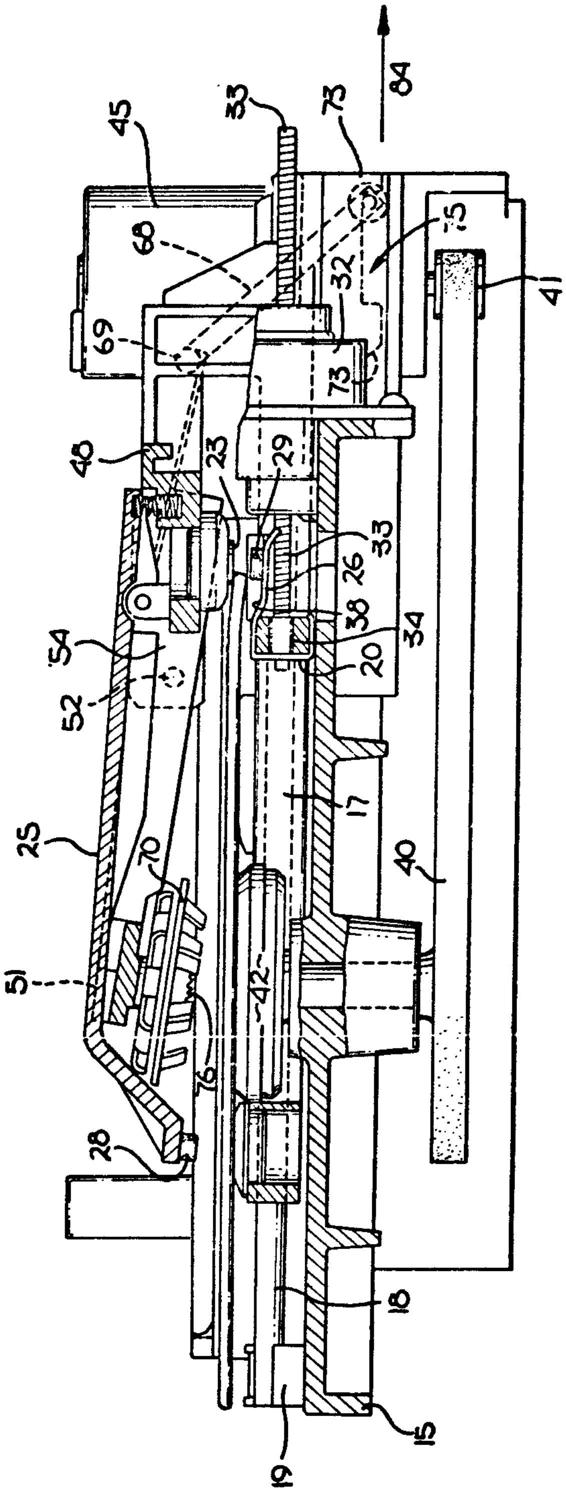

45
33
73
84
68
32
75
41
69
73
48
23
29
33
26
38
34
20
54
52
25
70
17
40
51
42
76
28
18
15
19

Fig. 4

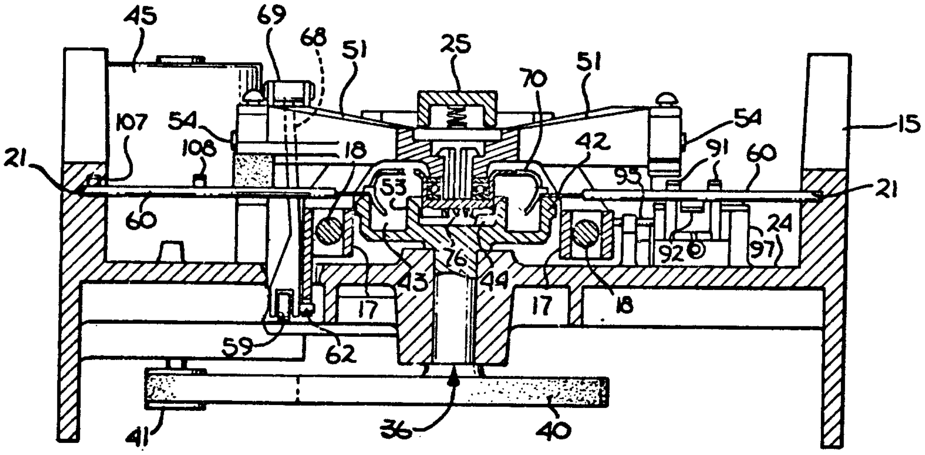


Fig. 5

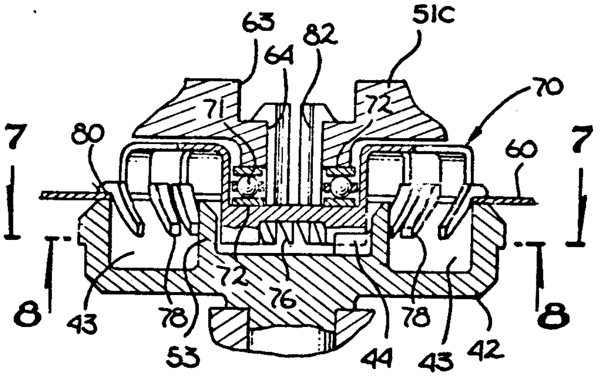
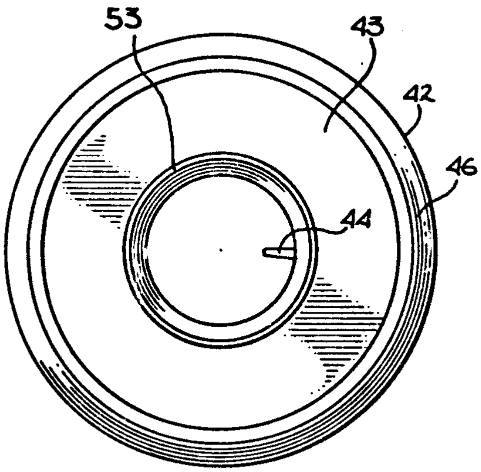


Fig. 6



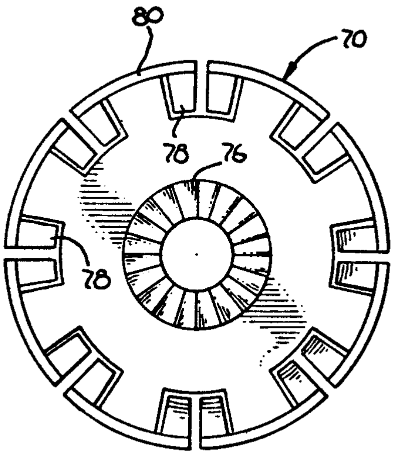


Fig. 7

Fig. 8

U.S. Patent Aug. 14, 1984 Sheet 6 of 6 4,466,033

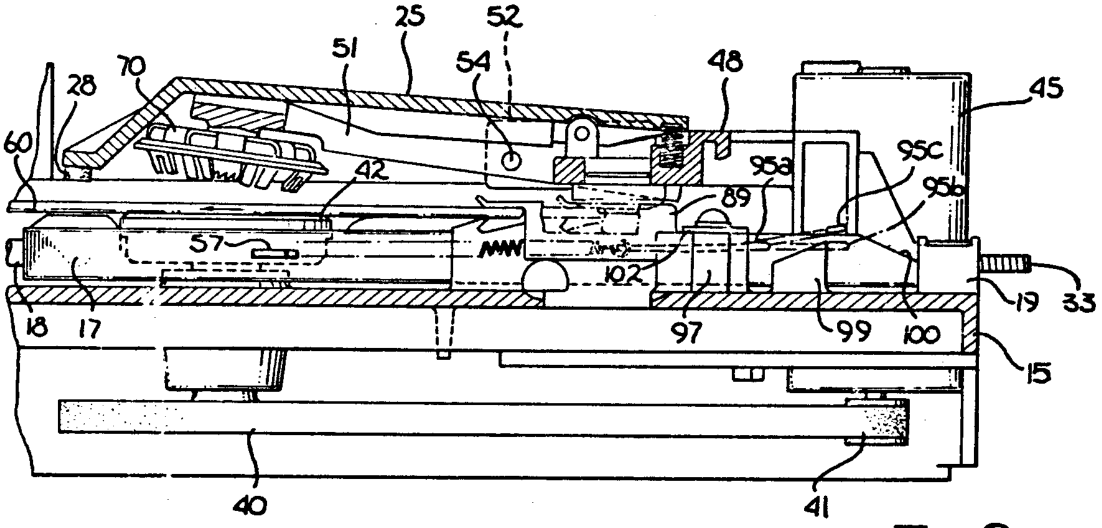


Fig. 9

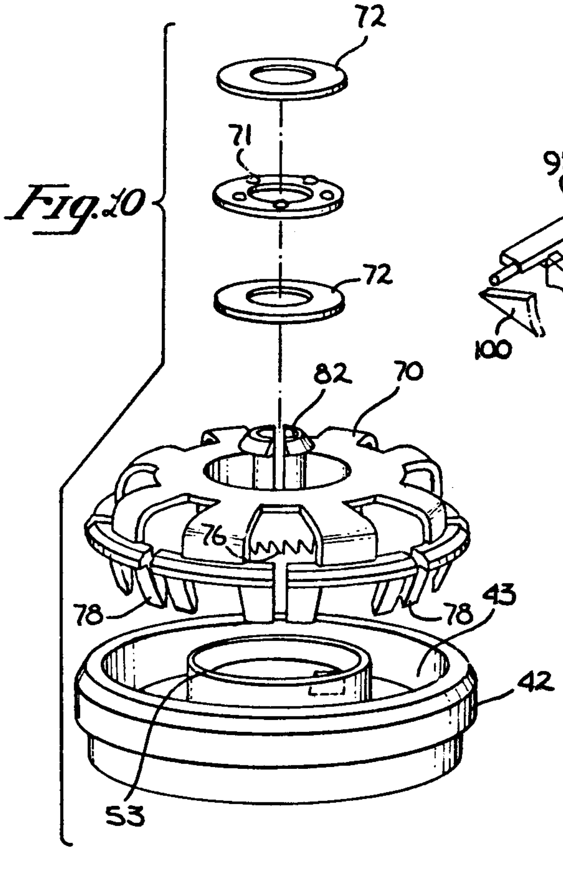


Fig. 10

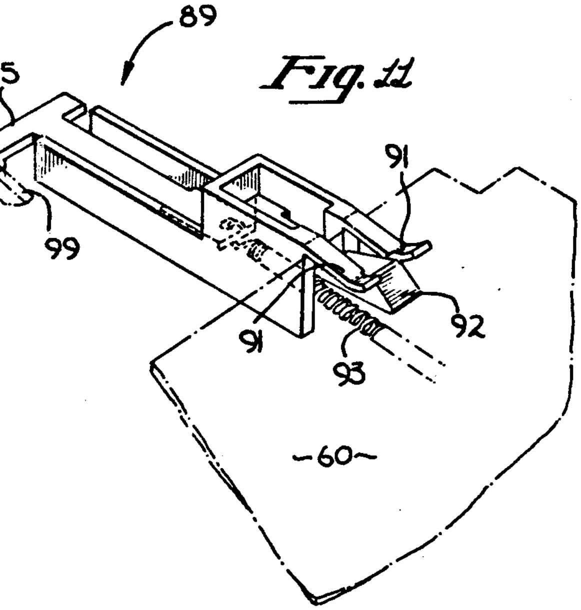


Fig. 11

“APPLE_PAT_4_466_033_07” 147 KB 2000-02-21 dpi: 300h x 300v pix: 1938h x 2817v

DISK DRIVE WITH AUTOMATIC DISC CLAMPING AND EJECTING

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention relates to the field of drives for magnetic discs, particularly "floppy" disc.

2. Prior Art

There are numerous commercially available floppy disc drives which have been marketed in large quantities for several years. For the most part, these disc drives require that the disc be manually clamped once the disc is inserted (before rotation) and, manually removed after use. As will be seen, the invented disc drive provides automatic clamping of a disc, and also automatic ejection. Importantly, these two features are obtained without additional drive mechanisms. Both features result from movement of the linear actuator used to position the magnetic heads. Several attempts have been made to provide a reliable floppy disc drive at a reasonable cost which reads and writes information onto both sides of a disc. In one prior art disc drive, the upper and lower heads are disposed one directly above the other. This has proven to be a cumbersome and unreliable arrangement, particularly since it requires movement of one of the heads in order to insert and remove the floppy disc. The invented disc drive permits access to both sides of the disc. With a unique arrangement of the magnetic heads, neither head is moved for the insertion or removal of a disc.

Floppy disc drives have become widely used in countless computer systems including the personal computer field. This wide distribution has increased the need for an inexpensive, yet reliable disc drive. As will be seen, the described disc drive is readily assembled with fewer critical parts when compared to prior art drives. The simplicity of the overall design provides improved reliability.

SUMMARY OF THE INVENTION

A floppy disc drive which includes automatic disc clamping and ejection is described. A drive assembly which includes a spindle assembly for engaging and rotating a disc is mounted on a base. A carriage assembly is secured on rails to the base for reciprocating movement under the control of a linear actuator. The carriage extends about opposite sides of the drive wheel. A first magnetic head is mounted on the carriage on one side of the drive wheel for engaging one surface of the disc. A second magnetic head is mounted on the carriage on the opposite side of the drive wheel for engaging the opposite side of the disc. Clamping means including a clamper, automatically engages the disc and drive wheel when the carriage is actuated. An overcenter mechanism disposed between the clamping means provides this automatic engagement and disengagement. A springloaded ejector is loaded by the manual insertion of the disc. A ramp on the carriage releases the ejector body when the carriage is moved into a predetermined position, thereby ejecting the disc.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a plan view of the invented disc drive showing the lifter arm in its lower position.

FIG. 2 is a cross-sectional elevation view of the disc drive of FIG. 1 generally taken through section line 2—2 of FIG. 1.

FIG. 3 is a cross-sectional elevation view of the disc drive of FIG. 1 generally taken through section line 3—3 of FIG. 1.

FIG. 4 is a cross-sectional elevation view of the disc drive of FIG. 1 showing the lifter arm in its raised position, this view is generally taken through the staggered section line 4—4 of FIG. 1.

FIG. 5 is a cross-sectional front view of the disc drive taken generally through section line 5—5 of FIG. 1.

FIG. 6 is an exploded view of a portion of the spindle assembly of FIG. 5.

FIG. 7 is a plan view of the drive wheel of the spindle assembly, generally taken through section line 7—7 of FIG. 6.

FIG. 8 is a plan view of the clamper taken generally through section line 8—8 of FIG. 6.

FIG. 9 is a cross-sectional elevation view of the disc drive taken generally through section 9—9 of FIG. 1. This view is used to show the movement of the ejector mechanism.

FIG. 10 is a perspective view showing, in assembly form, the clamper, drive wheel and clamper trust bearing.

FIG. 11 is a perspective view showing the ejector body and its engagement with a disc.

FIG. 12 is a vertical cross-section of a photo-sensor assembly used to calibrate the position of the carriage.

DETAILED DESCRIPTION OF THE INVENTION

A disc drive is described which is particularly suitable for use with floppy discs. In the following description, numerous specific parts are described in detail in order to provide a thorough understanding of the present invention. However, it will be obvious to one skilled in the art that the inventive concepts described may be employed without the described embodiments. In other instances, well-known parts have not been described in detail in order not to obscure the present invention in unnecessary detail.

The main components of the invented disc drive generally comprise: (1) a base 15 shown well in the cross-sectional elevation view of FIG. 5; (2) a spindle assembly 36 which includes the drive wheel 42 and the related drive motor and pulleys also shown in FIG. 5; (3) a carriage 17, driven by a linear actuator which includes motor 32, the magnetic heads 22 and 23 are affixed to this carriage (See FIG. 1); (4) a lifter arm 51 which includes a clamper 70 for clamping a disc to the drive wheel 42 (See FIG. 1); and, (5) an ejector mechanism for ejecting a disc which includes the ejector body 89 of FIG. 11. Other miscellaneous components include function switches 107 and 108 and other parts as shall be described.

In the presently preferred embodiment the body 15 (See FIG. 1 and FIG. 5) is a generally elongated, H-shaped metal casting which includes an upper surface 24 on which the carriage assembly and lifter arm are mounted. A hub and centrally disposed bore are formed in the body 15 allowing it to receive the spindle assembly 36. Grooves 21 are formed on opposite inner surfaces of the upright sides of the body 15 so that a disc 60 may be retained within the body. Numerous other attaching points, bosses, and the like are integrally formed with the body 15; they are described below, where pertinent to the present invention.

The spindle assembly includes at its upper end, a drive wheel 42 best illustrated in FIGS. 5, 6, and 7. The wheel 42 is coupled through a shaft to a flywheel 40. Bearings are provided to allow the wheel, shaft and flywheel 40 to rotate freely within the body 15. The upper surface of the wheel 42 includes an annular surface 46, the inner circumference of which aligns with the centrally disposed bore of a floppy disc. A recess 43 is defined by the wheel 42 within the interior of the annular surface 46. Fingers 78 from the clamper 70 extend into this recess as will be described. As best shown in FIGS. 7 and 10, a centering cylinder 53 extends upward from the central portion of the wheel 42, which as will be discussed below, surrounds the central portion of clamper 70 during operation to insure direct coupling and alignment between wheel 42 and clamper 70. A radially disposed nub 44 extends upward from the interior of centering cylinder 53 to provide a direct drive coupling engagement with teeth 76 of clamper 70.

The rim of the drive wheel 40 is crowned to receive a belt which interconnects the drive wheel with a pulley wheel 41. This pulley wheel is directly driven by an electric motor 45. Once the motor is activated with a disc in place, the disc is rotated at a predetermined rate of rotation.

Referring primarily to FIGS. 1 and 4, the carriage 17 includes a lower, generally rectangular section and an integrally formed upper carriage section 48. The carriage 17 includes sleeves 20 which engage the rails 18. These rails are mounted at mounts 19 above the surface 24 of body 15. The carriage is thus able to move in a reciprocating fashion along the rails from one end to the other of the body 15. The carriage 17 includes a generally centrally disposed, elliptically shaped opening 39 (FIG. 1) which encircles the disc drive wheel 42.

The carriage is driven by a linear actuator which consists of a stepping motor 32, a lead screw 33 and a lead screw nut 34 which is attached to the carriage.

A pair of magnetic heads are affixed to the carriage on opposite sides of the drive wheel 42. The first head 22 is affixed to the forward portion of the carriage 17 (see FIG. 1) and faces upward so that it may contact the underside of a disc engaging the disc drive. The second magnetic head 23 is affixed to the upper carriage portion 48 and faces downward such that it may engage the upper surface of a disc. It is important to note that with this arrangement both heads remain fixed to the carriage; this allows the heads to remain at a precise fixed distance from one another.

A leaf spring 25 extends from the upper carriage portion 48 as best seen in FIG. 1 to a position above the magnetic head 22. A resilient pad 28 is affixed to the lower surface of the spring 25 over the head 22. The spring 25 urges the pad against the disc 60 assuring that the disc 60 contacts the head 22 when the lifter arm 51 is in its lower position. The spring 25 passes over a portion 51*c* (See FIG. 1) of the arm 51, and thus when the arm is raised, as best shown in FIG. 4, the pad 28 is moved away from the disc. This prevents interference between the disc and the pad 28 when the disc is inserted or removed. Another spring 26 (see FIG. 4) mounted to the carriage includes an upwardly facing resilient pad 29. Spring 26 urges the disc (through pad 29) against the head 23. The body 15 defines two downwardly facing cams 38. When the carriage is moved rearwardly (in the direction indicated by arrow 84) the spring 26 moves downward, away from the disc allowing the disc to be removed.

Referring to FIGS. 1, 2 and 4, the lifter arm 51, like the carriage, is a molded plastic member. This irregularly shaped member includes a U-shaped section centered at the portion 51*c*, and a forward extending beam 51*a* (FIG. 1). The U-shaped sections of the lifter arm 51 are mounted for pivotal movement on pivots 54. These pivots extend inwardly from the upstanding right-angle shaped supports 52 (FIG. 2). The supports 52 extend upwardly from surface 24 of base 15. One end of the U-shaped section of arm 51 is coupled to the base 15 through a spring 55 best seen in FIG. 2. This spring urges the arm into its raised position, for example, the pad 28 is moved away from head 22. The end of the other U-shaped section of arm 51 includes an overcenter mechanism 50.

As best seen in FIGS. 3, 4 and 5, the overcenter mechanism includes a leaf spring 67 which extends rearwardly from the arm 51. The spring is hinged at hinge 69 to an arm 68. The free end of arm 68 includes a wheel 59 which rolls on a horizontal surface of the base 15. The axle of this wheel (pin 62 of FIG. 5) extends into a slot 75. This slot is defined between two downwardly extending portions 73 of the carriage. When the carriage is moved to its full forward position, the pin 62 reaches the end of the slot 75 and then the wheel 59 is urged forward (overcenter) to the position shown in FIG. 3. In this position, the spring 67 urges the lifter arm 51 downward and this provides clamping pressure to assure that the disk rotates with the drive wheel 42. Once the overcenter mechanism is locked, the carriage can move to position the heads on the disk without unlocking the mechanism since slot 75 is wide enough to permit such movement. When the carriage is moved to its full rearward position as shown in FIG. 4, the pin 62 contacts the forward end of the slot 75 causing the overcenter mechanism to unlock and assume the position shown in FIG. 4.

The overcenter mechanism has been found to provide ample clamping pressure, and as is apparent, it operates without any manual assistance. Importantly, no additional actuators are required since the mechanism is actuated by the linear actuator used to position the heads on the disc.

The forwardly extending section 51*a* of the arm 51 (FIG. 1) includes a downwardly facing pad 66. This pad is positioned over a boss 65 which extends upwardly from the base 15. When the arm is in its lower position, the disc with its jacket is disposed between the boss 65 and pad 66. The pressure exerted by the pad against the jacket provides cleaning of the disk in a well-known manner.

Another resilient pad 27 (FIG. 1) is affixed to a lower surface of the arm 51 and when the arm is in its lower position, this pad urges the disc jacket against the disc to also provide cleaning. An upstanding portion of the body 15 extends upward below the pad 27 to provide a lower surface upon which the disc jacket rests.

The portion 51*c* of the lifter arm includes a pair of concentric bores 63 and 64, best seen in FIG. 6. Bore 64 receives the shaft 82 of the clamper 70. The annular shoulder between the bore 63 and 64 provides a surface for locking the flared end of the split shaft 82.

The clamper, best seen in FIGS. 8 and 10, is a molded plastic part which includes the shaft 82 with its flared end, and a plurality of flexible fingers 78. In the presently preferred embodiment, clamper 70 is formed out of NORYL 731, a tradename of the General Electric, Co. An annular surface 80 is formed about the fingers 78 and is made to cooperatively engage the surface 46 of the drive wheel 42. The clamper 70 includes a plurality of concentrically disposed teeth 76 which lockingly engage nub 44 to provide coupling between the clamper and the drive wheel. In operation, as will be discussed, the centering cylinder 53 surrounds the outside circumference of teeth 76 to insure that clamper 70 remains substantially centered in the middle of the spindle assembly, thereby maintaining the disk in an on-center configuration. A thrust bearing 71 is disposed between washers 72 on the shaft 82 when the clamper engages the bore 64. This permits the clamper to freely rotate below the arm 51.

One advantage to the clamper 70 is its ease of assembly onto the lifter arm. The washers and thrust bearing 71 are placed on the shaft 82 and then the shaft 82 is snapped into locking engagement within the bores 63 and 64. The shaft 82 is a split member; there is sufficient resiliency for the flared end of the shaft to readily pass through bore 64 before the flared end locks on the shoulder defined between bore 63 and 64.

Referring to FIGS. 1, 2, and 11, the ejector body 89 comprises an elongated molded plastic member defining upper claws 91 and a lower claw 92 at one end, and a perpendicularly disposed finger 95 at the other end. A notch 90 is cut into the disc jacket to allow claw 92 to more securely grip the jacket. The elongated ejector body 89 slides within a track 97. The track is defined by a an upstanding portion of the body member 15. A small plate 102 and a screw 101 retain the ejector body within the track 97. One end of a spring 93 is coupled to the ejector body, the other end of this spring is secured to the forward portion of the body 15. This spring urges the ejector body forward (towards the end of the drive which receives the discs).

The body 15 defines a forward sloping ramp 99, while the carriage includes a ramp 100. When a disc is inserted into the drive, the manual insertion of the disc urges the finger 95 over ramp 99 and causes it to be latched behind the ramp. The ramp 100 when moving in the direction of arrow 84, lifts the finger from its latched position, allowing the spring to move the body member forward thereby ejecting the disc. (Note when the ramp 100 moves in the direction opposite to arrow 84, it does not affect the latched finger 95.) The operation of the ejector mechanism shall be described in greater detail in conjunction with FIG. 9.

Referring now to FIG. 1, the disc drive includes function switches 107 and 108. When a disc is inserted into the drive, these switches are opened (the jacket moves the contacts apart) unless a notch is into the jacket. In FIG. 1, a notch 60*a* is shown around switch 108 to illustrate that with this notch, switch 108 remains closed. These switches may be used in a plurality of different ways. One switch is used for a protective function and prevents erasing of certain discs, for example, those containing programs. It will be appreciated that while switches 107 and 108 are used in the presently preferred embodiment, both switches may be replaced with a light emitting diode (LED) and photodector combination to achieve substantially the same result. Thus, upon insertion of a disk into the drive, the disk jacket would interrupt the beam emitted by the LED and thereby open or close the circuit in accordance with the particular function desired.

In operation, prior to the insertion of a disc, the carriage is driven by the linear actuator (motor 32 and lead screw 33) to its full rearward position as shown in FIG. 4. This causes the lifter arm 51 to be moved to its upward. In this position, the clamper 70 is moved clear of the drive wheel 42 and the pads, such as pad 28 and 29 are moved clear of the magnetic heads. A disc may be inserted into the disc drive along the grooves 21 shown in FIG. 5.

Once the disc is in place, the linear actuator is activated, causing the carriage to move forward. Upon the first forward movement of the carriage, the wheels 59 roll forward locking the overcenter mechanism (lifter arm down).

Referring to FIG. 6, with disc 60 inserted with the disc drive, the aperture of the disc should be concentric with the drive wheel 42. In this position, the edge of the disc aperture should precisely rest on annular surface 46. As clamper 70 moves downward, centering cylinder 53 surrounds the outer circumference of teeth 76 to insure that clamper 70 is precisely aligned with drivewheel 42. In practice, it has been found that without cylinder 53 nub 44 during operation tends to drive clamper 70 off-center relative to drive wheel 42. Typically, the fingers 78 of clamper 70 as they move downward into recess 43, urge the disc into concentric registry with the drivewheel. However, the use of centering cylinder 53 insures that precise alignment and a direct engagement between teeth 76 and nub 44 is achieved each time a disk is inserted.

With the clamper in its down position, the disc is held in place between the annular surface 80 of the clamper and the corresponding surface 46 of the drive wheel 42. Also, the nub 44 is urged into engagement with the teeth 76 of the clamper, thereby providing positive coupling between the clamper and the drive wheel. As will be appreciated, the direct coupling between the clamper and the drive wheel provides a driving force of equal magnitude on both surfaces of the disk. The springs 67 provide sufficient pressure to assure that the disc 60 rotates, without slippage, between with the drive wheel 42 and clamper 70.

With reference once again to FIG. 1, the linear actuator drives the carriage fully forward until a wedge shaped blade 57 formed integrally with the carriage interrupts a light beam within a calibration photo-sensor 58. Photo-sensor 58 is mounted, in the presently preferred embodiment, to the base 15 generally adjacent to the carriage near boss 65. As illustrated in FIG. 12, photo-sensor 58 is generally U-shaped and includes a photo-emitter 61, such as for example an LED, and a corresponding photo-detector 71. Emitter 61 and detector 71 are spaced apart so as to allow blade 57 to pass therebetween. Once the carriage moves forward sufficiently to interrupt the light beam, electrical circuitry driving motor 32 senses this interruption and the position of the carriage is calibrated. Thus, both magnetic heads are in predetermined positions with respect to the drive wheel and disc, such that the position of the carriage relative to the disc tracks may be determined after its subsequent movement along the rails 18.

The motor 45 may now be actuated and the disc brought up to speed. The carriage is moved by the linear actuator to the desired track to allow information to be read from or written onto the disc in a well-known manner. Of course, with the opposite facing heads 22 and 23 both sides of the disc can be accessed without removal of the disc.

When it becomes necessary to remove the disc, the linear actuator drives the carriage to its full rearward position as shown in FIG. 4, causing the lifter arm to raise, thereby freeing the disc.

Referring now to FIG. 9, as mentioned, as the disc is initially inserted, the ejector body 89 is moved rearwardly tensioning spring 93. The finger 95 is shown in three positions in FIG. 9 to illustrate its movement. Finger 95*a* illustrates the position of the finger when the disc is first inserted. As the ejector body is moved rearwardly by the disc the finger moves over the ramp 99 and locks behind the ramp as shown by finger 95*b*. When the ramp 100, which is part of the carriage, moves rearwardly, it urges the finger over the ramp (finger 95*c*) allowing the spring 93 to pull the ejector body forward, ejecting the disc. Angle 96 illustrates the ejector cam angle, formed when the ramp 100 urges the finger from behind the ramp 99.

Thus, a disc drive has been described which permits access to both sides of a disc. A single linear actuator drives both magnetic heads, provides clamping for the disc and triggers the ejector mechanism. The disc drive has numerous features which makes it easy to assemble and which provide high reliability.

We claim:

1. A disc drive comprising:

a base;

a drive assembly including a drive wheel for engaging and rotating a disc, said drive assembly being mounted to said base;

a carriage assembly mounted on said base for reciprocating movement;

a linear actuator coupled to said base for driving said carriage in said reciprocating movement;

at least one magnetic head mounted on said carriage for engaging said disc;

a lifter arm pivotally mounted on said base;

a clamper rotatably mounted on said lifter arm such that as said lifter arm pivots toward said drive wheel, said clamper is brought into engagement with said drive wheel with said disc therebetween to assure rotation of said disc with said drive wheel;

an overcenter mechanism disposed between said arm and said base, said overcenter mechanism being coupled to said arm to control the pivotal movement of said arm, said mechanism being capable of movement actuated by said reciprocating movement of said carriage such that movement of said

carriage to its full rearward position causes pivotal movement of said arm, said pivotal movement occurring only when said carriage is at its full rearward position, thereby causing said clamper to move into said engagement with said drive wheel and to lift from said engagement,

whereby said disc is automatically clamped for rotation when said linear actuator is actuated.

2. The disc drive defined by claim 1 wherein said drive wheel includes an annular outer surface upon which said disc rests and a recessed surface within said annular surface, and wherein said clamper includes an outer annular surface for engaging said annular surface of said drive wheel.

3. The disc drive defined by claim 2 wherein said clamper includes resilient fingers which extend into said recess of said drive wheel when said clamper is in said engagement with said drive wheel, said fingers for providing alignment of said disc on said drive wheel.

4. The disc drive defined by claim 3 including a thrust bearing disposed between said clamper and said lifter arm.

5. The disc drive defined by claim 1 including coupling means on said clamper and said disc wheel to assure rotation of said clamper with said drive wheel when said clamper is in said engagement with said drive wheel.

6. The disc drive defined by claim 1 wherein said carriage extends about opposite sides of said drive wheel and wherein said one magnetic head is mounted on said carriage on one side of said drive wheel to engage one surface of said disc and wherein a second head is mounted on said carriage on the opposite side of said drive wheel to engage the opposite side of said disc.

7. The disc drive defined in claim 1 further comprising:

an ejector means for ejecting said disc from said disc drive, said ejector means being tripped by said carriage movement thereby causing said disc to be ejected,

whereby said disc is automatically clamped and ejected by movement of said carriage.

8. The disc drive defined in claim 7 wherein said ejector means is spring-loaded by the manual insertion of said disc into said disc drive.

9. The disc drive defined by claim 8 wherein said tripping of said ejector means is caused by a ramped surface on said carriage.

* * * * *

United States Patent [19]

Sander

[11] **Patent Number:** **4,533,909**

[45] **Date of Patent:** **Aug. 6, 1985**

[54] **COMPUTER WITH COLOR DISPLAY**

[75] Inventor: **Wendell B. Sander**, San Jose, Calif.

[73] Assignee: **Apple Computer, Inc.**, Cupertino, Calif.

[21] Appl. No.: **560,529**

[22] Filed: **Dec. 12, 1983**

Related U.S. Application Data

[60] Continuation of Ser. No. 394,801, Jul. 2, 1982, abandoned, which is a division of Ser. No. 150,630, May 16, 1980, Pat. No. 4,383,296.

[51] **Int. Cl.^{3}** **G09F 9/30**
[52] **U.S. Cl.** **340/703**; 340/803; 340/802
[58] **Field of Search** 340/701, 703

[56] **References Cited**

U.S. PATENT DOCUMENTS

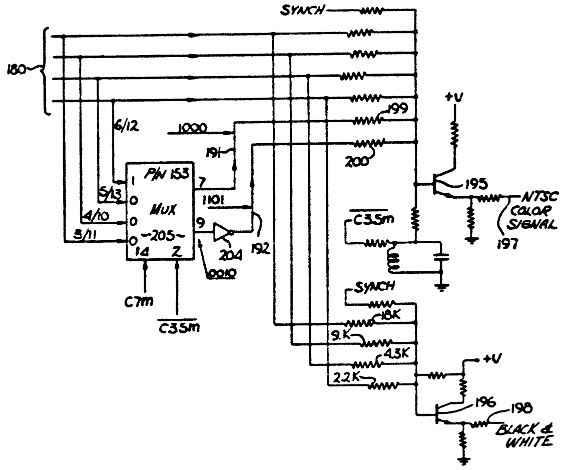
4,136,359	1/1979	Wozniak	358/17
4,310,838	1/1982	Juso et al.	340/701
4,360,804	11/1982	Ohura	340/703

Primary Examiner—David L. Trafton
Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman

[57] **ABSTRACT**

A microcomputer system with video display capability, particularly suited for small business applications and home use is described. The CPU performance is enhanced by permitting zero page data to be stored throughout the memory. The circuitry permitting this capability also provides a pointer for improved direct memory access. Through unique circuitry resembling "bank switching" improved memory mapping is obtained. 4-bit digital signals are converted to an AC chroma signal and a separate luminance signal for display modes. Display modes include high resolution modes, one of which displays 80 characters per line.

11 Claims, 9 Drawing Figures



Apple /// Plus Computer

"APPLE_PAT_4_533_909_01" 451 KB 2000-02-27 dpi: 600h x 600v pix: 3840h x 6140v

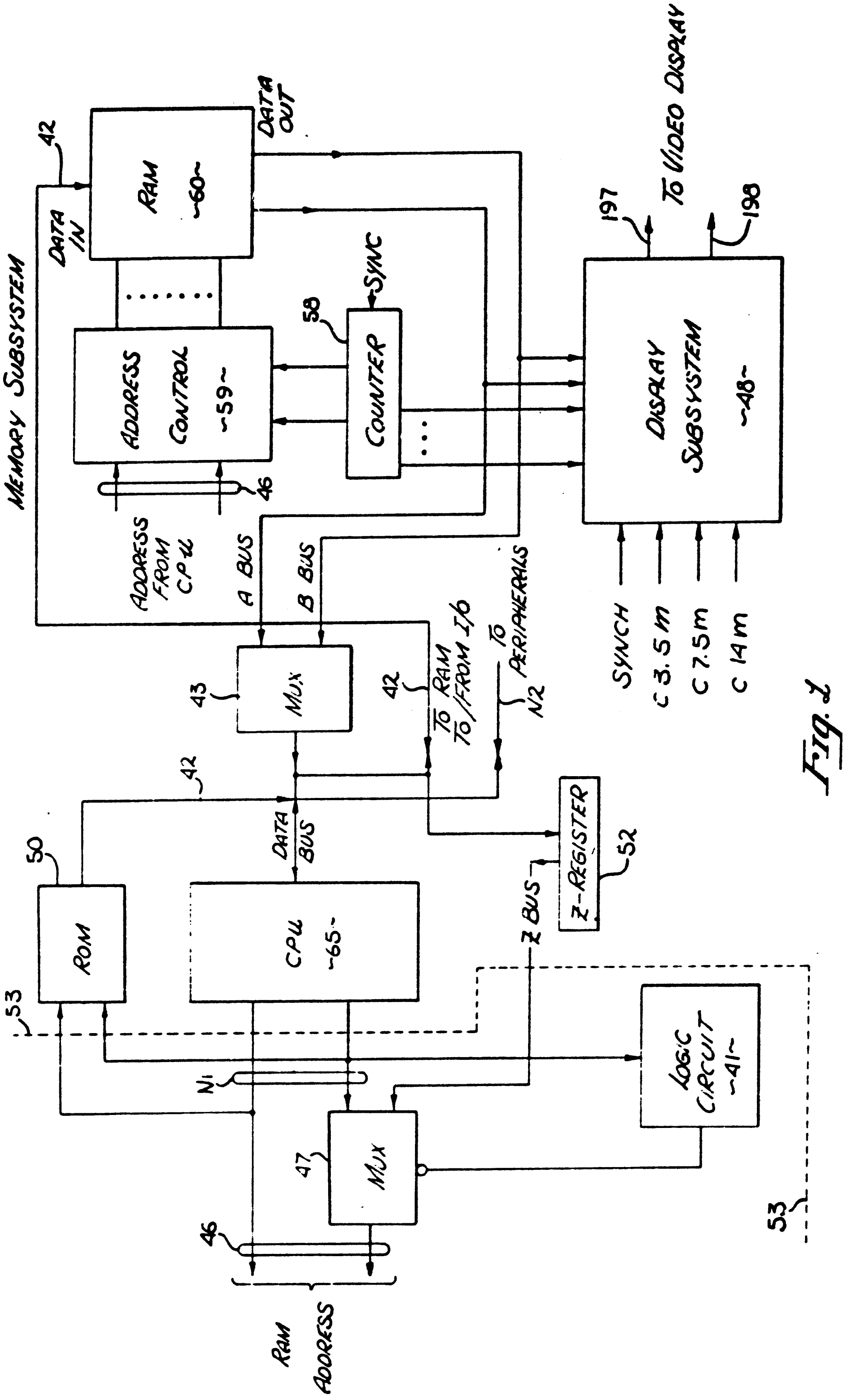

MEMORY SUBSYSTEM
RAM
~60~
DATA IN
DATA OUT
42
ADDRESS CONTROL ~59~
COUNTER
58
SYNC
46
ADDRESS FROM CPU
A BUS
B BUS
TO VIDEO DISPLAY
197
198
DISPLAY SUBSYSTEM ~48~
SYNCH
C 3.5 m
C 7.5 m
C 14 m
MUX
43
42
TO RAM
TO/FROM I/O
TO PERIPHERALS
N2
42
DATA BUS
Z-REGISTER
52
Z BUS
ROM
50
CPU ~65~
53
N1
MUX
47
LOGIC CIRCUIT ~41~
53
46
RAM ADDRESS

Fig. 1

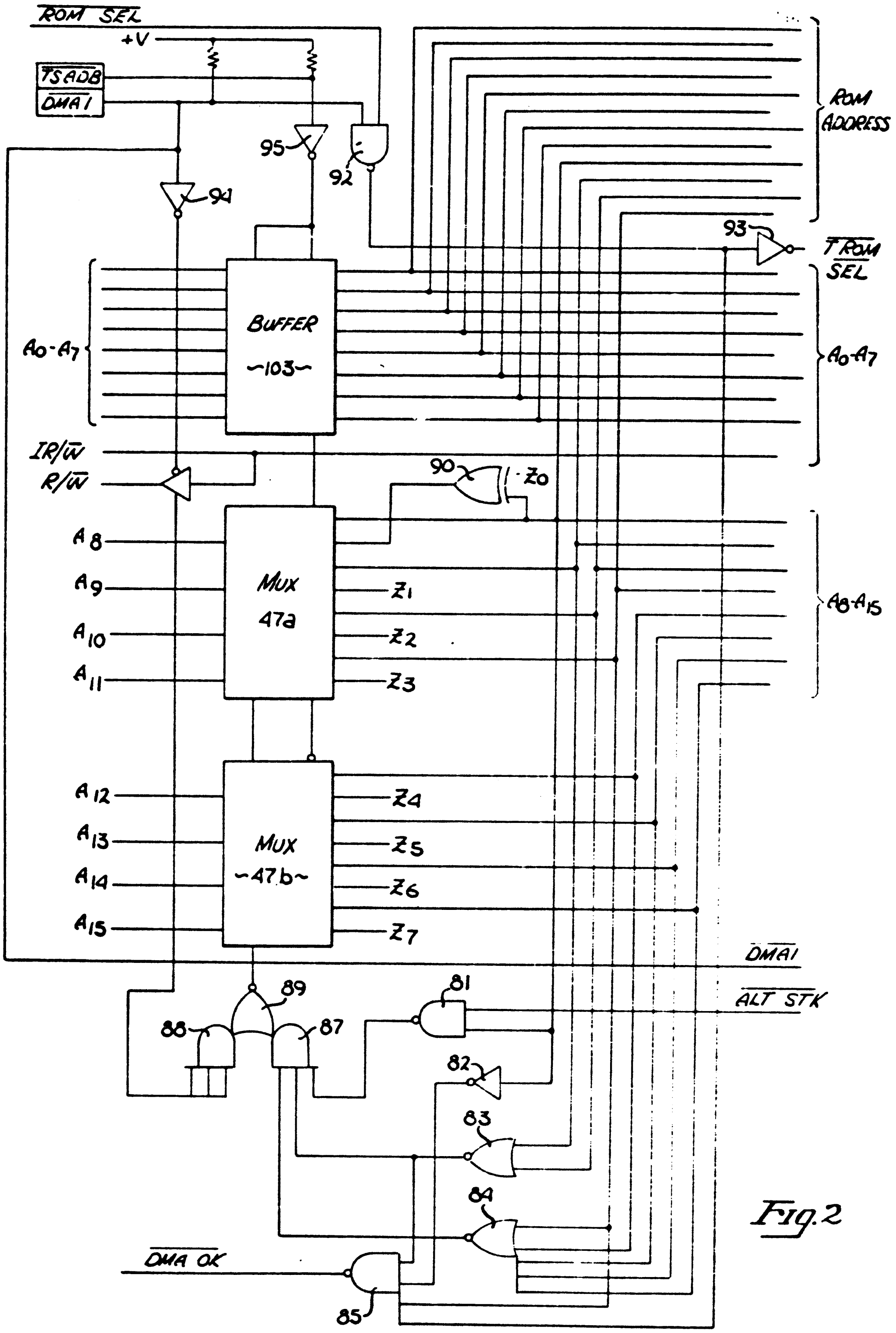


Fig. 2

"APPLE_PAT_4_533_909_03" 399 KB 2000-02-27 dpi: 600h x 600v pix: 3592h x 5681v

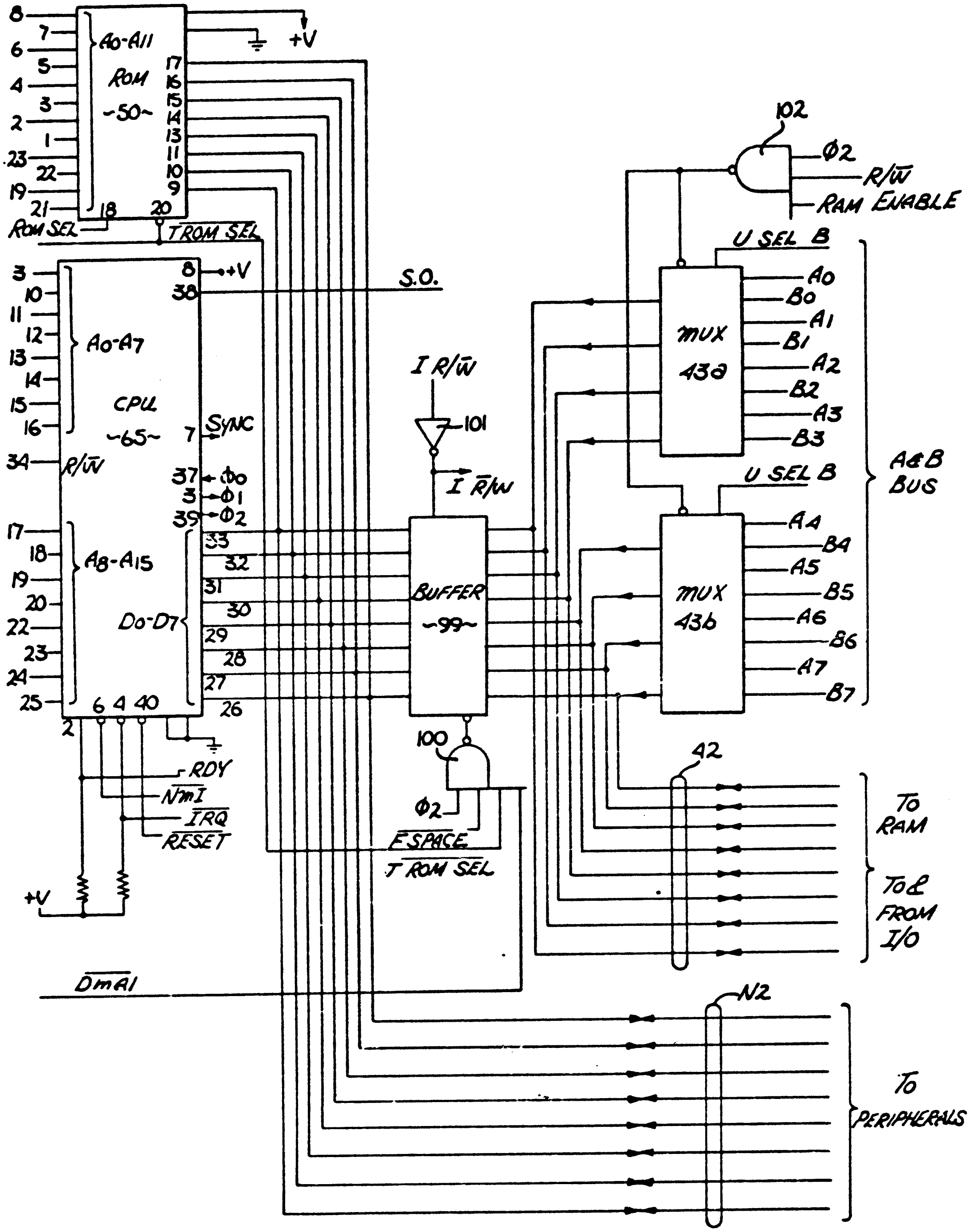

8
7
6
5
4
3
2
1
23
22
19
21
A0-A11
ROM
~50~
17
16
15
14
13
11
10
9
18
20
+V
ROM SEL
T ROM SEL
3
10
11
12
13
14
15
16
A0-A7
CPU
~65~
8
+V
38
S.O.
7
SYNC
34
R/W̄
37
Φ0
3
Φ1
39
Φ2
17
18
19
20
22
23
24
25
A8-A15
D0-D7
33
32
31
30
29
28
27
26
6 4 40
2
RDY
NMI
IRQ
RESET
+V
DMA1
I R/W̄
101
I R/W
BUFFER
~99~
100
Φ2
E SPACE
T ROM SEL
102
Φ2
R/W̄
RAM ENABLE
U SEL B
MUX
43a
A0
B0
A1
B1
A2
B2
A3
B3
U SEL B
MUX
43b
A4
B4
A5
B5
A6
B6
A7
B7
A&B
BUS
42
TO
RAM
TO&
FROM
I/O
N2
TO
PERIPHERALS

Fig. 3

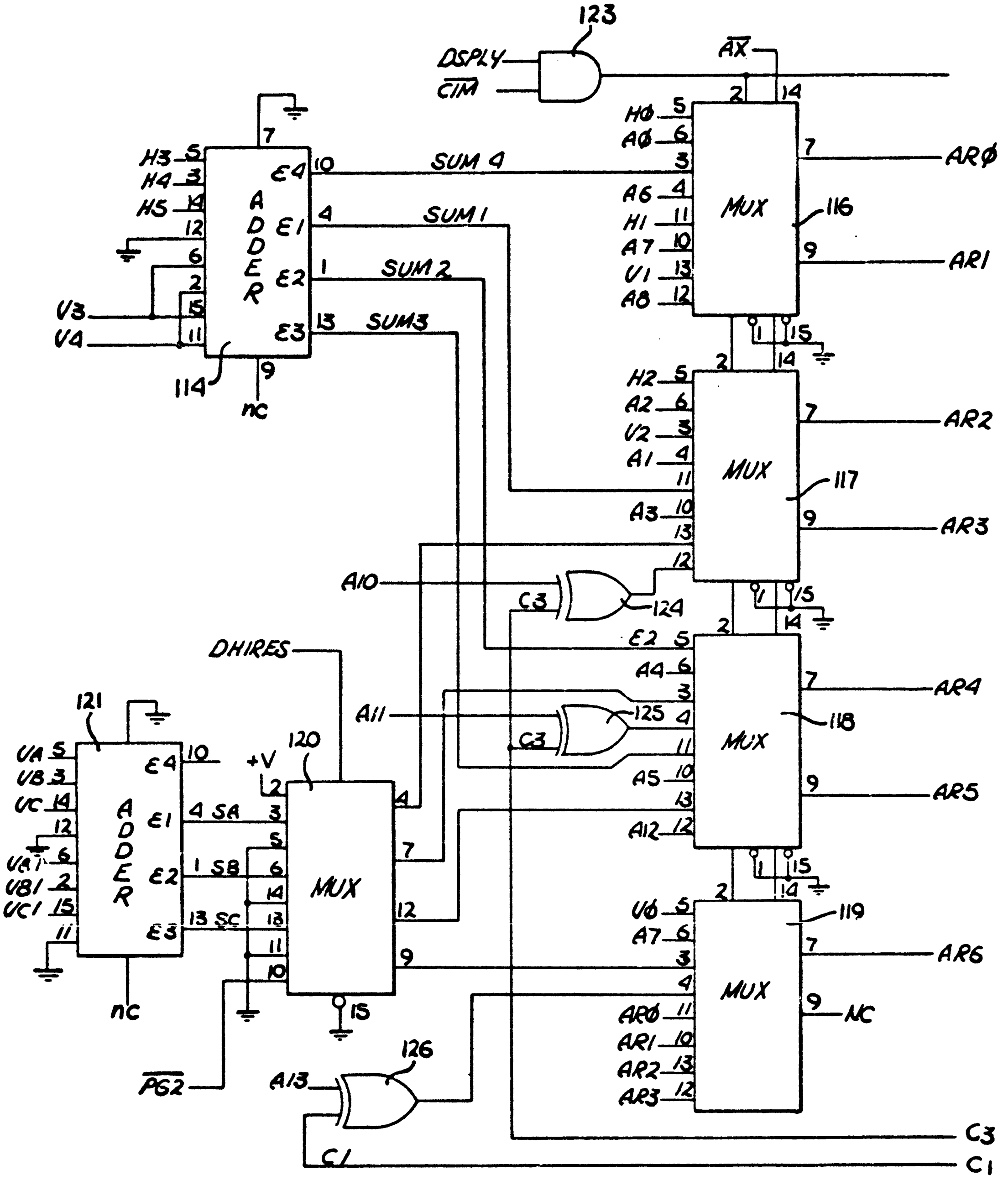

123
DSPLY
CIM
AX
ADDER
114
H3
H4
H5
V3
V4
E4
E1
E2
E3
SUM 4
SUM 1
SUM 2
SUM3
nc
MUX
116
H0
A0
A6
H1
A7
V1
A8
AR0
AR1
117
H2
A2
V2
A1
A3
AR2
AR3
A10
C3
124
E2
A4
A5
A12
118
125
A11
AR4
AR5
DHIRES
121
VA
VB
VC
VA1
VB1
VC1
+V
120
4 SA
1 SB
13 SC
119
V6
AR6
NC
AR2
AR3
PG2
A13
126
C1
C3
C1

Fig. 4

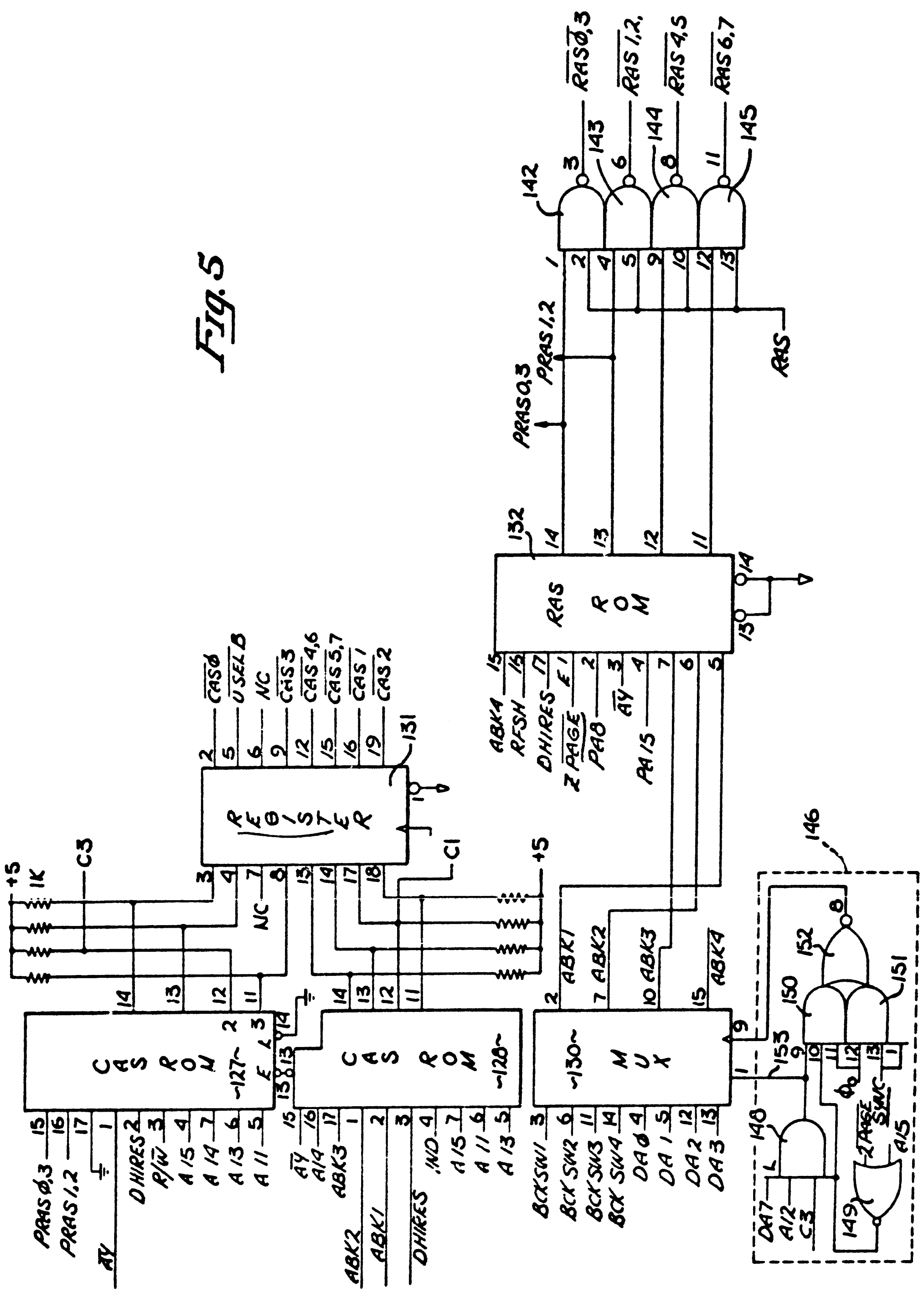


Fig. 5
RAS̄ 0,3
RAS̄ 1,2
RAS̄ 4,5
RAS̄ 6,7
142
143
144
145
RAS
PRAS 0,3
PRAS 1,2
132
RAS ROM
ABK4
RFSH
DHIRES
E1
2 PAGE
PA8
AY
PA15
CAS̄ 0
VSEL B
NC
CAS̄ 3
CAS̄ 4,6
CAS̄ 5,7
CAS̄ 1
CAS̄ 2
131
REGISTER
C1
+5
C3
1K
146
CAS ROM
127
CAS ROM
128
MUX
130
ABK1
ABK2
ABK3
ABK4
150
151
152
153
148
149
PRAS 0,3
PRAS 1,2
AY
DHIRES
R/W
A15
A14
A13
A11
A12
IND
BCK SW1
BCK SW2
BCK SW3
BCK SW4
DA0
DA1
DA2
DA3
DA7
C3
L
2 PAGE
SWC
A15
Φ0

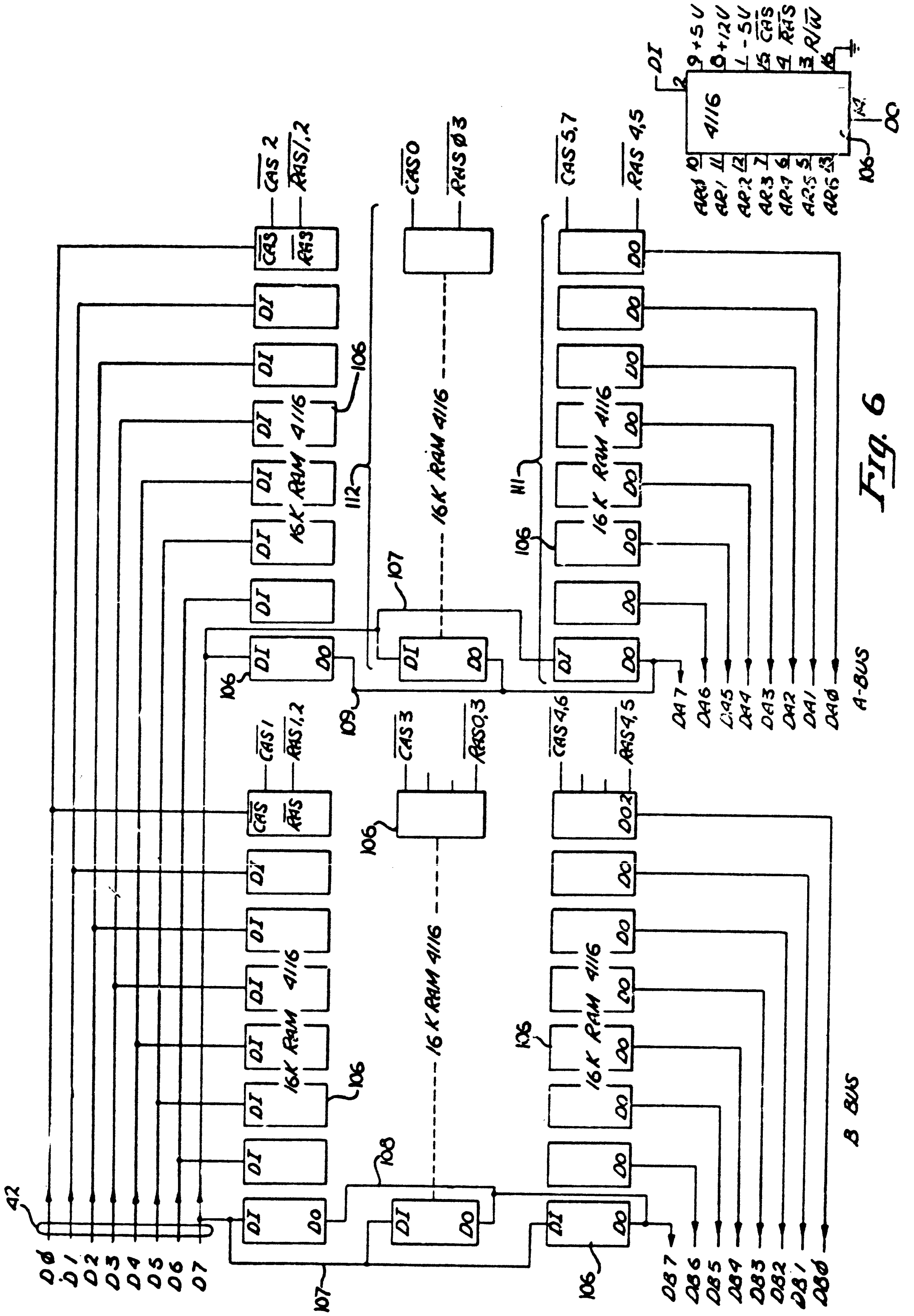


Fig. 6
A-BUS
B BUS
DØ
D1
D2
D3
D4
D5
D6
D7
42
DA7
DA6
DA5
DA4
DA3
DA2
DA1
DAØ
DB7
DB6
DB5
DB4
DB3
DB2
DB1
DBØ
CAS 2
RAS 1,2
CAS0
RAS Ø3
CAS 5,7
RAS 4,5
CAS1
RAS 1,2
CAS3
RAS0,3
CAS 4,6
RAS 4,5
CAS
RAS
DI
DO
DO2
16K RAM 4116
106
107
108
109
111
112
4116
9 +5V
8 +12V
1 -5V
15 CAS
4 RAS
3 R/W
16
2 DI
14 DO
AR0 10
AR1 11
AR2 12
AR3 7
AR4 6
AR5 5
AR6 13

U.S. Patent Aug. 6, 1985 Sheet 7 of 8 4,533,909

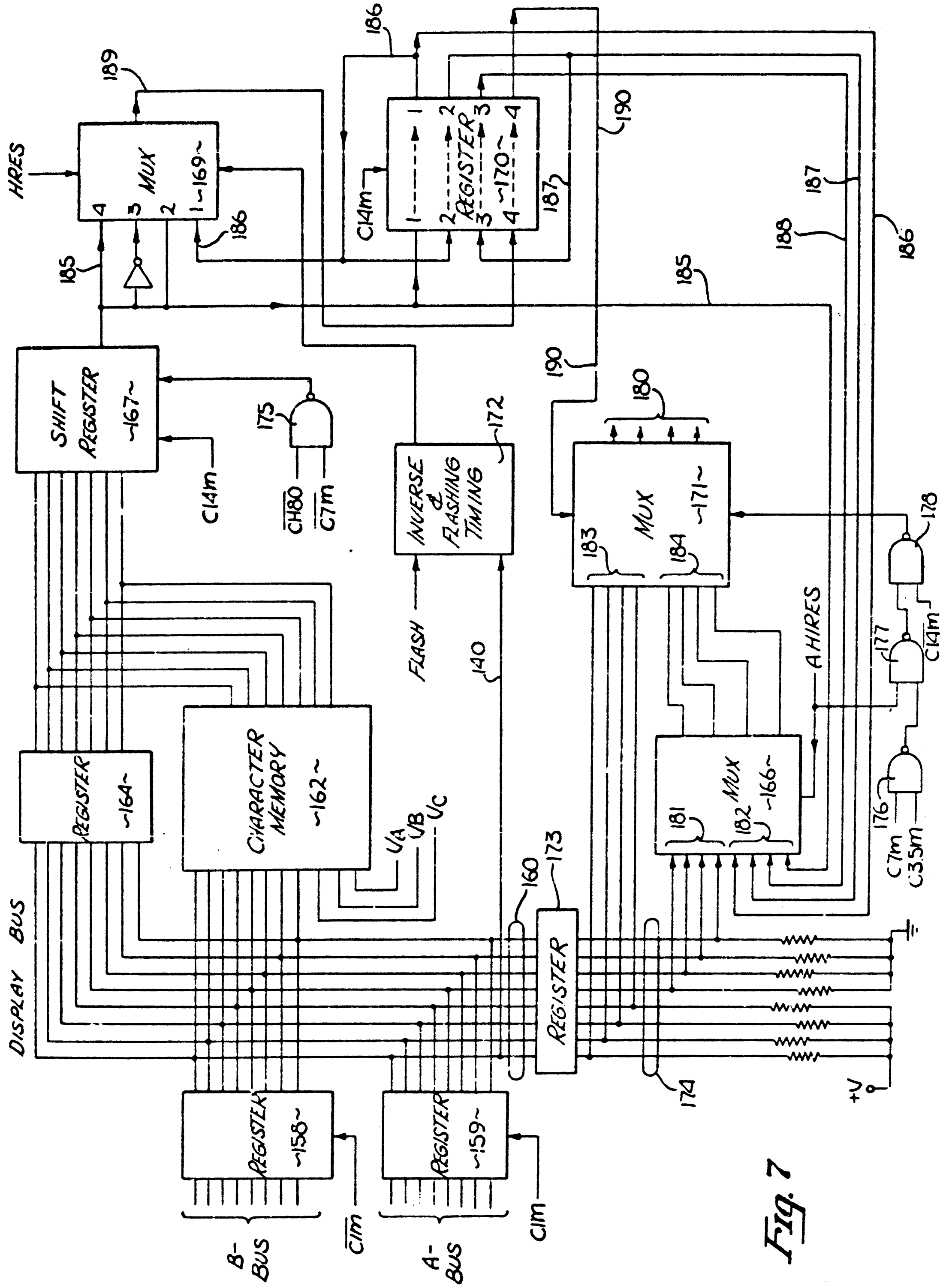


Fig. 7

“APPLE_PAT_4_533_909_08” 533 KB 2000-02-27 dpi: 600h x 600v pix: 3648h x 5508v

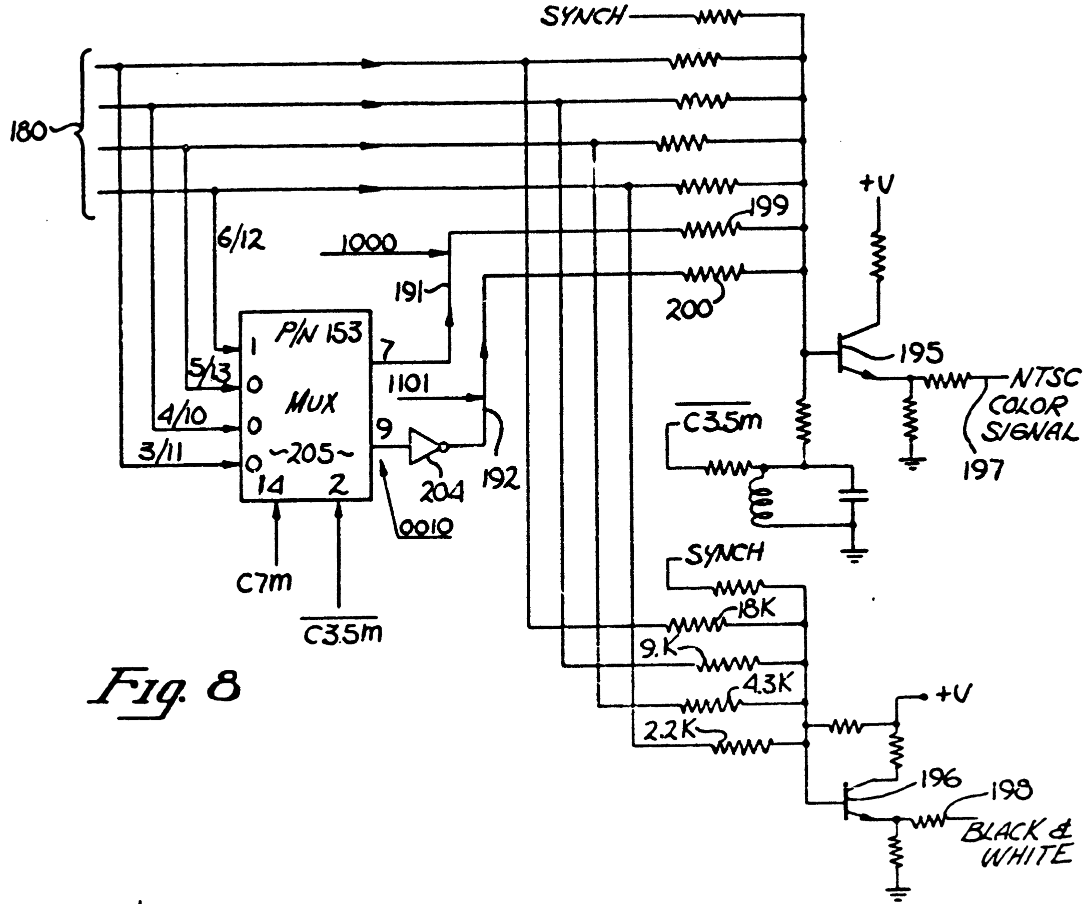

SYNCH
180
6/12
1000
191
199
200
+V
P/N 153
1
0
0
0
5/13
4/10
3/11
MUX
~205~
14
2
7
9
1101
204
192
0010
C7m
$\overline{C3.5m}$
195
NTSC COLOR SIGNAL
197
$\overline{C3.5m}$
SYNCH
18K
9.K
4.3K
2.2K
+V
196
198
BLACK & WHITE

Fig. 8

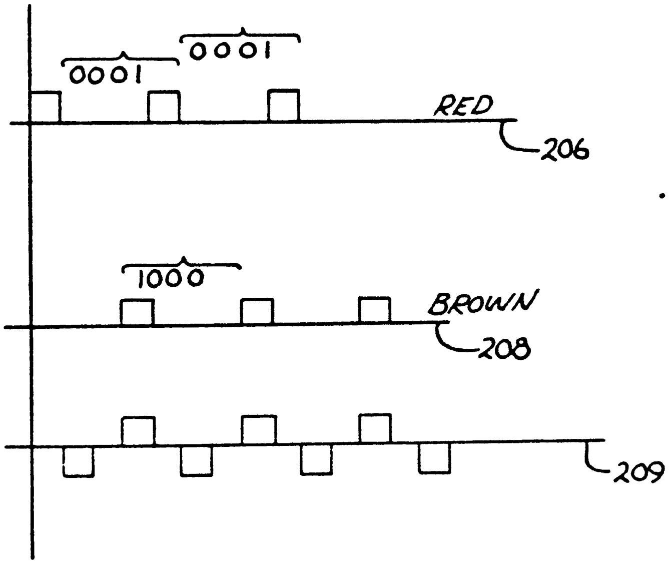

0001
0001
RED
206
1000
BROWN
208
209

Fig. 9

COMPUTER WITH COLOR DISPLAY

This is a continuation of application Ser. No. 394,801 filed July 2, 1982, now abandoned, which is a divisional of application Ser. No. 150,630 filed May 16, 1980, now U.S. Pat. No. 4,383,296.

BACKGROUND OF THE INVENTION

The invention relates to the field of digital computers, particularly microcomputers, having video display capabilities.

Prior Art

In the last few years, there has been rapid growth in the use of digital computers in homes by hobbyists, for small business and for routine engineering and scientific application. For the most part, these needs have been met with self-contained, relatively inexpensive microcomputers or microprocessors with essential peripherals, including disc drives and with relatively easy to manage computer programs. The design of computers for these needs requires considerable ingenuity since each computer must meet a wide range of applications and because this market is particularly cost conscious.

A home or small business computer must, for example, operate with a number of different program languages, including those requiring relatively large memories, such as Pascal. The computer should interface with a standard raster scanned display and provide a wide range of display capabilities, such as high density alpha-numeric character displays needed for word processing in addition to high resolution graphics displays.

To meet these specialized computer needs, generally requires that a relatively inexpensive microprocessor be used and that the capability of the processor be enhanced through circuit techniques. This reduces the overall cost of the computer by reducing, for example, power needs, bus structures, etc. Another important consideration is that the new computers be capable of using programs developed for earlier models.

As will be seen, the presently described microcomputer is ideally suited for home and small business applications. It provides a wide range of capabilities including advanced display capabilities not found in comparable prior art computers.

The closest prior art computer known to applicant is commercially available under the trademark, Apple-II. Portions of that computer are described in U.S. Pat. No. 4,136,359.

SUMMARY OF THE INVENTION

A digital computer which includes a central processing unit (CPU) and a random-access memory (RAM) with interconnecting address bus and data bus is described. One aspect of the present invention involves the increased capability of the CPU by allowing base page or zero page data to be stored throughout the memory. Alternate stack locations and an improved direct memory access capability are also provided by the same circuitry. Detection means are used for detecting a predetermined address range such as the zero page. This detection means causes a special register (Z-register) to be coupled into the address bus. The contents of this Z-register provide, for example, a pointer during direct memory access, or alternate stack locations for storing data normally stored on page one.

The memory of the invented computer is organized in an unusual manner to provide compatibility with the 8-bit data bus and yet provide high data rates (16-bits/MHz) needed for high resolution displays. A first plurality of memory devices are connected to a first memory output bus; these memory devices are also connected to the data bus. The memory includes a second plurality of memory devices which are also connected to the data bus; however, the outputs of these second devices are coupled to a second output memory bus. First switching means permit the first and second memory buses to be connected to the display for high data rate transfers. Second switching means permit either one of the memory buses to be connected to the data bus during non-display modes.

The addressing capability of the memory is greatly enhanced not only through bank switching, but through a novel remapping which does not require the CPU control associated with bank switching. In effect, the "unused" bits from one of the first and second memory buses are used for remapping purposes. This mode of operation is particularly useful for providing toggling between two separate portions of the memory.

The display subsystem of the described computer generates video color signal in a unique manner. A 4-bit color code as used in the prior art, is also used with the described display subsystem. However, this code is used to generate an AC chrominance signal and a separate DC luminance signal. This provides enhanced color capability over similar prior art color displays.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram showing the major components and subsystems of the invented and described microcomputer system.

FIGS. 2 and 3 together show the central processing unit (CPU) and the architecture associated with this CPU, particularly the address bus and data bus. FIG. 2 is a circuit diagram primarily showing the address bus and the logic means associated with this bus. FIG. 3 is a circuit diagram primarily showing the data bus and its interconnection with the memory buses (A bus and B bus), bootstrap read-only memory, and input/output ports.

FIGS. 4, 5 and 6 show the memory subsystem. FIG. 4 is a circuit diagram primarily showing the circuitry for selecting between address signals from the address bus and display counter signals. FIG. 5 is a circuit diagram primarily showing the generation of various "select" signals for the memory devices. FIG. 6 is a circuit diagram showing the organization of the random-access memory and its interconnection with the data bus and memory output buses.

FIGS. 7 and 8 illustrate the display subsystem of the invented computer. FIG. 7 is a circuit diagram showing the circuitry for generating the digital signals used for the video display. FIG. 8 is a circuit diagram of the circuitry used to convert the digital signals to analog video signals.

FIG. 9 is a graph of several waveforms used to describe a prior art circuit and the circuit of FIG. 8.

DETAILED DESCRIPTION OF THE INVENTION

A microcomputer system capable of driving a raster scanned video display is disclosed. In the following description, numerous specific details such as specific part numbers, clock rates, etc., are set forth to provide a thorough understanding of the present invention. However, it will be obvious to one skilled in the art that the inventive concepts described in this patent may be practiced without these specific details. In other instances, well-known circuits have been shown in block diagram form in order not to obscure the present invention in unnecessary detail.

Referring first to FIG. **1**, in general the described computer includes a central processing unit (CPU) **65**, its associated data bus **42**, address bus **46** a memory subsystem and a display subsystem **58**.

The address bus **46** from the CPU is coupled to the memory subsystem to permit the selection of locations in memory. Some of the address signals pass through a multiplexer **47**. For some modes of operation, signals from a register **52** are coupled through the multiplexer **47** onto the bus **46**. The register **52** is identified as the Z-register and is coupled to the multiplexer **47** by the Z bus. The general description of the multiplexer **47** and its control by the logic circuit **41** are described in detail in conjunction with FIG. **2**. In general, the circuitry shown to the left of the dotted line **53** is included in FIG. **2** while the CPU **65**, memory **50**, data bus **42** and multiplexer **43** are shown in detail in FIG. **3**.

The address bus N**1** is coupled to the read-only memory **50**. The output of this memory is coupled to the computer's data bus **42**. The read-only memory (ROM) **50**, as will be described, stores test routines, and other data of a general bootstrap nature for system initialization.

The data bus **42** couples data to the random-access memory (RAM) **60** and to and from I/O ports. This bus also couples data to the Z-register **52** and other commonly used registers not illustrated. The data bus **42** receives data from the RAM **60** through the A bus and B bus which are selected by multiplexer **43**. The peripheral bus N**2** is used, as is better illustrated in FIG. **3**, for coupling to peripherals.

The memory subsystem is shown in detail in FIGS. **4**, **5** and **6**. The address control means which receives addresses on bus **46**, makes the final selection of memory locations within the RAM **60**. Bank switching, addressing for display purposes, scrolling and other memory mapping is controlled by the address control means **59** as will be described in greater detail in conjunction with FIGS. **4** and **5**. The RAM **60** is shown in detail in FIG. **6**. The counter **58** which is sychronized with the horizontal and vertical display signals, provides signals both to the address control means **59** and to the display subsystem **48**.

The display subsystem receives data from the RAM **60** on the A bus and B bus and converts these digital signals to video signals which control a standard raster scanned display. A standard NTSC color signal is generated on line **197** and a black and white video signal on line **198**. The same signals used to generate these video signals can be used to generate separate red, green, blue (RGB) video signals. The display subsystem **48** receives numerous timing signals including the standard color reference signal shown as 3.5 MHz (C3.5M). This subsystem is described in detail in FIGS. **7** and **8**.

COMPUTER ARCHITECTURE

In the presently preferred embodiment, the CPU **65** (microprocessor) employed with the described computer is a commercially available component, the 6502A. This 8-bit processor (8-bit data bus) which has a 16-bit address bus is shown in FIG. **3** with its interconnections to the remainder of the computer. The pin number for each interconnection is shown adjacent to the corresponding line. In many cases, the nomenclature associated with the 6502A (CPU **65**) is used in this application. For example, pin **6** receives the nonmaskable interrupt signal ($\overline{\text{NMI}}$), and pin **4** is coupled to receive the interrupt request signal ($\overline{\text{IRQ}}$). Some of the signals employed with the CPU **65**, which are well-known in the art, and which are not necessary for the understanding of the present invention are not described in detail in this application, such as the various synchronization signals and clocking signals. The address signals from the CPU **65** are identified as A_0–A_7 and A_8–A_{15}. The data signals associated with the CPU **65** are shown as D_0–D_7. As will be apparent to one skilled in the art, the inventive concepts described in this application may be employed with other microprocessors.

Referring now to FIGS. **2** and **3**, the general architecture, particularly the architecture associated with the CPU **65** can best be seen. The address signals A_0–A_7 are coupled to a buffer **103** by the bus shown primarily in FIG. **2**. These address signals are also coupled to the ROM **50**. The signals A_0–A_7 after passing through the buffer **103** are coupled to the memory subsystem. The address signals A_8–A_{15} (higher order address bits) are coupled through lines shown in FIG. **2** to the multiplexers **47*a*** and **47*b***. The contents of the Z-register **52** of FIG. **1** is also connected to the multiplexers **47*a*** and **47*b*** through the Z-bus (Z_1–Z_7). The multiplexers **47*a*** and **47*b*** allow the selection of either the signals A_8–A_{15} from the CPU **65** or the contents of the Z-register (Z_1–Z_7) for addressing the RAM **60**. The output of these multiplexers are shown as A_8–A_{15}; this designation is used even when the Z-bus is selected. Note in the case of the Z_0 signal, this signal is coupled to the multiplexer **47*a*** through the exclusive OR gate **90** for reasons which are explained later. The address signals A_8–A_{11} are also coupled to the ROM **50**, thus the signals A_0–A_{11} are used for addressing the ROM **50**. The signals A_8–A_{15} are connected to the logic circuit shown in the lower left-hand corner of FIG. **2**; this logic circuit corresponds to the logic circuit **41** of FIG. **1**.

The input and output data signals from the CPU **65** are coupled by a bidirectional bus to the bidirectional buffer **99** (FIG. **3**). This buffer is selectively disabled by gate **100** to allow the output of ROM **50** to be communicated to CPU **65** and during other times not pertinent to the present discussion. The direction of flow through the buffer **99** is controlled by a read/write signal coupled to the buffer through inverter **101**. Data from the CPU **65** is coupled through the buffer **99** and bus **42** to the RAM **60** or to I/O ports. Data from the RAM **60** is communicated to CPU **65** or bus N**2** from the A bus and B bus through the buffer **99**. The 4 lines of the A bus and 4 lines of the B bus are coupled to the multiplexer **43*a***. Similarly, the other 4 lines of the A and B buses are coupled to the multiplexer **43*b***. Multiplexers **43*a*** and **43*b*** select the 8 lines of the A bus or B bus and communicate the data through to buffer **99** and bus **42**. These multiplexers are selectively disabled (for example, during writing) by gate **102**. As will be described later, the 16 lines of the A bus and B bus permits the reading of 16-bits from the RAM at one time. This provides a data rate of 16-bits/MHz which is necessary, for example, for an 80 character per line display. The data is loaded into the RAM **60**, 8-bits at a time.

The ROM 50, as mentioned, stores test programs, data needed to initialize various registers, character generation data (for RAM 162 of FIG. 7) and other related data. Specific programs employed in the presently preferred embodiment of the computer are set forth in Table 1 of U.S. Pat. No. 4,383,296. The ROM 50 is selected by control signals coupled to its pins 18 and 20, identified as signals ROM SEL and $\overline{\text{T ROM SEL}}$. Any one of a plurality of commercially available read-only memories may be used for the ROM 50. In the presently preferred embodiment, commercially available Part No. SY2333 is used.

Referring now to this logic circuit (lower left-hand corner of FIG. 2), the NAND gate 81 receives the address signal A_8 and also the alternate stack signal identified as $\overline{\text{ALT STK}}$. The output of this gate provides one input to the AND gate 87. The A_8 signal is also coupled through the inverter 82 to one input terminal of the NAND gates 85 and 86. The address signals A_9 and A_{10} are coupled to the input terminals of the NOR gate 83. The output of this gate is coupled to one input terminal of the NAND gates 85 and 86 and the AND gate 87. The address signals A_{11}–A_{15} are coupled to the input terminals of the NOR gate 84. The signal A_{11} is also coupled to an input terminal of the NAND gate 85.

The outputs of the AND gates 87 and 88 (through NOR gate 89), controls the multiplexers *47a* and *47b*. When the output of gate 89 is low the Z-bus is selected, otherwise the address signals from the CPU 65 are selected.

The logic circuit above-described, along with the Z-bus and Z-register provide enhanced performance for the computer. First, this circuit permits the zero page or base page data to be stored throughout the RAM 60 rather than just on zero page. Secondly, this circuit enables addressing of alternate stack locations (other than page one). Lastly, this circuit through the Z-register provides a RAM pointer for direct memory access (DMA).

Assume for purposes of discussion that the CPU 65 is addressing the zero page of memory. That is, the higher order address bits A_8–A_{15} are all zeros. The zeros for A_9–A_{15} are detected by the gates 83 and 84. If all the inputs to these gates are zeros, the outputs of these gates are high which condition is communicated to the gate 87. A_8 which is also low, insures that the output of gate 81 will be high. Thus, all the inputs to gate 87 are high, causing the signal at the output of the gate 89 to drop. When this occurs, the Z-bus is selected. Instead of all the binary zeros from the CPU being coupled to the main memory (RAM 60), the contents of the Z-register form part of the address for the memory. Therefore, even though the CPU 65 has selected the zero page, nonetheless data may be written into or from any location of RAM 60 (including the zero page). This enhances the performance of the CPU, since for example, the time consumed in shifting data to and from a single zero page is minimized.

Normally, the CPU 65 selects page one for stack locations. This occurs when A_8 is high and A_9–A_{15} are low. Assume first that the alternate stack locations have not been selected. Both inputs to gate 81 are high and its output is low. The low input to the gate 87 prevents the selection of the Z-bus. Thus, for these conditions the address signals A_0–A_7 select stack locations on page one.

Next assume that page one has been selected by the CPU and that the $\overline{\text{ALT STK}}$ signal is low, indicating the alternate stack locations are to be selected. (A flag is set by the CPU to change the $\overline{\text{ALT STK}}$ signal). Since the $\overline{\text{ALT STK}}$ signal is low and A_8 is high, a high output occurs from the gate 81. All the inputs to gates 83 and 84 are low, therfore, high outputs occur from both these gates. The conditions of gate 87 are met, causing a high output from this gate and lowering the output from the gate 89. The Z-bus is thus selected by the multiplexers *47a* and *47b*. This allows the contents of the Z-register to be used as alternate locations. Non-zero page locations are assured by inverting A_8. The exclusive OR gate 90 acts as a selective inverter. If A_8 is high and Z_0 is low, then A_8 at the output of the multiplexer *47a* will be low. Note that during zero page selection when A_8 is low, the Z_0 signal is directly communicated through gate 90 to the output of multiplexer *47a*.

Thus, the logic circuits along with the $\overline{\text{ALT STK}}$ signal allows alternate stack locations to be selected through the Z-bus. This further enhances the performance of the CPU which would otherwise be limited to page one for stack locations.

The logic circuit of FIG. 2 is also used along with the Z-register to provide a pointer during direct memory access (DMA). Assume that direct access to the computer's memory is required by a peripheral apparatus. To initiate the DMA mode the CPU provides an address between F800 and F8FF. Through a logic circuit not illustrated in FIGS. 2 and 3, the $\overline{\text{ROM SEL}}$ signal is brought low for addresses between F000 and FFFF. This signal is communicated to gate 93 and causes the output of gate 92 to rise ($\overline{\text{DMA1}}$ is high at this time). This rise in potential is communicated to one input of the gate 85. Additionally, gate 85 senses that the address bits A_8, A_9 and A_{10} are low This information is coupled to gate 85 through the inverter 82 and the NOR gate 83 as high signals. Also the fact that A_{11} is high is directly communicated to gate 85 Thus with the address between F800 and F8FF the $\overline{\text{DMA OK}}$ signal drops in potential. This is sensed by the peripheral apparatus which in turn causes the $\overline{\text{DMA 1}}$ signal to drop and provides a ready signal to the CPU 65. With the completion of this handshake, data may begin to be transferred to the RAM.

The $\overline{\text{DMA 1}}$ signal through gate 93 and inverter 93 forces the $\overline{\text{T ROM SEL}}$ signal low. This signal in addition to being communicated to the ROM 50, is coupled to the buffer 99 through gate 100, disabling this buffer (during the reading of ROM 50). Also, the ready signal causes the CPU to come to a hard stop. Importantly, the $\overline{\text{DMA 1}}$ signal, after passing through the inverter 94 and the gates 88 and 89, assures the selection of the Z-register. The contents of the Z-register are fixed and provide a pointer to a page in the RAM.

Under the above conditions, the CPU increments the lower 8-bits of the address signal. The ROM 50 furnishes the instructions for incrementing the address, specifically SBC #1 and BEQ. The peripheral apparatus provides the data or receives the data in synchronization with the CPU operation. The peripheral also furnishes a read/write signal to indicate which operation is to occur. Data is then written into RAM via bus N2 and bus 42, or read from RAM via the A and B buses and bus N2.

Importantly, with the above DMA arrangement, addresses from the peripheral apparatus are not necessary and the Z-register is used to provide a pointer to a page in RAM 60.

MEMORY SUBSYSTEM

The memory subsystem shown in FIG. 1 as the address control means 59 and RAM 60 is illustrated in detail in FIGS. 4, 5 and 6 as mentioned. In FIGS. 4 and 5, the memory control means is shown, while in FIG. 6 the memory devices and their organization are illustrated. The address control means of FIGS. 4 and 5 receives the address signals from the CPU 65 (A_0–A_{15}), the count in the vertical and horizontal counters (counter 58 of FIG. 1) which are used during display modes, control signals from the CPU and other signals. In general, this control means develops the address signals which are coupled to the RAM of FIG. 6 including the column address and row address signals, commonly referred to as $\overline{CAS}$ and $\overline{RAS}$. Other related functions are also shown in FIGS. 4 and 5, such as the circuitry which provides display scrolling, indirect RAM addressing and memory mapping.

The CPU 65 of FIG. 3 provides a 16-bit address for addressing the memory. Under ordinary circumstances this address limits the memory capacity to 64 K bytes. This size memory is insufficient in many applications, as for example, to effectively use the Pascal program language. As will be described in greater detail, the address control means of FIGS. 4 and 5 enable the use of a memory having a 96K byte or 128K byte capacity. One well-known technique which is used with the present invention for increasing this capacity is bank switching; this switching occurs under the control of the CPU. In addition, the address control means uses a unique indirect addressing mode which provides the benefits of bank switching, however, this mode does not require CPU control. This greatly enhances CPU operation with the larger memory (as will be described) when compared to the CPU controlled bank switching.

Referring first to FIG. 6, the RAM configuration is illustrated for a capacity of 96K bytes. The memory is organized into six rows, each of which includes eight 16K memory devices such as rows 111 and 112. In the presently preferred embodiment, Part No. 4116 MOS dynamic RAMs are used. (The pin designations and signal designations refer to this memory device.) Obviously, other memory devices may be employed.

Input data to these memory devices 106 is provided from the bus 42. Each line in the bus 42 is connected to the data input terminal of one device 106 in each row. The interconnection of this bus with each of the memory devices is not shown in FIG. 6 in order not to overcomplicate this drawing. By way of example, however, line 107 connects the data bit D7 to the data input terminal of one of the memory devices in each of the six rows.

Three rows of devices 106 have their output terminals coupled to the A bus, and three rows are similarly coupled to the B bus. By way of example, line 108 connects three output terminals of devices 106 to the DB7 line of the B bus while line 109 connects three output terminals of the devices 106 to the DA7 line of the A bus.

The described memory devices 106 are each organized as a 16KX1 memory. Thus, each device receives a 14-bit address which is time multiplexed into two, 7-bit addresses. This multiplexing occurs under the control of the $\overline{CAS}$ and $\overline{RAS}$ signals as is well-known. The lines coupling the address signals to each of the devices in FIG. 6 are not illustrated. However, in the lower right-hand corner of FIG. 6, the various signals applied to each device (including the address signals), along with the corresponding pin numbers are shown. Other circuitry not illustrated is the refresh control circuitry which operates in a well-known manner in conjunction with the $\overline{CAS}$, $\overline{RAS}$ and address signals to refresh the dynamic devices.

Each row of memory devices 106 receives a unique combination of $\overline{CAS}$ and $\overline{RAS}$ signals. For example, row 111 receives $\overline{CAS}$ 5, 7 and RAS 4, 5; similarly, row 112 receives $\overline{CAS}$ 0 and $\overline{RAS}$ 0, 3. The generation of these $\overline{CAS}$ and $\overline{RAS}$ signals is described in conjunction with FIG. 5. These signals (along with the 14-bit address signals) permit the selection of a single 8-bit location in the 96K byte memory (for writing) and also the selection (for reading) of 16-bit locations.

The memory of FIG. 6 may be expanded to a 128K byte memory by using 32K memory devices, such as Part No. 4132. In this case, four rows of eight, 32K memory devices are used with each row receiving two $\overline{CAS}$ and $\overline{RAS}$ signals.

Before reviewing FIG. 4, a general understanding of the organization of the display is helpful. The display, during certain modes, is organized into 80 horizontal segments and 24 vertical segments for a total of 1920 blocks. 11-bits of the counter 58 of FIG. 1 are used as part of the address signals for the memory to access data for displaying during these modes. These counter signals are shown in FIG. 4 as H_0–H_5 and V_0–V_4. During other display modes each horizontal segment is further divided into 8 segments (e.g. for displaying 80 alpha numeric characters per line). This requires 3 additional vertical timing signals shown as V_A, V_B and V_C in FIGS. 4 and 7.

Often in the prior art, two separate counters are used to supply the timing/address signals for accessing a memory when the data in the memory is displayed. The count in one counter represents the horizontal lines of the screen (vertical count) and the other the position along each line, (horizontal or dot count). In many prior art displays the most significant bit of the dot counter is used to increment the line counter. Data in memory intended for display is mapped with a one-to-one correlation to the counts in these counters. In another prior art system (implemented in the Apple-II computer sold by Apple Computer, Inc.) this one-to-one correlation is not used. Rather, to conserve on circuitry, a single counter is employed and a more dispersed mapping is used in the memory. (Note that where a maximum horizontal count of 80 is used, this number cannot be represented by all ones in a digital counter and thus the vertical counter cannot easily be incremented by the most significant bit in the horizontal counter.) Since this more dispersed mapping technique is part of the prior art and not critical to an understanding of the present invention, it shall not be described in detail. However, the manner in which it is implemented shall be discussed in conjunction with the adder 114 of FIG. 4. For purposes of discussion, the signals from the counter 58 of FIG. 1 are designated as either vertical (V) or horizontal (H).

Referring now to FIG. 4, the selection of either the counter signals on the address signals from the CPU is made by the multiplexers 116, 117, 118 and 119. Each of these commercially available multiplexers (Part No. 153) couples one of four input lines to an output line. There are eight inputs to multiplexers 116, 117 and 118 and the outputs of these multiplexers provide the address signals for the memories (AR0 through AR5). The multiplexer 119 has four inputs on its pins 3, 4, 5, 6 and provides a single output on pin 7, the AR6 address signal. (The signals supplied to pins 11, 12 and 13 of multiplexer 119 are for clamping purposes only.)

The $\overline{AX}$ signal is applied to the pin 14 of each of the multiplexers. The signal on this line and the signal applied to pin 2, determines which of the four inputs is coupled to each of the outputs of the multiplexers. The $\overline{AX}$ signal is a RAM timing signal for clocking the first 7 bits and second 7 bits of the multiplexed 14-bit address applied to each of the memory devices 106. The other control signal to the multiplexers is developed through the AND gate 123. The inputs to this gate are the display signal (DSPLY) which indicates that the computer is in a display mode and a clocking signal, specifically a 1 MHz timing signal ($\overline{C1M}$). The output of the AND gate 123 determines whether the address signals from the CPU or the signals associated with the counter 58 of FIG. 1 are selected.

Assume for purposes of discussion that the display has not been selected, and thus, the output of gate 123 is low. The $\overline{AX}$ signal then selects for pin 7 of multiplexer 116 first the address signal A_0 and then A_6. Likewise, each of the multiplexers selects an address signal (except for those associated with exclusive OR gates 124 and 125 which shall be discussed). If the display signal is high and an output is present from the gate 123, then, by way of example, the $\overline{AX}$ signal first causes the H_1 signal and then the V_1 signal to be connected to the AR1 address line. Similarly, signals corresponding to the vertical and horizontal count are coupled to the other address lines during display modes.

The adder 114 is an ordinary digital adder for adding two 4-bit digital nibbles and for providing a digital sum signal. A commercially available adder (Part No. 283) is employed. The carry-in terminal (pin 7) is grounded and no carry-outs occur since one of the inputs (pin 12) is grounded. The adder sums the digital signal corresponding to H_3, H_4 and H_5 with the digital signal corresponding to V_3, V_4, V_3, V_4. The resultant sum signal is coupled to the multiplexers 116, 117 and 118 as illustrated. The summing of these horizontal and vertical counter signals is used to provide the more dispersed mapping as previously discussed.

The adder 121 is identical to adder 114 and is coupled to sum the three least significant vertical counter bits from the counter 58 (FIG. 2) with the signals VA1, VB1 and VC1. The sum is selected by the multiplexer 120 during the high resolution display modes and also during scrolling as will be described. These sum signals are coupled to the multiplexers 117, 118 and 119. During the low resolution display modes, the multiplexer 120 couples ground signals or the page 2 signal ($\overline{PG2}$) to the multiplexers 117, 118 and 119. (The $\overline{PG2}$ signal is used for special mapping purposes, not pertinent to the present invention.) During the high resolution modes when the display is not being scrolled, the VA1, VB2 and VB3 signals are at ground potential and thus no summing occurs within adder 121 and the VA, VB and VC signals are coupled directly to the multiplexers 117, 118 and 119.

The address signals A_{10}, A_{11}, and A_{13} from the CPU are coupled to the multiplexers 117, 118 and 119, respectively, through exclusive OR gates 124, 125, and 126, respectively. The other input terminals to gates 124 and 125 receive the C_3 signal, while the other input terminal of the gate 126 receives the C_1 signal. (The development of the C_1 and C_3 signals is illustrated in FIG. 5.) The gates 124, 125 and 126 provide mapping compensation within the memory. As the computer and memory are presently implemented, the sequence in which the various portions of the display are generated is not the same as the sequence in which the data is removed from memory for display. These gates provide compensating addresses and, in effect, cause a remapping so that the proper sequence is maintained when data is read from the memory for the display. These gates are shown to provide a complete disclosure of the presently preferred embodiment, however, they are not critical to the present invention.

In operation, the circuitry of FIG. 4, as mentioned, selects the address signals which are applied to each of the memory devices, either from the CPU or counter if the display mode is selected. It should be noted that not all of the address bits from the CPU are coupled to the multiplexers 116 through 119. Some of these address bits, as will be described in conjunction with FIG. 5, are used to develop the various $\overline{CAS}$ and $\overline{RAS}$ signals and thus select different rows within the memory of FIG. 6.

The scrolling operation which is used is somewhat unusual in that each line of the display is separately moved up (line-by-line) with one line of data in memory being moved for each frame. This technique provides a uniform, esthetically pleasing, scroll. Scrolling the screen one line per frame can be achieved by moving all the data in the memory into a new position for each frame. This would be very time consuming and impractical. With the described technique, only one-eighth of the data in the memory is moved for each new frame.

Referring to the adder 121, as mentioned, the signals V_A, V_B and V_C are the three least significant vertical counter bits from the counter 58. These bits or counts, by way of example, represent the 8 horizontal lines of each character. In adder 12, a 3-bit digital signal, VA1, VB1 and VC1, is added to the count from counter 58. This 3-bit signal is constant during each frame, however, it is incremented for each new frame.

During a first frame, 000 is added to the vertical count. During a second frame, 001 is added; and during a third frame 010 is added, and so on. By adding this digital signal to the count from counter 58, the addresses to the memory are changed in the vertical sense. During the first frame when 000 is added, the display remains unaffected. During the next frame, when 001 is added to the vertical count, instead of first displaying the first line of a character, the second line of each character is displayed at the top of each character space and each subsequent line of the character is likewise moved up one line. If data in memory is not moved, the first line of the character would appear at the bottom of each character. Note when 001 is added to 111 from the counter, 000 results. Thus, the first line of characters would be addressed when the beam is scanning the eighth line of characters. To prevent this, the data corresponding to the first line of each character is moved in memory for this frame. The first line of one character is moved up and becomes the bottom line of the character directly above it. When 010 is added, the process is again repeated. For example, the third line of each character is first displayed in each character space and the second line of each character is moved up to become the bottom line of the character directly above it. This process is repeated to scroll the data. The movement of data in memory is controlled by the CPU in a well-known manner.

Thus, through use of adder 121, an even, continuous scroll is obtained without moving all the data in memory for each frame. Rather, only ⅛th of the data is moved for each frame.

Referring now to FIG. 5, the circuitry used to extend the addressing from the CPU is illustrated. In general, the $\overline{\mathrm{CAS}}$ signals are generated by the ROMs 127 and 128. The $\overline{\mathrm{RAS}}$ signals are generated by the ROM 132. The multiplexer 130 allows the selection of either the bank switching signals, or the unique indirect addressing mode when "bank switching" occurs without direct commands from the CPU.

The CAS ROM 127 receives as an address the following signals: PRAS , ϕ3, PRAS 1,2, $\overline{\mathrm{AY}}$, DHIRES, R/$\overline{\mathrm{W}}$, A_{11}, A_{13}, A_{14}, and A_{15}. As the PRASϕ, 3 and PRAS 1, 2 represent the RAS signals being used. These signals are high when the respective RAS signal is active. As previously mentioned, the AY signal is high for display modes and the DHIRES signal is high for high resolution display modes. The CAS ROM 128 receives as address signals the ABK1, ABK2, and ABK3 signals and also DHIRES, $\overline{\mathrm{AY}}$, IND, A_{11}, A_{13}, A_{14}, and A_{15}.

The ROMS 127 and 128 are programmed to implement the following equations.

$$\overline{\mathrm{PCAS0}} = (\mathrm{PRAS0,\ 3} \cdot (\overline{\mathrm{DHIRES}} \cdot \overline{\mathrm{AY}} + \mathrm{AY} \cdot (\overline{\mathrm{A15}} \cdot \overline{\mathrm{A14}} \cdot \overline{\mathrm{A13}} \cdot \overline{\mathrm{A11}} \cdot \overline{\mathrm{R/WN}} + \overline{\mathrm{A15}} \cdot \overline{\mathrm{A14}} \cdot \overline{\mathrm{A13}} \cdot \mathrm{R/WN} + \mathrm{A15} \cdot \overline{\mathrm{A14}} \cdot \mathrm{A13} + \mathrm{A15} \cdot \mathrm{A14} \cdot \mathrm{A13} \cdot \overline{\mathrm{A11}}))) \tag{1}$$

$$\overline{\mathrm{PCAS2}} = (\mathrm{DHIRES} \cdot \overline{\mathrm{AY}} + \mathrm{AY} \cdot (\overline{\mathrm{ABK1}} \cdot \overline{\mathrm{ABK2}} \cdot \overline{\mathrm{ABK3}} \cdot \overline{\mathrm{IND}} + \mathrm{ABK1} \cdot \mathrm{ABK2} \cdot \mathrm{ABK3}) \cdot (\overline{\mathrm{A15}} \cdot \mathrm{A14}) + \mathrm{AY} \cdot \mathrm{IND} \cdot \overline{\mathrm{ABK1}} \cdot \overline{\mathrm{ABK2}} \cdot \overline{\mathrm{ABK3}} \cdot \overline{\mathrm{A15}} \cdot (\overline{\mathrm{A14}} \cdot \mathrm{A13} + \mathrm{A14} \cdot \overline{\mathrm{A13}})) \tag{2}$$

$$\mathrm{PCAS3} = (\mathrm{PRAS0,\ 3} \cdot (\overline{\mathrm{DHIRES}} \cdot \overline{\mathrm{AY}} + \mathrm{AY} \cdot (\overline{\mathrm{A15}} \cdot \overline{\mathrm{A14}} \cdot \overline{\mathrm{A13}} \cdot \mathrm{A11} + \mathrm{A15} \cdot \mathrm{A14} \cdot \overline{\mathrm{A13}} \cdot \overline{\mathrm{A11}} + \mathrm{A15} \cdot \mathrm{A14} \cdot \overline{\mathrm{A13}}))) \tag{3}$$

$$\overline{\mathrm{PCAS4,6}} = (\mathrm{AY} \cdot \overline{\mathrm{IND}} \cdot \overline{\mathrm{ABK3}} \cdot \overline{\mathrm{A15}} \cdot (\mathrm{ABK1} \cdot \overline{\mathrm{ABK2}} + \mathrm{ABK1}) \cdot \mathrm{ABK2}) \cdot (\overline{\mathrm{A14}} \cdot \mathrm{A13} + \mathrm{A14} \cdot \overline{\mathrm{A13}}) + \mathrm{AY} \cdot \mathrm{IND} \cdot \overline{\mathrm{ABK3}} \cdot (\overline{\mathrm{ABK2}} \cdot \overline{\mathrm{ABK1}} \cdot \mathrm{A15} + \overline{\mathrm{ABK2}} \cdot \mathrm{ABK1} + \mathrm{ABK2} \cdot \overline{\mathrm{ABK1}} \cdot \overline{\mathrm{A15}}) \cdot \overline{\mathrm{A14}} + \mathrm{AY} \cdot \overline{\mathrm{IND}} \cdot \mathrm{ABK1} \cdot \mathrm{ABK2} \cdot \overline{\mathrm{ABK3}} \cdot (\overline{\mathrm{A15}} \cdot \overline{\mathrm{A14}} \cdot \mathrm{A13} + \mathrm{A15} \cdot \overline{\mathrm{A14}} \cdot \overline{\mathrm{A13}}) + \mathrm{AY} \cdot \mathrm{IND} \cdot \overline{\mathrm{ABK3}} \cdot \mathrm{ABK2} \cdot \overline{\mathrm{A15}} \cdot \mathrm{ABK1} + \mathrm{A15} \cdot \overline{\mathrm{ABK1}}\mathrm{ABK1}) \cdot (\overline{\mathrm{A14}} \cdot \overline{\mathrm{A13}} + \mathrm{A14} \cdot \overline{\mathrm{A13}})) \tag{4}$$

$$\mathrm{PCAS5,\ 7,} = (\mathrm{AY} \cdot \overline{\mathrm{IND}} \cdot \overline{\mathrm{ABK3}} \cdot (\mathrm{ABK1} \cdot \overline{\mathrm{ABK2}} + \overline{\mathrm{ABK1}} \cdot \overline{\mathrm{ABK2}}) \cdot (\overline{\mathrm{A15}} \cdot \mathrm{A14} \cdot \mathrm{A13} + \mathrm{A15} \cdot \overline{\mathrm{A14}} \cdot \overline{\mathrm{A13}}) + \mathrm{AY} \cdot \mathrm{IND} \cdot \overline{\mathrm{ABK3}} \cdot (\overline{\mathrm{ABK2}} \cdot \overline{\mathrm{ABK1}} \cdot \mathrm{A15} + \overline{\mathrm{ABK2}} \cdot \mathrm{ABK1} + \mathrm{ABK2} \cdot \overline{\mathrm{ABK1}} \cdot \overline{\mathrm{A15}}) \cdot \mathrm{A14} + \mathrm{AY} \cdot \overline{\mathrm{IND}} \cdot \mathrm{ABK1} \cdot \mathrm{ABK2} \cdot \overline{\mathrm{ABK3}} \cdot (\overline{\mathrm{A15}} \cdot \mathrm{A14}) + \mathrm{AY} \cdot \mathrm{IND} \cdot \overline{\mathrm{ABK3}} \cdot \mathrm{ABK2} \cdot (\overline{\mathrm{A15}} \cdot \mathrm{ABK1} + \mathrm{A15} \cdot \overline{\mathrm{ABK1}}) \cdot (\overline{\mathrm{A14}} \cdot \mathrm{A13} + \mathrm{A14} \cdot \overline{\mathrm{A13}})) \tag{5}$$

In effect these ROMs are programmed to allow selection of predetermined rows in the memory, based on the address signals A_{10}, A_{13}, A_{14} and A_{15} (ignoring for a moment the contribution of the $\overline{\mathrm{RAS}}$ signals and the other signals appearing in the equations).

The outputs of the $\overline{\mathrm{CAS}}$ ROMs 127 and 128 are coupled to the register 131. Register 131 is a commercially available register which permits the enabling of output signals (Part No. 374). During accessing of the memory the various $\overline{\mathrm{CAS}}$ signals ($\overline{\mathrm{CAS\ 0}}$ through $\overline{\mathrm{CAS}}$ 7) are coupled to the memory of FIG. 6 to permit selection of the appropriate memory devices. The signal USELB from $\overline{\mathrm{CAS}}$ ROM 127 through register 131 selects either the A bus or B bus. This signal is coupled to the multiplexers 43*a* and 43*b* of FIG. 3.

During normal operation, the multiplexer 130 selects the bank switching signals BCKSW 1 through BCKSW 4. These four signals (or alternatively four signals from the A bus) provide four of the inputs (address signals) to the ROM 132. The other inputs to this ROM are the DHIRES, Z PAGE, PA8, PA15, RFSH (refresh), and $\overline{\mathrm{AY}}$ signals. These address signals select the RAS 0, 3; RAS 1, 2; RAS 4, 5 and RAS 6, 7 signals. The ROM 132 is programmed to implement the following four equations.

$$\mathrm{PRAS0,\ 3} = \overline{\mathrm{AY}} \cdot (\overline{\mathrm{DHIRES}} + \mathrm{RFSH}) + (\mathrm{ABK4} \cdot (\mathrm{Z\ Page} \cdot \overline{\mathrm{PA8}})) + \mathrm{ABK1} \cdot \mathrm{ABK2} \cdot \mathrm{ABK3}) \cdot \mathrm{AY} \tag{6}$$

$$\mathrm{PRAS1,\ 2} = \overline{\mathrm{AY}} \cdot (\mathrm{DHIRES} + \mathrm{RFSH}) + \mathrm{AY} \cdot (\overline{\mathrm{ABK1}} \cdot \overline{\mathrm{ABK2}} \cdot \overline{\mathrm{ABK3}} \cdot (\mathrm{ABK4} \cdot (\mathrm{ZPAGE} \cdot \overline{\mathrm{PA8}}) \cdot \overline{\mathrm{PA15}}) + \mathrm{ABK1} \cdot \mathrm{ABK2} \cdot \mathrm{ABK3}) + \mathrm{AY} \cdot \overline{\mathrm{ABK3}} \cdot (\overline{\mathrm{ABK1}} \cdot \mathrm{ABK2} \cdot \mathrm{ABK4} \cdot (\mathrm{ZPAGE} \cdot \overline{\mathrm{PA8}}) \cdot \mathrm{PA15} + \mathrm{ABK1} \cdot \mathrm{ABK2} \cdot (\mathrm{ABK4} \cdot (\mathrm{ZPAGE} \cdot \overline{\mathrm{PA8}}) \cdot \overline{\mathrm{PA15}}) \tag{7}$$

$$\mathrm{PRAS4,\ 5} = \mathrm{RFSH} \cdot \overline{\mathrm{AY}} + \mathrm{AY} \cdot \overline{\mathrm{ABK2}} \cdot \overline{\mathrm{ABK3}} \cdot (\overline{\mathrm{ABK1}} \cdot \mathrm{ABK4} \cdot (\mathrm{ZPAGE} \cdot \overline{\mathrm{PA8}}) \cdot \tag{8}$$

-continued

$$\mathrm{PA15} + \mathrm{ABK1} \cdot (\mathrm{ABK4} \cdot (\mathrm{ZPAGE} \cdot \overline{\mathrm{PA8}}) \cdot \overline{\mathrm{PA15}})$$

$$\mathrm{PRAS6,\,7} = \mathrm{RFSH} \cdot \overline{\mathrm{AY}} + \mathrm{AY} \cdot \overline{\mathrm{ABK3}} \cdot (\mathrm{ABK1} \cdot \overline{\mathrm{ABK2}} \cdot \mathrm{ABK4} \cdot (\mathrm{ZPAGE} \cdot \overline{\mathrm{PA8}}) \cdot \mathrm{PA15} + \overline{\mathrm{ABK1}} \cdot \mathrm{ABK2} \cdot (\mathrm{ABK4} \cdot (\mathrm{ZPAGE} \cdot \overline{\mathrm{PA8}}) \cdot \overline{\mathrm{PA15}}) \quad (9)$$

Thus, the bank switching signals (along with the other input signals to ROM 132) select predetermined rows in memory in conjunction with the $\overline{\text{CAS}}$ signals.

The output signals of the ROM 132 are coupled through the NAND gates **142**, **143**, **144** and **145** to the memory. The other input terminals of these gates receive the RAS timing signal. In this manner, the output signals of the ROM 132 are clocked through the gates **142** through **145** to provide the $\overline{\text{RAS}}$ signals shown in FIGS. **5** and **6**.

An important feature to the presently described computer is provided by the circuitry shown within the dotted line **146**. The AND gate **148** receives, at its input terminals, the DA7, A_{12}, and C_3 signals. The NOR gate **149** receives the zero page and A_{15} signal. The output of gate **149** provides one input to the gate **148** and also one input to the AND gate **150**. The output of gate **148** provides another input signal to gate **150** and this signal (line **153**) is one of the two control signals coupled to the multiplexer **130**. The AND gates **150** and **151** also receive a SYNC signal and the ϕ_0 signal. The output of the gates **150** and **151** are coupled to a NOR gate **152** with the output of the gate **152** (line **154**) coupled to the other control terminal of the multiplexer **130**.

The gates **150**, **151** and **152** effectively form a clock for multiplexer/register **130** (multiplexer **130** is a commercial part, Part No. 399, which effectively is a register/multiplexer). This selects the lower four input lines to the multiplexer **130**. However, because of the synchronization signal applied to gate **151**, the multiplexer **130** selects the bank switching signals each time an OP code is fetched by the CPU.

To understand the operation of the circuit shown within the dotted line **146** it should be recalled that the memory of FIG. **6** provides a 16-bit output. As mentioned, during certain display modes, 16-bits/msec. are needed for display purposes. In nondisplay modes, only 8-bits are required, particularly for interaction with the CPU. When the memory is addressed by the CPU during the indirect addressing modes the data on the A bus is not ordinarily used. However, with the circuitry shown within the dotted line **146**, this otherwise "unused" data is put to use to provide the equivalent of the bank switching signals through multiplexer **130**.

Whenever the CPU selects a predetermined range of addresses, the multiplexer **130** selects the equivalent of the bank switching signals from the A bus provided DA7 is high. (This occurs when addressing as zero page the address space −1800 through 1FFF.) Once the signal on line **153** is high it is latched through gates **150**, **151** and **152** causing the multiplexer **130** to select the four bits from the A bus (assuming the timing signals are high). Even if the next reference from the CPU is not to this special address range, the multiplexer **130** nonetheless remains latched with the four bits from the data bus. Once the SYN pulse drops, however, which is an indication that an OP code is being fetched, the signal on line **154** rises in potential, causing the multiplexer to switch back to the bank switching signals.

Effectively, what occurs is that when the CPU selects this special address range, (and provided DA7 is high) the bits DA0 through DA3 which are stored in memory, cause a remapping, that is, the address from the CPU accesses a different part of the memory. With the fetching of each OP code, the mapping automatically returns to the bank switching signals. Importantly, the remapping, which occurs is controlled by the bits stored in the RAM (DAϕ through DA3). Thus, with the remapping information stored in RAM, toggling can occur between different portions of the memory without requiring bank switching signals, or the like from the CPU. This enhances the CPU's performance since CPU time is not used for remapping. Additionally, it provides an easy tool for programming.

For some program languages it is desirable to separate data and the program into separate portions of the memory. For example, the 128K memory can be divided into two 64K memories, one for program and one for data. Switching can occur between these memory portions without the generation of bank switching signals by the CPU with the above described circuit. This arrangement is particularly useful when using the Pascal program language.

DISPLAY SUBSYSTEM

The display subsystem **48** of FIG. **1** receives data from the A bus and B bus and converts the data into video signals which may be used for displaying alphanumeric characters or other images on a standard raster scanned cathode ray tube display. The display subsystem **48** specifically generates on line **197**, a standard NTSC color video signal and a video black and white video signal on line **198** (FIG. **8**). This display subsystem, in addition to other inputs, receives a synchronization signal, and several clocking signals. For sake of simplicity, the standard color reference signal of 3.579545 MHz is shown as C3.5M. Twice this frequency and four times this frequency are shown as C7 M and C14M, respectively.

Before describing the details of the display subsystem **48**, a discussion of a prior art display system will be helpful in understanding the present display subsystem. In U.S. Pat. No. 4,136,359, a video display system is described which is implemented in a commercially available computer, Apple-II, sold by Apple Computer, Inc., of Cupertino, Calif. In this system, 4-bit digital words are shifted in parallel into a shift register These words are then circulated in the shift register at 14 MHz to define a waveform having components at 3.5 MHz. Referring to FIG. **9**, line **206**, assume that the digital word 0001 is placed in the shift register and circulated at a rate of 14 MHz. The resultant signal which has a component of 3.5 MHz is shown on line **206**. The phase relationship of this component to the 3.5 MHz reference signal determines the color of the resultant video signal. This relationship is changed by changing the 4-bit word placed in the shift register. As explained in the above-referenced patent, if the signal 1000 is placed in the register and circulated, the resultant phase relationship of the 3.5 MHz component results in the color brown, this signal is shown on line **208**. With this prior art technique, the luminance was determined by the DC component of the signals such as shown on lines **206** and **208**.

The display subsystem 48 of FIG. 1 also uses 4-bit words to generate the various color signals in a manner somewhat similar to the above-described system. Referring to FIG. 8 4-bit words representative of colors (16 possible colors) are coupled to the bus 180 (The generation of these words shall be described in detail in conjunction with FIG. 7.) Instead of using a shift register which circulates the 4-bit word, the same result is achieved by using a multiplexer 205 which sequentially selects each of the lines of the bus 180. The signals on bus 180 also provide a luminance signal and a black and white video signal with a gray scale.

The 4 lines of the bus 180 are coupled to multiplexer 205; this multiplexer also receives the C7M and the $\overline{C3M}$/ timing signals (again, Commercial Part No. 135 is used with the pin connections shown in FIG. 8). These two timing signals cause each of the four lines to be sequentially selected and coupled to line 191. (Note that the order in which each of the lines of the bus 180 is selected does not change.)

In effect, the multiplexer operates to serialize the parallel signal from bus 180. Assume for sake of explanation that the digital signals on bus 180 are 1000 as indicated in FIG. 8. The signal on line 191 will then be 10001000 . . . . The output of the multiplexer 205 coupled to the input of the inverter 204 also receives in a sequential order, the signals from bus 180, however, in a different order. For the example shown, the input to inverter 204 is 00100010 . . . . After inversion, this results in the signal 11011101 . . . on line 192. Effectively, the signals on lines 191 and 192 are added by resistors 199 and 200. The resultant waveform is an AC signal (no DC component) shown in FIG. 9 on line 209. Thus, with the described circuit, a chroma signal is generated, having a predetermined phase relationship to the 3.5 MHz color reference signal. This phase relationship which is varied by changing the signals on bus 180 determines the color of the video signal on line 197.

In the prior art display discussed above, the DC component of the color signal determines the luminance. In the present invention, the signals on bus 180 are coupled to the base of transistor 195, consists of an AC signal from resistors 199 and 200, and the luminance level also determined by the signals on bus 180. These inputs to transistor 195, along with the $\overline{C3.5M}$ signal, generate a NTSC color signal on line 197 of improved quality when compared to the discussed prior art system.

In some cases, the signals on bus 180 are all binary ones or all binary zeros. When this occurs, there is no AC component from resistors 199 and 200 (no color signal) and the resultant signal on line 197 is either "black" or "white".

The lines of bus 180 are also coupled through resistors to the base of a transistor 196. Each of these resistors have a different value to provide a "weighting" to the binary signal. This weighting is used for non-color displays to provide "gray" shades as opposed to having a display with only black and white. The binary signals on bus 180 drive the transistor 196 to provide a video signal on line 198. RGB is generated with weighted sums of these same five signals.

Referring now to FIG. 7, data from memory is coupled from the A bus and B bus to registers 159 and 158, respectively. These registers are clocked by the 1 MHz clocking signal and its complement, thus permitting the sequential transfer of 8-bit words every 0.5 msec. As will be described, in some display modes the data is

transferred at the 2 MHz rate, and in other display modes, at a 1 MHz rate.

The registers 158 and 159 are coupled to an 8 line display bus 160. This display bus transfers data to registers 164 and 173, and also addresses to a memory 162. The registers 164 and 173 and memory 162 are enabled during specific display modes as will be apparent.

The character memory 162, in the presently preferred embodiment, is a random-access memory which stores patterns representative of alpha-numeric characters. Each time the computer is powered up, the character information is transferred from the ROM 50 into the character memory 162 during an initialization period. During character display modes, the signals from the display bus 160 are addresses, identifying particular alpha-numeric characters stored within the character memory 160. The vertical counter signals V_A, V_B, and V_C (previously discussed in conjunction with adder 121 of FIG. 4) identify the particular line in each character which is to be displayed. Thus, the generation of the digital signals representative of each of the characters occurs in an ordinary manner. The 7-bit signal representative of each line of each character (memory output) is coupled to the shift register 167. Through timing signals not shown, either the register 164 or the character memory 162 is selected to allow the shift register 167 to receive either data directly from the A bus or B bus, or alpha-numeric character information from the memory 162.

The 7-bits of information from either memory 162 or register 164 are serialized by the shift register 167 either at a 7 MHz rate or 14 MHz rate, depending upon the display mode. The serialized data is coupled by line 185 to the multiplexer 169, pins 1 and 4. The inverse of this data is also coupled to multiplexer 169, pin 3. Line 185 is also coupled as one input to the multiplexer 166 and to the register 170 (input 1).

The output 1 of register 170 (line 186) is coupled to the multiplexer 169, pin 1; to register 170 (input 2); and to multiplexer 166. Output 2 of register 170 (line 187) is coupled to input 3 of register 170 and also to multiplexer 166. Output 3 of register 170 (line 187) provides a third input to the multiplexer 166. Input 4 of the register 170 receives the output of the multiplexer 169 (line 189). Output 4 of register 120 (line 190) provides one control signal for the multiplexer 171.

The multiplexer 171 selects either the four lines of bus 183 or the four lines of bus 184. The output of multiplexer 171, bus 180, provides the 4-bit signal discussed in conjunction with FIG. 8. During one of the high resolution display modes (AHIRES), the multiplexer 171 is controlled by a timing signal from the output of the gate 178.

The multiplexer 166 selects either the lines of bus 181 or bus 182. The output of this multiplexer provides the signals for the bus 184. In all but the AHIRES display mode, multiplexer 166 selects bus 181. Thus, typically, the multiplexer 171 receives the signals from bus 174.

For purposes of description above, and also for purposes of explaining for some of the display modes below a simplifying assumption has been made. The signals coupled to the bus 180 by multiplexer 171, for most modes, are controlled by the serialized signal on line 190. This serialized signal is in sychronization with the C7M or C14M clocking signals. The multiplexer 205 of FIG. 8, which as described above, does the "spinning" for the parallel digital signal on bus 180, operates in sychronization with the multiplexer 171. In the description above, and except when otherwise noted below, it is assumed that, by way of example, if the multiplexer 171 is coupling all binary ones and zeros onto bus 180, the signal on line 191 will be either ones or zeros. Also for this condition the signal on line 192 will be all binary zeros or ones, and thus, no AC signal is generated at the base of transistor 195. However, as actually implemented, there is a "phase" difference between the clocking of the multiplexer 171 when compared to the sampling of the signals from bus 180 by the multiplexer 205. This results in a first constant AC signal on the gate of transistor 195 even when it appears that all binary ones are on bus 180, and a second constant AC signal when all binary zeros are on the bus 180. Thus, in this specification, when it states that "black" or "white" signals are being generated, instead, as currently implemented, two constant colors are generated on a color display. Where a true black and white is desired, color suppression is introduced such as through the color burst signal.

The circuit of FIG. 7, along with the circuit of FIG. 8, provides the capability for several distinct display modes. The first of these modes provides a display consisting of 40 characters (or spaces) per horizontal line. This requires a data rate of 8-bits/MHz or half the data rate the memory is capable of delivering. In this mode, data is loaded from the A bus during every other 0.5μ sec period. (B bus is not used during this mode.) This data addresses the character memory 162, and along with the signals V_A, V_B and V_C, provides the appropriate character line (7-bits) to the shift register 167. During this mode, registers 164 and 173 are disabled. The shift register 167 for this mode shifts the data at a data rate of 7 MHz (note $\overline{CH80}$ is high, allowing the 7 MHz signal from gate 175 to control the shift register 167). Each 7-bit signal is shifted serially onto line 185 and then to line 189 since multiplexer 169 selects pin 4. The data is shifted through the register 170 onto line 190. The serial binary signal on line 190 causes the selection of buses 183 or 184

The four lines of bus 183 during this mode are coupled to +V (register 173 is disabled); therefore the selection of bus 184 provides four binary ones. The selection of bus 184 provides four binary zeros through bus 181. Thus, the serial binary signal on line 190 provides either all binary ones or all binary zeros to bus 180. As discussed the circuit of FIG. 8 will provide a black and white display with 40 characters per line.

If the inverse and flashing timing means 172 is selected, each time the shift register 167 is loaded, multiplexer 169 shifts between pins 3 and 4. This causes the characters to change from white characters on a black background to black characters on a white background, and so on.

During the 80 character per line display mode, the registers 158 and 159 are each loaded during sequential 0.5μ sec periods (this utilizes the 2 MHz cycle rate previously discussed). The shift register 167 shifts the character data from memory 162 at a 14 MHz rate. The serialized data at the 14 MHz rate is shifted through the register 170 and again controls the multiplexer 171 as previously described. (Note that register 170 is always clocked at the 14 MHz rate.) Flashing again can be obtained as previously discussed.

In another alpha-numeric character display mode, the background of each character may be in one color and the character itself (foreground) in another color. This mode provides 40 characters per line. The character identification (address for RAM 162), is furnished on the A bus to register 159 at a frequency of 1 MHz. The color information (background color and foreground color) is furnished on the B bus as two 4-bit words to register 158. In the manner previously described, the address from register 159 selects the appropriate character from memory 162 and provides this information to shift register 167. The color information from the B bus is transferred to register 173. For purposes of explanation, assume that the 4-bits identifying the color red for the background are on bus 184 (from register 173 and multiplexer 166) and that 4-bits representing the color blue for the foreground are on bus 183. (Note that when register 173 is enabled, the signals from the register override the binary ones and zeros which otherwise appear on the lines of bus 174.) The serial binary signal representative of the character itself on line 190, selects either the color blue from bus 183 for the character itself or the color red from bus 184 for the background. The digital signals representative of these colors are transferred to bus 180 and provide the color data to the circuit of FIG. 8. For black and white displays, a "gray" scale is provided through the weighting circuit associated with transistor 196 of FIG. 8. Again, the multiplexer 169 may, through the timing means 172, alternate between the signal of line 185 and its inverse, which will have the effect of interchanging the foreground and background colors.

During the high resolution graphics modes, the character memory 162 is not used, but rather, data from the memory directly provides pattern information for display. This requires more mapping of data from within the main memory since new data is required for each line of the display. (Note that when characters are displayed, the character memory 162 provides the different signals required for the 8 lines of each character row. During these high resolution modes, the register 164 is enabled and the character memory 162 is disabled. Thus, the data from the A bus and B bus is shifted into the shift register 167. In these modes, the "HRES" signal to multiplexer 169 causes this multiplexer to select between pins 1 and 2. Pin 2 provides the signal directly from the shift register 167 while the signal on pin 1 is effectively the signal on line 185 delayed by one period of the C14M signal. This delay occurs through the register 170 from input 2 to output 2 since register 170 is clocked at C14M.

During a first graphics mode, data from the display bus 160 is loaded into shift register 167 at the rate of 7-bits/MHz. The data is serialized on line 185 and in the manner previously described for displaying characters, controls the selection of all binary ones and all binary zeros through the multiplexer 171. Note, as mentioned before, in the presently preferred embodiment, unless color suppression is used, this will not result in a black and white display, but rather a two-color display. If a high bit is present on line 140 of the display bus, the inverse and flashing timing means 172 causes the multiplexer 169 to alternate between pins 1 and 2. This switching occurs at a 1 MHz rate and provides a phase shift for every other 7-bits of data coupled to the multiplexer 171 on line 190. This results in an additional color being generated on the display for every other 7-bits of data.

For the above-described graphics modes when shift register 161 is shifting at a 7 MHz rate, 8-bits may be coupled to the bus 160 during each period. Specifically, as in the case of the differing background and foreground colors for the 40 character per line display mode, two 4-bit color words are shifted into register 173 at a rate of 1 MHz. Then, the multiplexer 171 selects between two predetermined colors on buses 183 and 184. Note these colors can be changed at a 1 MHz rate.

In an additional color mode identified as "AHIRES", multiplexer 171 operates under the control of gates 176, 177 and 178. In effect, multiplexer 171 selects bus 184 and latches the signals on this bus every four cycles of the C14M clock. Data is shifted into the shift register 167 from the A bus and B bus every 0.5μ sec the register 167 operates under the control of the C14M signal. Each data bit on line 185 is shifted first to line 186, then to line 187 and finally to line 188. These lines are coupled to the multiplexer 171 through multiplexer 166 which selects bus 182 since AHIRES is high. In effect, what occurs is that 4-bit color words are serialized onto line 185 and then brought back into parallel on bus 182. Since multiplexer 171 latches the signals on bus 184 every four cycles of the C14M signal, a new color word is generated at a 3.5 MHz rate on the bus 180. The resultant display is 140 by 192 colored blocks wherein each block can be any one of 16 colors.

In the last display mode, typically used with color suppression, data is shifted into the shift register 167 from the display bus at the rate of 14-bits/MHz. The data is serialized onto line 185 and controls the selection of either all binary ones or all zeros through multiplexer 171. This provides the highest resolution graphics display for the system.

Thus, a microcomputer with video display capability has been described. The computer is fabricated from commercially available parts and provides high utilization of these parts. Numerous existing programs including many of those which operate on the Apple-II computer, may be employed in the above-described computer.

I claim:

1. In a digitally controlled, raster scanned, video display for use with a microcomputer, or the like, which display provides color images in response to chroma signals having predetermined phase relationships to a reference signal of frequency (f), a circuit for providing a digitally controlled chroma signal comprising:

digital word generation means for generating predetermined digital signals;

serializing means coupled to said generation means for repeating said word in a serial form at a predetermined frequency so as to provide frequency components at said frequency f;

converting means, coupled to said serializing means for converting outputs from said serializing means to an AC signal;

whereby a video chroma signal is generated.

2. The circuit defined by claim 1 including additional circuit means coupled to said digital word generation means for providing a DC luminance signal.

3. The circuit defined by claim 1 wherein said digital words are coupled to a resistive weighting network for providing a gray scale video signal.

4. The circuit defined by claim 1 wherein said digital words are 4-bit words and wherein said predetermined frequency is equal to 4f.

5. The circuit defined by claim 4 wherein said serializing means comprises a multiplexer which is controlled in sychronization with said frequency f.

6. The circuit defined by claim 5 wherein said converting means includes an inverter coupled to an output of said multiplexer.

7. The circuit defined by claim 6 including additional circuit means coupled to said digital word generation means for providing a DC luminance signal.

8. The circuit defined by claim 1 wherein said digital word generation means comprises:

a source of digital data for controlling said display;

a first register coupled to receive data from said source of data;

a multiplexer for selecting between two buses, the output of said multiplexer coupled to said serializing means, said buses coupled to said first register,

a shift register coupled to receive data from said source of data, said shift register providing a serialized digital signal for controlling said multiplexer.

9. The circuit defined by claim 8 including a character memory for storing data representative of alpha numeric characters, said memory coupled to receive address from said source of data, the output of said memory coupled to said shift register.

10. The circuit defined by claim 9 wherein when said first register is disabled, one of said two buses is clamped to provide all binary ones, and the other of said buses provides all binary zeros.

11. The circuit defined by claim 10 wherein said shift register is controlled by a plurality of clocking signals, all of which are synchronized with said frequency f.

* * * * *

FINIS

United States Patent [19]

Atkinson

[11] **Patent Number: 4,622,545**

[45] **Date of Patent: Nov. 11, 1986**

[54] **METHOD AND APPARATUS FOR IMAGE COMPRESSION AND MANIPULATION**

[75] Inventor: **William D. Atkinson**, Los Gatos, Calif.

[73] Assignee: **Apple Computer, Inc.,** Cupertino, Calif.

[21] Appl. No.: **428,635**

[22] Filed: **Sep. 30, 1982**

[51] **Int. Cl.⁴** **G09G 1/00**

[52] **U.S. Cl.** **340/747**; 340/703; 340/748; 340/734; 358/183; 358/261

[58] **Field of Search** 340/703, 750, 798, 799, 340/801, 803, 748, 734, 747, 723; 358/183, 22, 261

[56] **References Cited**

U.S. PATENT DOCUMENTS

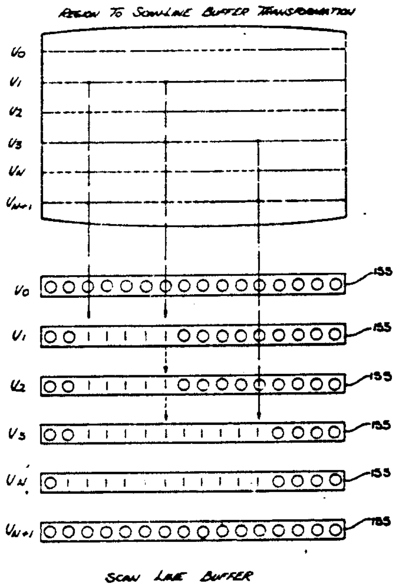
3,305,841	2/1967	Schwartz	340/748
4,233,601	11/1980	Hankins et al.	340/703
4,266,242	5/1981	McCoy	358/183
4,334,245	6/1982	Michael	358/183
4,360,831	11/1982	Kellar	358/183
4,399,467	8/1983	Subramaniam	358/261
4,420,770	12/1983	Rahman	358/183

Primary Examiner—Marshall M. Curtis
Attorney, Agent, or Firm—Blakely Sokoloff Taylor & Zafman

[57] **ABSTRACT**

Apparatus and methods are disclosed which are most advantageously used in conjunction with a digital computer to provide improved graphics capability. These techniques permit the representation and manipulation of any arbitrarily shaped image in terms of "inversion points". Inversion points defining a region are sorted and stored such that the region shape may be regenerated at a later time from the inversion points. Means are provided to compare existing regions and new regions to be displayed, and region operators are provided to specify a precedence between the existing and new regions. Thus, new regions are appropriately "clipped" such that only portions of a new region may actually be displayed to achieve the desired graphic representation.

35 Claims, 20 Drawing Figures



Lisa QuickDraw "Regions"

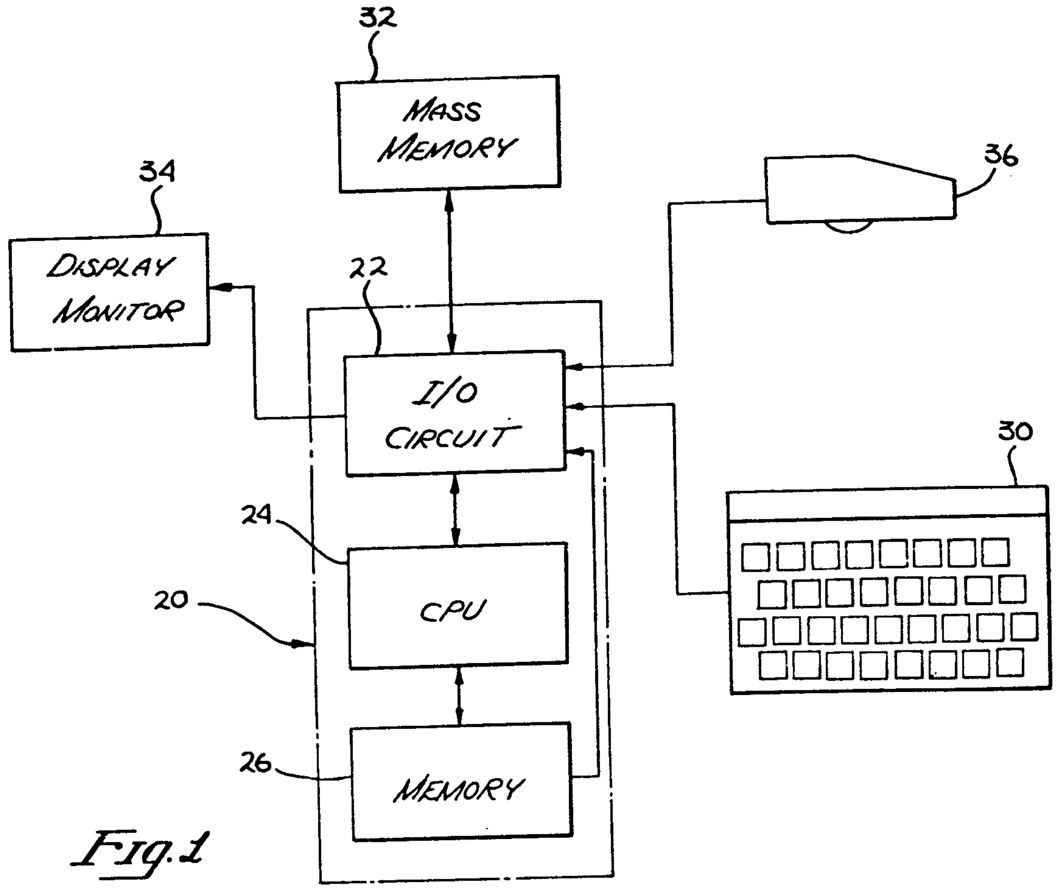

32
MASS MEMORY
34
DISPLAY MONITOR
22
36
I/O CIRCUIT
30
24
CPU
20
26
MEMORY

Fig. 1

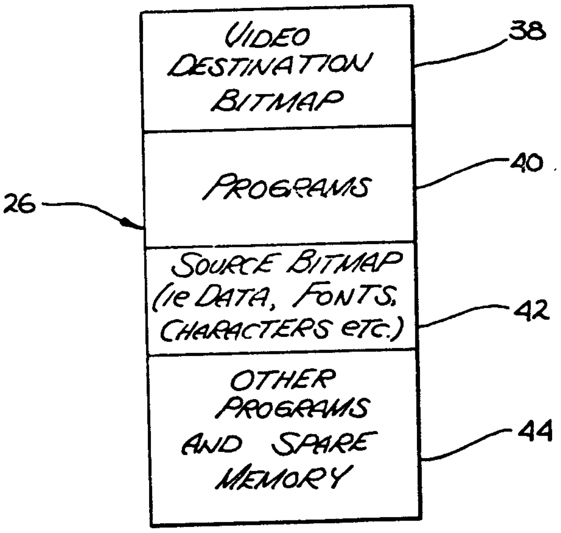

VIDEO DESTINATION BITMAP
38
PROGRAMS
40
26
SOURCE BITMAP (ie DATA, FONTS, CHARACTERS etc.)
42
OTHER PROGRAMS AND SPARE MEMORY
44

Fig. 2

Fig. 3

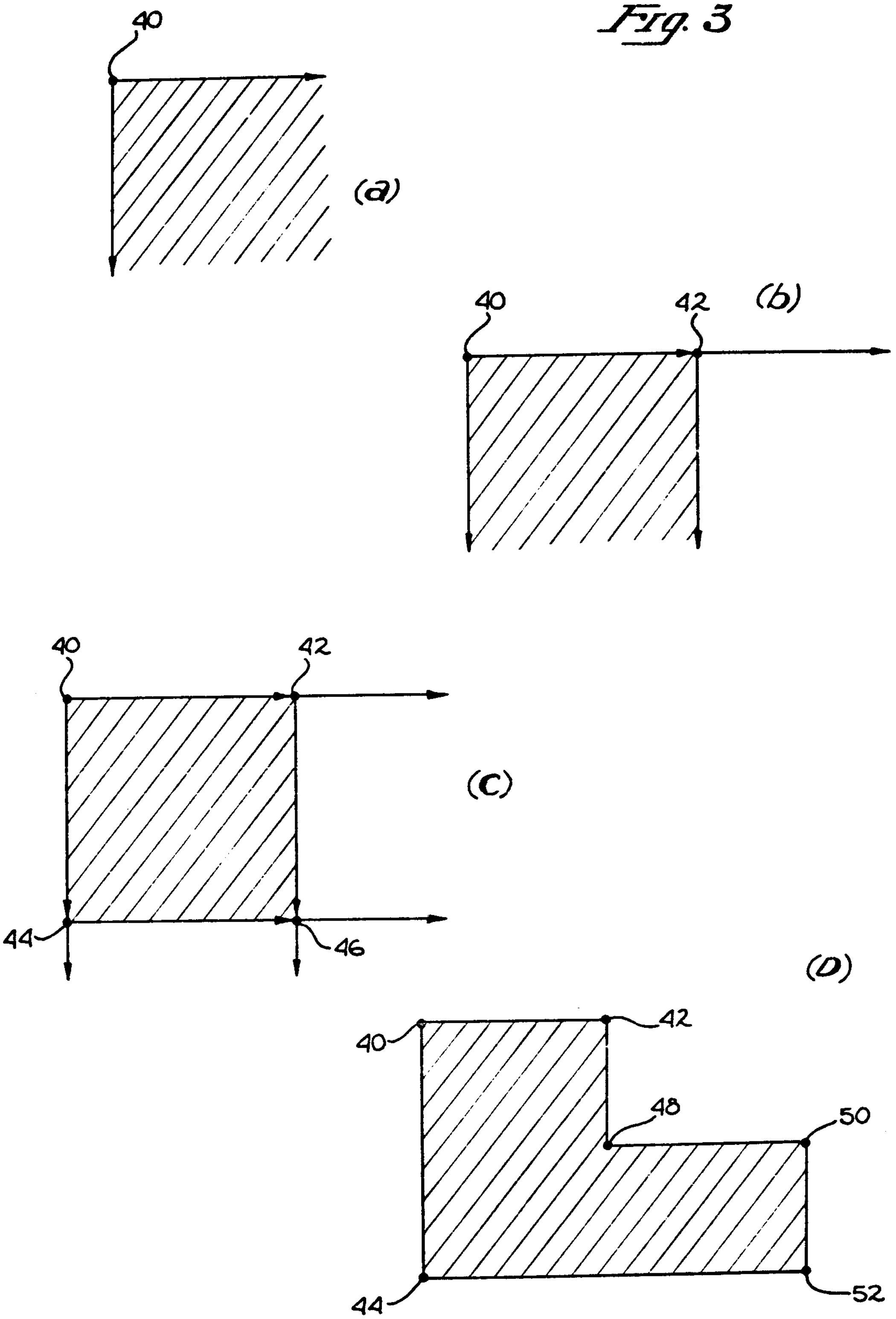


Fig. 3

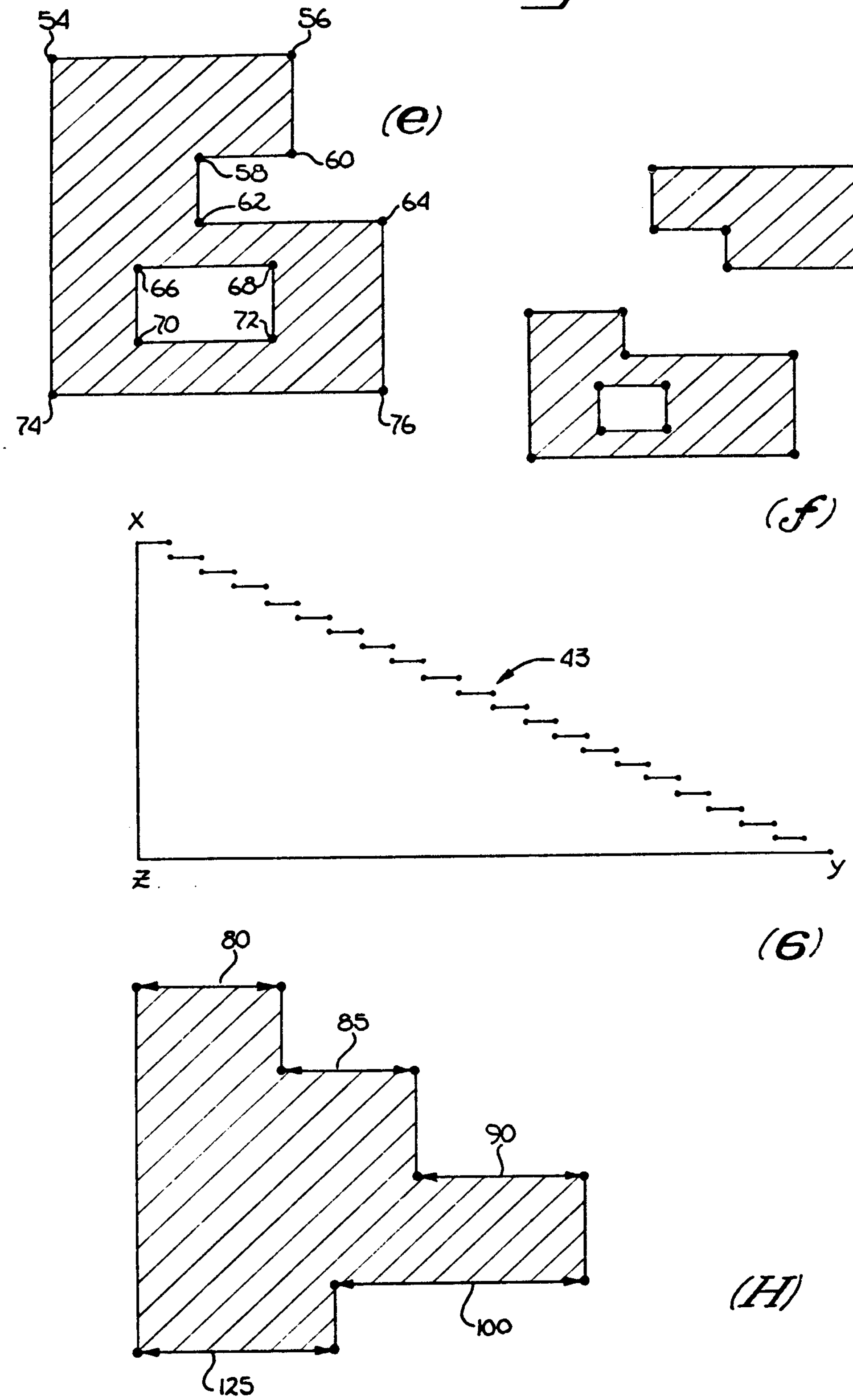
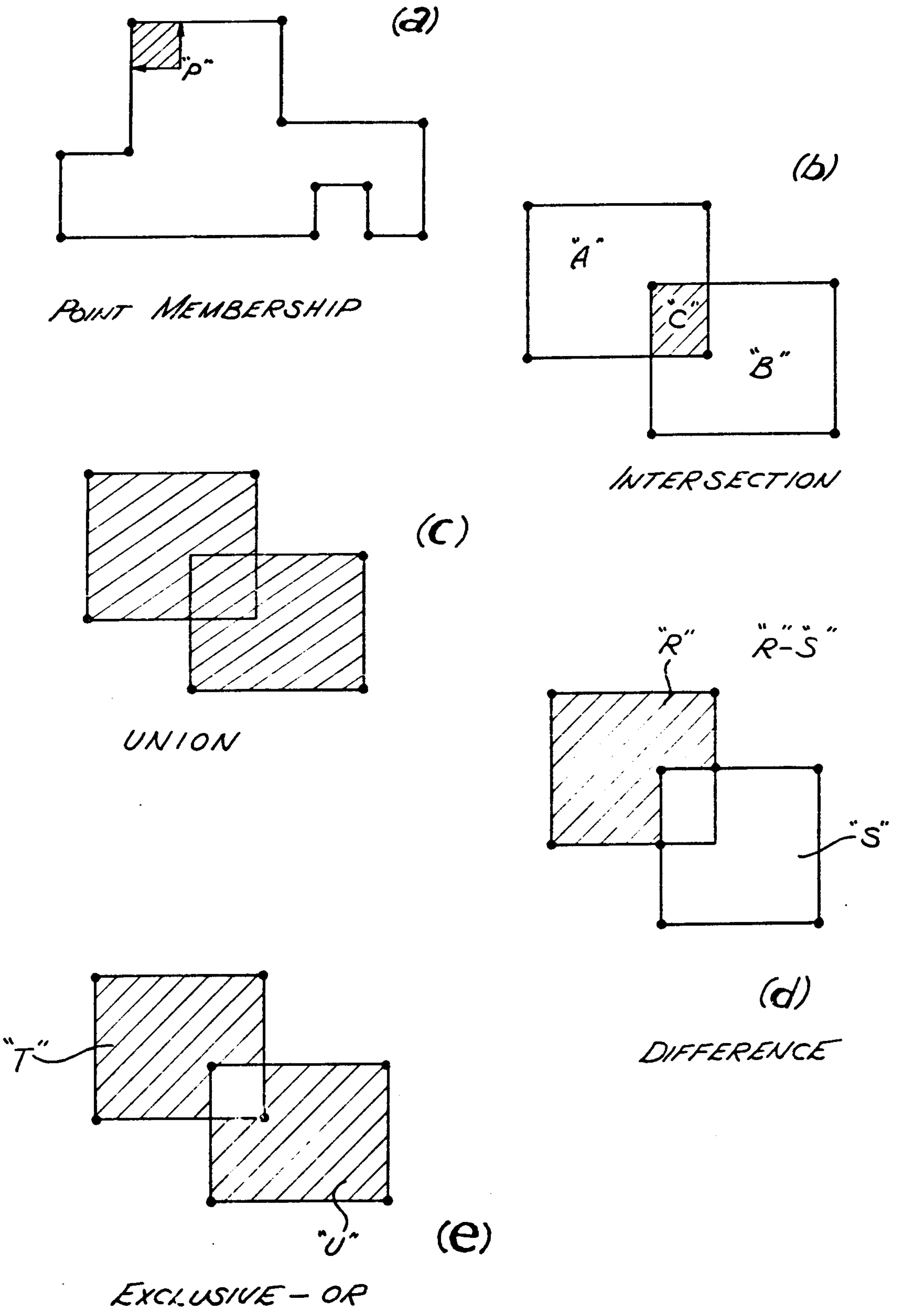
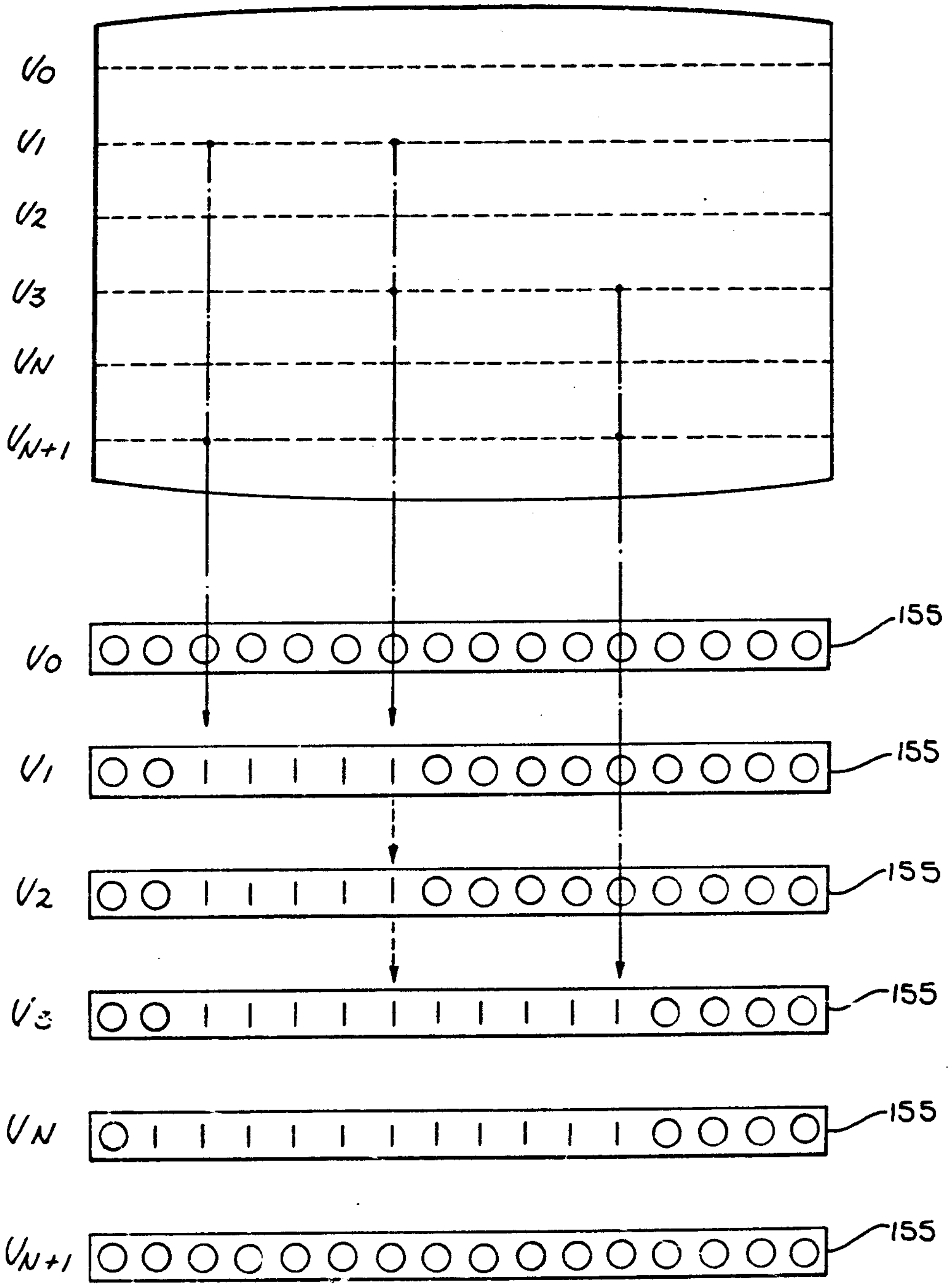


Fig. 4


(a)
"P"
POINT MEMBERSHIP
(b)
"A"
"C"
"B"
INTERSECTION
(c)
UNION
"R"
"R"-"S"
"S"
(d)
DIFFERENCE
"T"
"U"
(e)
EXCLUSIVE — OR

REGION TO SCAN-LINE BUFFER TRANSFORMATION



SCAN LINE BUFFER

Fig. 5

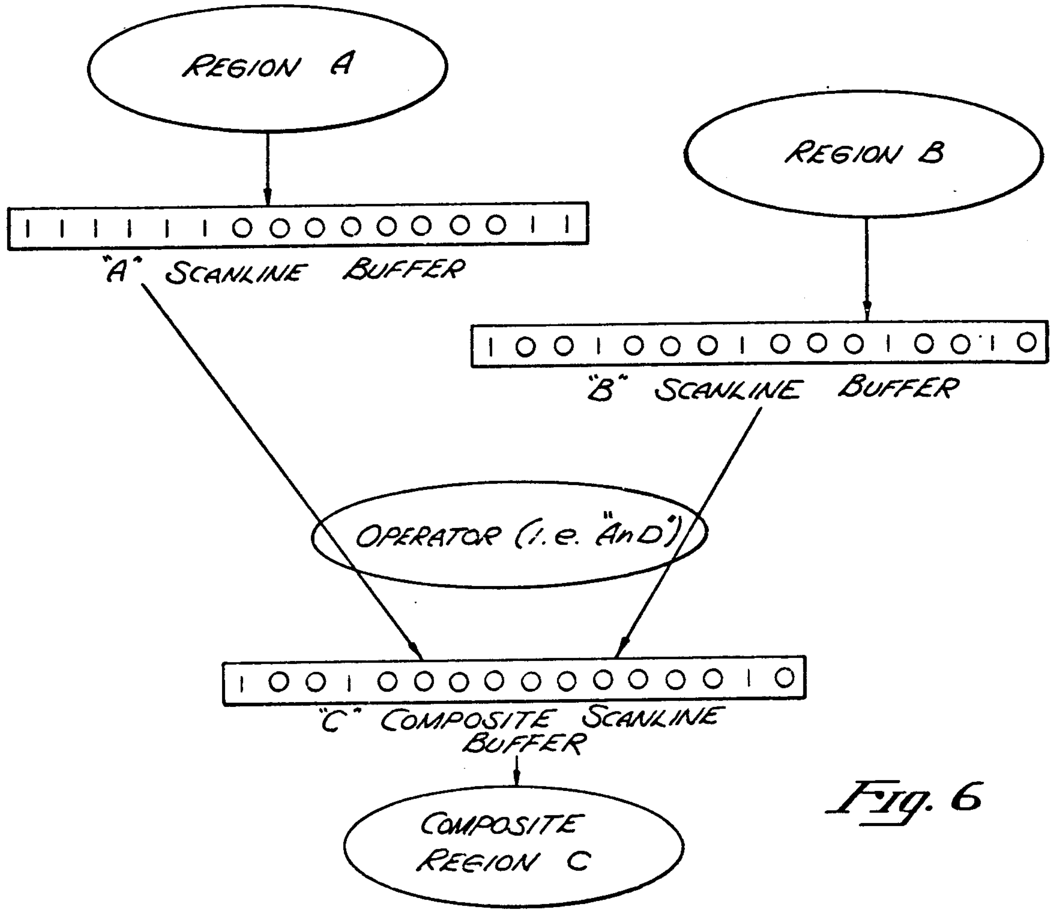


Fig. 6

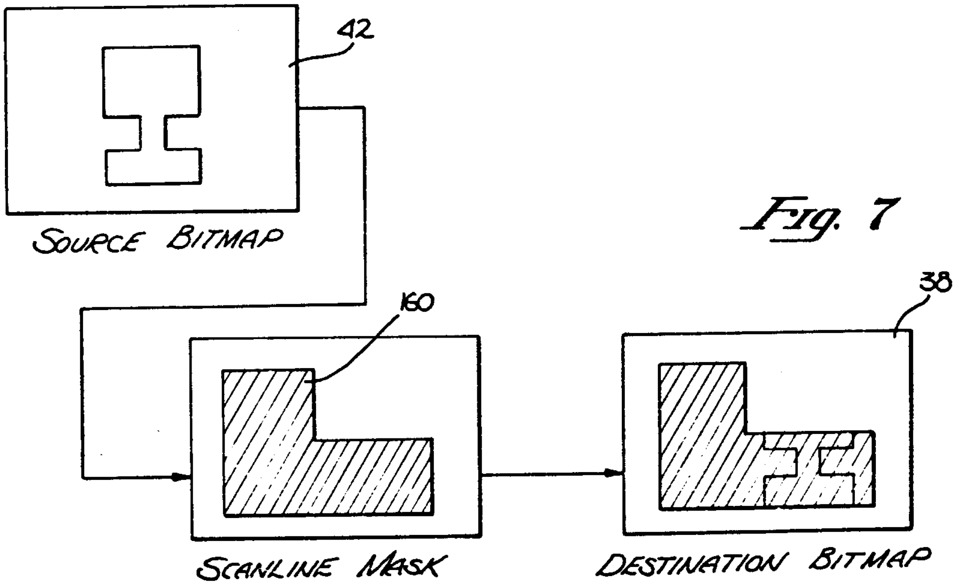


Fig. 7

"APPLE_PAT_4_622_545_07" 101 KB 2000-02-21 dpi: 300h x 300v pix: 1839h x 2921v

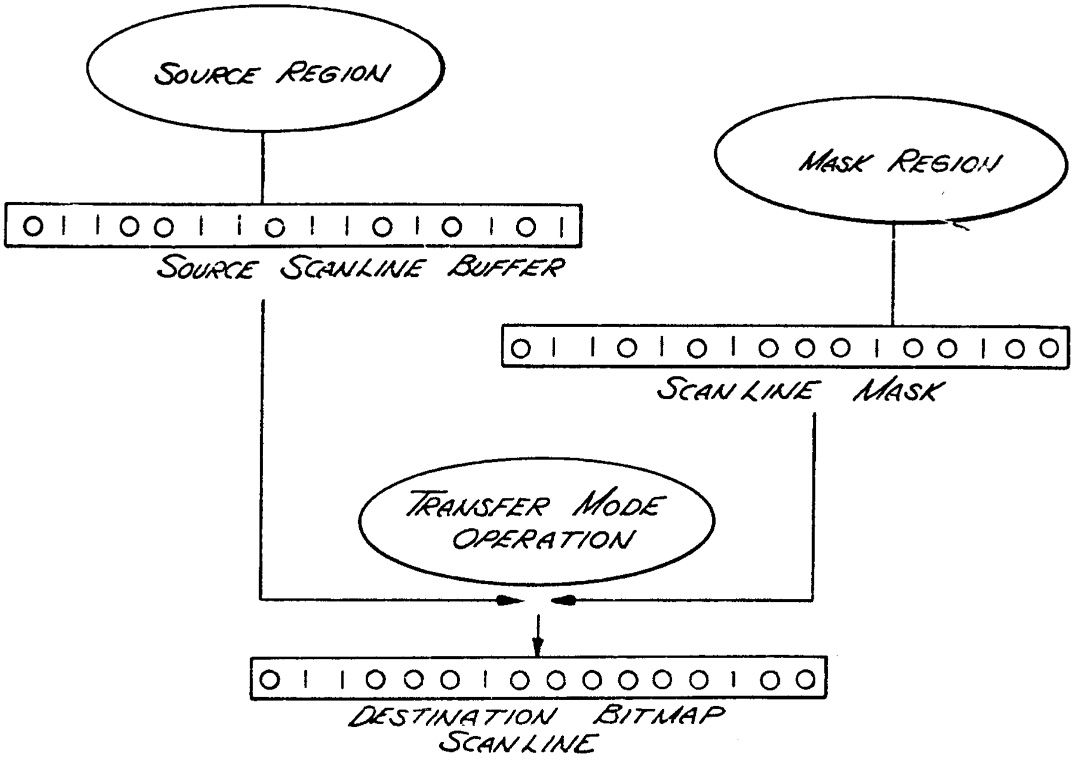

SOURCE REGION
0 1 1 0 0 1 1 0 1 1 0 1 0 1 0 1
SOURCE SCANLINE BUFFER
MASK REGION
0 1 1 0 1 0 1 0 0 0 1 0 0 1 0 0
SCANLINE MASK
TRANSFER MODE OPERATION
0 1 1 0 0 0 1 0 0 0 0 0 0 1 0 0
DESTINATION BITMAP SCANLINE

Fig. 8

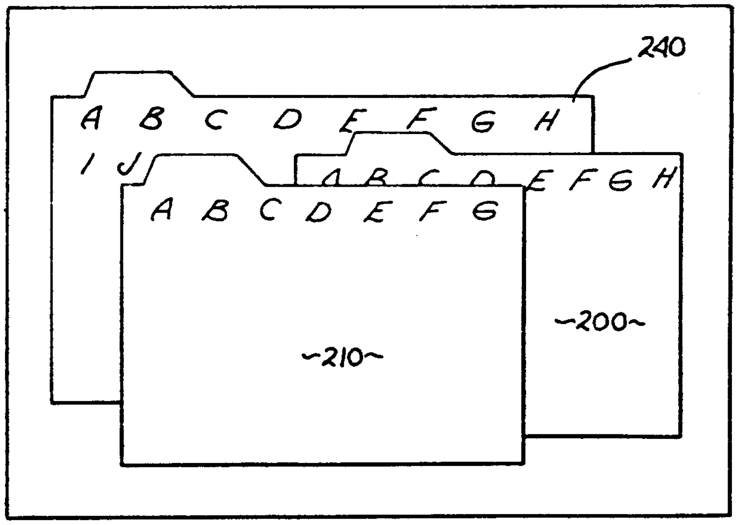

240
A B C D E F G H
I J
A B C D E F G H
A B C D E F G
~200~
~210~

Fig. 9

METHOD AND APPARATUS FOR IMAGE COMPRESSION AND MANIPULATION

BACKGROUND OF THE INVENTION

1. Field

The present invention relates tc apparatus and methods for displaying graphic information. More particularly, the present invention relates to data processing apparatus and methods for generating and manipulating images and data on a display system.

2. Prior Art

In the computing industry, it is quite common to represent and convey information to a user through graphic representations. These representations may take a variety of forms, such as for example alphanumeric characters, cartesian or other coordinante graphs, as well as shapes of well known physical objects, etc. Historically, humans have interfaced with computers through a system of discrete commands which typically comprise a combination of both text and mathematical symbolic characters. Examples of such systems are numerous and include the programming languages of Fortran, Algol, PL/I, Basic, and Cobol, which transform a given set of user commands into machine executable "object" code.

However, the ease with which a user becomes proficient in programming or interacting with a computer based system is generally a function of how close the system models the logical thought of the user himself. If the user is able to enter commands in the order in which he would find most logically appropriate, rather than having to transpose his desired command into the code of a programming language, greater user effeciency in using the system is achieved.

One system which has been developed to minimize the learning and acclamation period which a user must go through in order to become proficient in the interaction with a computer system is frequently referred to as an "object-oriented" or "Smalltalk" system. The Smalltalk approach is to replace many common coded programming commands with two-dimensional graphics and animation on a computer display. Quantitatively, it has been found that since people readily think in terms of images, a person can absorb and manipulate information presented in a visual context much faster than if represented by text. The particular type of graphic interface by which the user interacts with the machine may vary for any given application.

One common Smalltalk interface approach utilizes multiple "windows" displayed on a cathode ray tube (CRT) in which combinations of text and graphics are used to convey information. For example, each window may take the form of a file folder, of the type used in a standard filing cabinet, overlapping other folders, with the "top" fully visible folder constituting the current workfile. A user may add or delete information from a file, refile the file folder in another location, and generally operate on the file just as if an actual file in an office was being used. Thus, by graphically presenting an image which represents the object of the user's command, and allowing the user to operate on and manipulate the image in substantially the same way he would as if the image constituted the actual object, the machine becomes easier to operate to the user and a stronger man-machine interface is achieved. See, for example, D. Robson, "Object-Oriented Software Systems", *BYTE,* August 1981, Page 74, Vol. 6, No. 8; and L. Tesler,

"The Smalltalk Environment", *BYTE,* August 1981, page 90, Vol. 6, No. 8.

Although a variety of graphic representations are desired in Smalltalk or other systems, traditionally large amounts of memory have been required in order generate, store and manipulate graphics characters. In its simplest form, a block of memory may be allocated in a data processing storage system with each memory bit (a 1 or 0) mapped onto a corresponding picture element (pixel) on the display system. Thus, an entire CRT screen full of data, in the form of images and/or text, is represented as either a 1 (black dot) or a 0 (white dot) in a block of memory known as a "bitmap". However, the use of a one-to-one correspondance between the bitmap and the CRT display requires a significant amount of storage space within the computer's core memory. In addition, the generation and manipulation of an image or character requires that virtually all bits in the bitmap be updated after any modification to an image or the like. This procedure is both repetitive and time consuming, and significantly hampers the practical use of interactive graphics display operating systems.

One method of providing the necessary graphic capabilities, for systems such as Smalltalk, is "BitBlt" (Bit Boundry Block Transfer) as developed by the Xerox Learning Research Group, Palo Alto Research Center, Palo Alto, Calif. See, D. Ingalls, "The Smalltalk Graphics Kernal," *BYTE,* page 168, August 1981, Vol. 6 No. 8. BitBlt utilizes regions which are themselves small bitmaps and define simple forms, such as for example an arrow head shaped form to be used as a cursor, a pattern, etc. BitBlt, as will be discussed more fully below, transfers characters from a source bitmap; such as for example a font file of characters, to a destination bitmap (i.e. a block of memory to be displayed on a CRT) at given coordinates. By incorporating the use of a "clipping rectangle" which limits the region of the destination bitmap which can be effected, a portion of a larger scene can be mapped into a window such that only that portion of the transferred scene which falls within the window will be transferred. In addition, a variety of transfer operations are provided which control the combination of a transferred scene or character with an existing scene previously stored at the destination bitmap. However, the BitBlt system is limited in terms of the types of images which can be transferred and manipulated. Specifically, BitBlt is constrained to transfers of rectangular areas. This limitation significantly restricts its use as a graphics tool since BitBlt is thereby unable to transfer data to overlapping windows or the like. In addition, large amounts of memory are required for the BitBlt system. Other limitations in prior art systems, such as BitBlt, are described in this Patent in order to more fully identify the nature of the present invention.

As will be disclosed below, the present invention provides a means whereby any arbitrarily shaped region may be defined and stored using significantly less memory than was previously possible in the prior art. Additionally, the present invention provides a means whereby operations may be performed on regions efficiently and quickly by a digital computer.

SUMMARY OF THE INVENTION

The present invention provides methods and apparatus which are most advantageously used in conjunction with a digital computer to provide improved graphics capability. These techniques permit the representation and manipulation of any arbitrarily defined region in terms of "Inversion Points". An inversion point is by definition a point at which the state of all points having coordinates to the right and below the subject point are inverted (e.g. binary zeros are converted to binary ones and visa versa). A "Region" is defined as any arbitrary area which may include a number of groups of disjoint areas. Thus, any shape, such as for example an "L" shape is treated simply as another region to be defined and operated on. By defining a set of inversion points for any given region, all of the points which constitute the region need not be stored in memory, rather, only the inversion points defining the region need be stored.

Briefly stated, in accordance with one typical embodiment of the present invention, there is provided means for generating an input representation of a region, which may comprise any arbitrary shape or area the perimeter of which need not be a continuous curve and may include disjoint areas. This input representation is most advantageously coupled to a digital computer. Once received, the digital computer determines the position of the inversion points needed to define the region and sorts the points left to right and top to bottom in accordance with their coordinates in the region. Algorithm means are provided to transfer and operate on regions (or portions thereof) within the computer memory and to display a resulting region on an appropriate device, such as for example a cathode ray tube (CRT) or the like.

A scan line mask comprises a one scan line buffer, which in binary form represents existing regions which are currently being displayed and stored in a destination bitmap. The destination bitmap comprises a block of memory in which each bit corresponds to a pixel or the like on the display device. The scan line mask vertically scans down and "slices" the existing regions into horizontal rows corresponding to each raster line on the CRT display. Similarly, data from a source bitmap or font file, in the form of characters or the like, to be added to a portion of the destination bitmap is also "sliced" and placed into a horizontal scan line buffer corresponding to each raster scan line of the CRT. As one horizontal scan line is transfered from the source bitmap or the like to the destination bitmap, the contents of the source scan line buffer are compared to the contents of the scan line mask, such that the source scan line is "masked" and only selected portions of the source buffer are transferred to the destination bitmap. By using a variety of region operators, precedence between existing and new regions may be specified. Thus, a pattern (such as for example striped, checked or the like) may be added to an existing region, text may be overlayed, scrolling of text within a region may be easily accomplished, and numerous other graphics operations may be completed.

The resulting destination bitmap is converted to signals which are then applied to a CRT or other display device, and the image is displayed in a conventional manner.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 illustrates a computer incorporating the present invention.

FIG. 2 shows a typical arrangement of program storage in the system of FIG. 1.

FIGS. 3(*a*)–(*h*) illustrate the use of inversion points to define a region.

FIGS. 4(*a*)–(*e*) illustrate operations on regions using inversion points which may be accomplished using the present invention.

FIG. 5 illustrates the process of converting a region defined by inversion points into a one scan line buffer scanning vertically down a region.

FIG. 6 symbolically illustrates the "AND" operation between two regions one scan line at a time.

FIG. 7 symbolically illustrates the operation of a bitmap mask to selectively mask portions of a source region to be displayed.

FIG. 8 symbolically illustrates the use of one scan line buffer and a scan line mask to selectively mask portions of a source region prior to its transfer to the destination bitmap for display.

FIG. 9 illustrates the result of one implimentation of the present invention using the inversion point scan line mask system.

NOTATION AND NOMENCLATURE

The detailed descriptions which follow are presented largely in terms of algorithms and symbolic representations of operations on data bits within a computer memory. These algorithmic descriptions and representations are the means used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art.

An algorithm is here, and generally, conceived to be a self-consistent sequence of steps leading to a desired result. These steps are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated. It proves convenient at times, principally for reasons of common usage, to refer to these signals as bits, values, elements, symbols, characters, terms, numbers, or the like. It should be borne in mind, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities.

Further, the manipulations performed are often referred to in terms, such as adding or comparing, which are commonly associated with mental operations performed by a human operator. No such capability of a human operator is necessary, or desirable in most cases, in any of the operations described herein which form part of the present invention; the operations are machine operations. Useful machines for performing the operations of the present invention include general purpose digital computers or other similar devices. In all cases there should be borne in mind the distinction between the method operations in operating a computer and the method of computation itself. The present invention relates to method steps for operating a computer in processing electrical or other (e.g., mechanical, chemical) physical signals to generate other desired physical signals.

The present invention also relates to apparatus for performing these operations. This apparatus may be specially constructed for the required purposes or it may comprise a general purpose computer as selectively activated or reconfigured by a computer program stored in the computer. The algorithms presented herein are not inherently related to any particular computer or other apparatus. In particular, various general purpose machines may be used with programs written in accordance with the teachings herein, or it may prove more convenient to construct more specialized apparatus to perform the required method steps. The required structure for a variety of these machines will appear from the description given below.

DETAILED DESCRIPTION

The following detailed description will be divided into several sections. The first of these will treat a general system arrangement for generating computer graphics. Subsequent sections will deal with such aspects of the present invention as defining an inputted region in terms of inversion points, the sorting of inversion points, operations on inversion points, generation of a scan line mask, and region transfer operations among others.

In addition, in the following description, numerous specific details are set forth such as algorithmic conventions, specific numbers of bits, etc., in order to provide a thorough understanding of the present invention. However, it will be obvious to one skilled in the art that the present invention may be practiced without these specific details. In other instances, well-known circuits and structures are not described in detail in order not to obscure the present invention unnecessarily.

GENERAL SYSTEM CONFIGURATION

FIG. 1 shows a typical computer-based system for generating computer graphic images according to the present invention. Shown there is a computer 20 which comprises three major components. The first of these is the input/output (I/O) circuit 22 which is used to communicate information in appropriately structured form to and from the other parts of computer 20. Also shown as part of computer 20 is the central processing unit (CPU) 24 and memory 26. These latter two elements are those typically found in most general purpose computers and almost all special purpose computers. In fact, the several elements contained within computer 20 are intended to be representative of this broad category of data processors. Particular examples of suitable data processors to fill the role of computer 20 included machines manufactured by the Apple Computer Co., Cupertino, Calif. Other computers having like capabilities may be of course be adapted in a straightforward manner to perform the several functions described below.

Also shown in FIG. 1 is an input device 30, shown in typical embodiment as a keyboard. It should be understood, however, that the input device may actually be a card reader, magnetic or paper tape reader, or other well-known input device (including, of course, another computer). A mass memory device 32 is coupled to the I/O circuit 22 and provides additional storage capability for the computer 20. The mass memory may include other programs, fonts for given characters, and the like and may take the form of a magnetic or paper tape reader or other well known device. It will be appreciated that the data retained within mass memory 32, may, in appropriate cases, be incorporated in standard fashion into computer 20 as part of memory 26.

In addition, a display monitor 34 is illustrated which is used to display the images being generated by the present invention. Such a display monitor may take the form of any of several well-known varities of CRT displays. A cursor control 36 is used to select command modes and edit graphics data, such as for example a particular image, and provides a more convenient means to input information into the system.

FIG. 2 shows a typical arrangement of the major programs contained within the memory 26 illustrated in FIG. 1. In particular, there is shown a video destination bitmap 38, which in the presently preferred embodiment comprises approximately 32 kilobytes of storage. This destination bitmap represents the video memory for the display monitor 34. Each bit in the destination bitmap corresponds to the upper left coordinate of a corresponding pixel on the display monitor. Thus, the destination bitmap can be described by a two-dimensional array of points having known coordinates. Of course, in the case where other display means are used, such as for example a printer or the like, the contents of the bitmap 38 would represent the data points to be displayed by the particular display device. Memory 26 also includes programs 40 which represent a variety of sequences of instructions for execution by the CPU. For example, the control program implimenting the operations and routines described in this Patent, monitor and control programs, disk operating systems and the like may be stored within this memory location.

Source bitmap 42 which may comprise regions, fonts, data structures, coordinates and characters are also stored in memory 26, or may be temporarily stored in mass memory unit 32 as may be required in any given application of the present invention. Additionally, space within memory 26 is reserved for other programs and spare memory which is designated at 44. These other programs may include a variety of useful computational or utility programs as may be desired.

INVERSION POINT REPRESENTATION OF DEFINED REGIONS

The present invention represents any arbitrarily defined region in terms of "inversion points". In addition, the present invention defines a "region" to be any arbitrary area which may include a plurality of disjoint areas of any shape or configuration. Referring now to FIG. 3(*a*), an inversion point 40 is illustrated. An inversion point is, by definition, a point at which the state of all points having coordinates to the right and below the inversion point are inverted. Thus, as depicted, all areas to the right and below the point 40 are dark since point 40 was defined on a previously white background. In terms of the physical implementation of the inversion point system, the position of an inversion point is described in terms of its coordinates in a memory bitmap.

As illustrated in FIG. 3(*b*), a vertical unbounded strip results when two inversion points, 40 and 42, are defined on a bitmap such as destination bitmap 38, and subsequently displayed on monitor 34. The addition of the point 42 on the bitmap inverts the state of all points having coordinates to its right and below it, cancelling the effect of point 40 within this area and thereby defining a darkened vertical strip.

Similarly, four inversion points 40, 42, 44 and 46 define a square or other quadrangle as shown in FIG. 3(*c*). As illustrated in FIGS. 3(*d*) and (*e*) other areas may be defined using inversion points, and voids within a given shape may be easily generated. In addition, it will be apparent that any given region may contain any number of disjoint areas, as shown in FIG. 3(*f*), inasmuch as all shapes within a region are simply defined by the coordinates of the inversion points.

Moreover, circular and other non-linear regions may be defined by proper positioning of inversion points. With reference to FIG. 3(*g*), a diagonal line 43 may be defined between points "X" and "Y" by a step series of two inversion points between "X" and "Y." Although a direct diagonal line between points would be preferred, the physical structure of the raster line display monitor 34 does not permit this. Each pixel on the CRT display occupies a unit area between given coordinates, where by convention a particular pixel is accessed by the coordinate of the grid point which lies at its top left. Thus, a step-like function of inversion points defining a series of horizontal line segments is required to approximate a diagonal line.

It will be appreciated that once any given region is defined in terms of its inversion points, in general only the inversion points need be retained in memory 26, unlike many prior art systems which require that virtually all points comprising an image be stored. In the presently preferred embodiment, a region is entered into the computer 20 by a user by means of cursor control 36 or other input device. The position of the inversion points defining the region is determined by detecting horizontal line segments which in part form portions of the imputted region. With reference to FIG. 3(*h*), line segments 80, 85, 90, 100 and 125 are thus identified. Inversion points are then defined at the coordinates corresponding to the end points of each line segment, thereby defining the entire region in terms of its inversion points. Vertical line segments within the region are ignored since they will be generated automatically, by definition, using the previously described inversion point convention. The specific sequence of operations which are required to be executed by computer 20 to detect and isolate horizontal line segments, will be apparent to those skilled in the data processing arts, and will not be set forth in this description. The inversion points of a region are sorted into an ordered list of points in a left to right, top to bottom order in accordance with their coordinates. For example, with reference to the region of FIG. 3(*e*) the list of inversion points in accordance with the convention would be as follows: 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76.

It has been found, that the use of the above convention permits simplified operations on regions such as those illustrated in FIGS. 4(*c*)-(*e*). Typical operations which may be performed using the present invention's use of ordered lists of inversion points are the functions of the determination of point membership, as well as the intersection, union, differerence, and exclusive-OR of regions.

Frequently, in the course of a graphics operation, it is necessary to determine if a point in the destination bitmap 38 (and thereby correspondingly displayed on the display monitor) lies within a particular region. This function is generally referred to as "point membership". Traditionally, the determination of point membership required rather extensive data manipulations and calculations. For example, one prior art method of determining point membership was to calculate and sum the angles from the point in question to the region of interest. If the sum of the angles equals 360 degrees then point membership within the region exists. It will be appreciated that this particular method of determining point membership requires numerous and repetitive calculations and is extremely time consuming.

However, the present invention's use of inversion points provides an efficient means to determine point membership. With reference to FIG. 4(*a*), the present invention scans through the previously ordered list of inversion points defining the region in question, from top to bottom. If an inversion point has a vertical coordinate greater then or equal to the vertical coordinate of the point in question (point "P" in FIG. 4(*a*)), and the inversion point's horizontal coordinate is less than that of point "P", a variable is "toggled" which is either true or false (and which was originally set, for example, to false). Thus, each time and inversion point above and to the left of the point in question is detected, the state of a true/false variable is switched. If, after scanning through the list of inversion points defining the region the variable is true (i.e. an odd number of state changes occurred) the point in question (i.e. point "P") lies within the particular region. However, if the variable is false (i.e. zero or an even number of state changes occurred) the point is not within the region. Thus, a quick and efficient method for determining point membership using inversion points is provided by the present invention which was not possible in the prior art.

REGION TO SCAN LINE BUFFER TRANSFORMATION

The present invention's use of ordered lists of inversion points provides a straightforward means of representing the contents of each raster scan line on monitor 34. Referring now to FIG. 5, portion of memory 40 (See FIG. 2) is allocated as a one scan line buffer. In the presently preferred embodiment, this can line buffer is sufficiently large such that each horizontal row of pixels on the CRT monitor screen or other output device is represented by a bit within the buffer. A region which has been previously defined in terms of an ordered list of inversion points may be represented by bit states within the scan line buffer. For every horizontal row displayed on monitor 34, designated V_o, V_1, V_2 . . . V_{n+1} in FIG. 5, inversion points having vertical coordinates corresponding to the particular horizontal row which is scanned are represented by an altered bit state (i.e. a 1 in an original scan line field of 0's) at appropriate coordinates on the scan line buffer. All bits between pairs of inversion points in scan line 155 are then inverted, such that a true representation of the region to be displayed is generated from the inversion point ordered list. Thus, as shown in FIG. 5, by scanning through each horizontal row to be displayed, any region may be horizontally and sequentially "sliced" into segments one scan line wide.

As will be discussed below, the use of a single raster scan line buffer allows a region to be transferred from a source bitmap 42 to the destination bitmap 38 and appropriately "masked" such that any arbitrary region may be transferred and manipulated, unlike prior art systems such as BitBlt which are confined to rectangular region transfers.

In addition, it will be appreciated that the region to scan line buffer transform is reversable. Once a region is represented in the form of a one scan line buffer, an ordered set of inversion points may be redefined by locating inversion states on the buffer as the buffer scans a region from its top (V_1) to bottom (V_{n+1}). Inversion point positions are located easily inasmuch as an inversion point position on the buffer is that point where a bit state change is sensed (i.e. a 1 where the next bit is a 0). More specifically, in the present embodiment the location of inversion points may simply be determined by an exclusive-OR operation between the current scan line (e.g., V_3) buffer contents and the previous (e.g., V_2) scan line buffer contents. Thus, the portions of regions which remain unchanged between subsequent vertical scan line positions are ignored inasmuch as a uniformity of content between one vertical scan line position and the next would indicate that no inversion points are present. In addition, horizontal positions of inversion points may then be determined by shifting the resulting exclusive-OR ed scan line to the right by 1 bit, and effectuating another exclusive-OR operation. For example, if after the exclusive-OR operation between scan line buffer V_n and V_{n-1} the result was 01110011, then by shifting the result to the right one bit and completing another exclusive-OR operation we obtain:

```
         01110011
         00111001(1)
01001010 - inversion point positions for scan line Vn
```

The specific commands to be executed by computer 20 in order to determine where in a scan line buffer a state change exists will be apparent to one skilled in the art, and will not be further described.

REGION OPERATORS

The present invention's use of a one scan line buffer to systematically represent the contents of regions permits the previously described operations of union, intersection, etc., to be easily accomplished. For example, the intersection operation illustrated in FIG. 4(*b*) provides an inversion point representation of the shaded area, and is obtained by executing an "AND" of the two overlapping regions "A" and "B." Referring now to FIG. 6, a one scan line buffer is defined for each region "A" and "B." For each horizontal raster row of the CRT display, the respective scan line buffer represents each region's contents in binary form. The contents of the scan line buffers are then operated upon in order to accomplish the desired function. In the case of FIG. 4(*b*), the contents would be "AND"ed together to result in a composite scan line. For example, if for vertical position V_1:

"A" scan line=11111100

"B" scan line=10010001

Then the composite scan line after an "AND" operation would be: 10010000. In addition, the identical "AND" operation is done for each horizontal row V_n comprising each region. The result of the above operation being a composite representation, one scan line at a time, of the resulting intersecting shaded region "C" of FIG. 4(*b*). The position of the inversion points comprising the shaded region "C" may then be extracted using known techniques, such as the exclusive-OR operation previously described.

Similarly, an "OR" operation between the two regions is utilized in order to achieve the union function of FIG. 4(*c*). To obtain the "Difference" of FIG. 4(*d*), the operation between the two regions would be (NOT "S") AND "R", wherein the state of all binary quantities represented within the "S" scan line buffer is inverted prior to "AND"ing the contents with the "R" scan line buffer.

Finally, the exclusive-OR operation of FIG. 4(*e*) is simply achieved by performing the exclusive-OR on each region's scan line buffer contents, in the same manner as was done in the above example of the "AND" operation. However, it will be apparent to one skilled in the art that the present invention's use of ordered lists of inversion points renders the exclusive-OR operation trivial. The operation may be accomplished by merge

sorting the inversion point lists of regions "T" and "U" of FIG. 4(*e*), and discarding any points having the same coordinates in both regions. In other words, computer 20 simply treats the ordered lists of inversion points defining regions "T" and "U" as one large list, and sorts all of the inversion points, left to right and top to bottom in accordance with the previously described convention. The resultant list of inversion points represents a region whose points are contained either in region "T" or "U" but not both.

It will be appreciated that numerous other operations, and combinations of operations, using the present invention's inversion point and scan line buffer method may be performed on arbitrary regions that was possible in prior art methods.

SCAN LINE MASK

With reference now to FIG. 7, the present invention's use of a scan line mask to provide arbitrary region clipping is symbolically illustrated. A previously defined region 160 which has been converted into an ordered list of inversion points is used as a "mask" to which all additional images to be displayed on the monitor 34 are compared, prior to affecting the destination bitmap 38. As shown in FIG. 9, it is frequently desired that multiple regions overlap with some predetermined precedence. As is illustrated, folders may be depicted as overlapping, text may be provided on each displayed folder, and other arbitrary regions may be displayed. However, as discussed above, prior art methods such as BitBlt are constrained to rectangular "region clipping". Thus, the versatility of prior art systems is severely limited by the constraint of operating on rectangular regions only, and their inability to selectively affect regions other than the topmost window (e.g. folder 210).

As symbolically illustrated in FIG. 7, other regions such as patterns or characters are compared to a bitmap "mask", one scan line at a time, of existing regions which are currently being displayed. As will be discussed below, by defining region operators various masking priorities may be defined. Thus, patterns may be provided as well as fonts and other characters within any arbitrary region. "Region clipping" is provided in accordance with the region operators such that portions of overlapping regions are selectively displayed.

Referring now to FIG. 8, each source bitmap 42 which may comprise an image, character, font or the like which is desired to be displayed is "sliced" and transformed into a one scan line buffer in accordance, with for example, the above discussion under the heading "Region to Scan line Buffer Transformation." Thus, any region to be displayed is represented by a one line scan buffer which horizontally scans the source bitmap 42 and provides a binary representation of the source region by proper expansion of inversion point positions along the buffer.

The regions which are presently being displayed form a bitmap "mask" region to which new regions to be displayed are compared. As is done with the new source regions to be added, the existing displayed regoin is transformed into a one scan line mask representing the contents in binary form of the destination region. Depending on the transfer mode operation specified, each scan line of the new region is selectively transferred to the destination bitmap 38 and displayed on the display monitor 24.

The specific type of transfer mode operator used is a function of the desired output. Region operators include the functions of OR, AND, exclusive-OR, NOT as well as any combination thereof. For example, if the current scan line mask for row V_1 on the CRT contains 01101010 and the current source scan line buffer for V_1, contains 01100110 then the result after an "AND" operation which would be displayed on monitor 34 would be:

```
      01101010 - scan line mask buffer contents
(AND) 01100010 - source scan line buffer contents
______________________________________________
      01100010 - destination bitmap scan line contents
                 to be displayed
```

Thus, it will be appreciated that not all portions of the new source region will be transferred to the display device, and is thereby "clipped" depending on the particular transfer operator chosen. In addition, it will be noted that the particular shape of the regions being operated upon is irrelevant to the method of the present invention. The use of inversion points and one scan line buffers allow any arbitrary region to be defined, masked and transferred by the present invention.

In the presently preferred embodiment, three separate scan line mask buffers are provided to which a new source region is compared. A "user region" mask comprises the existing region being displayed which the new region, if transferred, will affect. A "visible region" mask is defined as the visible portion of the existing region currently being displayed (e.g., folder 200 of FIG. 9). The "clipping region" comprises the visible portion of the user region to which the new source region will be "clipped", such that only a portion of the source region is transferred. Thus, a new source region to be transferred from the source bitmap 42 to the destination bitmap 38 is passed through the equivelent of three scan line mask buffers. In practice, each scan line mask is "AND" ed with one another and the composite scan line mask is then utilized to mask new regions.

With reference to FIG. 9, an example of an output displayed on monitor 34 in accordance with the present invention is illustrated. Region 200 was originally defined by a user and stored in memory 26 as an ordered list of inversion points. By specifying a proper region operator as described above, regions 210 and 240 have been displayed such that it appears that region 200 lies between regions 210 and 240. Similarly, text has been provided within each folder shaped region, and appropriate region clipping using the scan line mask method as described above insures that only those portions of each region which would be visible if actual folders were used is displayed.

Moreover, it will be apparent to one skilled in the art that although the present invention has been described with emphasis on binary representations on the display device 34, and therefore in black and white, that appropriate inversion point and scan line masking for color images may also be achieved. For example, to provide the colors of red, green and blue, three inversion point representations of a region may be utilized, one for each color respectively. Thus, the presence of an inversion point in one color region may selectively discharge a color gun in a color CRT or the like for that color. Similarly, various colors could be acheived by the appropriate combination of the three inversion point representations of each region stored in memory.

CODING DETAILS

No particular programming language has been indicated for carrying out the various procedures described above. This is in part due to the fact that not all languages that might be mentioned are universally available. Each user of a particular computer will be aware of the language which is most suitable for his immediate purposes. In practice, it has proven useful to substantially implement the present invention in an Assembly language which provides a machine executable object code.

Because the computers and the monitor systems which may be used in practicing the instant invention consist of many diverse elements, no detailed program listings have been provided. It is considered that the operations and other procedures described above and illustrated in the accompanying drawings are sufficiently disclosed to permit one of ordinary skill to practice the instant invention or so much of it as is of use to him.

Thus, methods and apparatus which are most advantageously used in conjunction with a digital computer to provide improved graphics capability have been disclosed. The present invention's use of inversion points and scan line masking allows any arbitrary region to be defined, manipulated and transferred faster and more efficiently than systems previously found in the art.

While the present invention has been particularly described with reference to FIGS. 1-9 and with emphasis on certain computer systems, it should be understood that the figures are for illustration only and should not be taken as limitations upon the invention. In addition, it is clear that the methods and apparatus of the present inventions has utility in any application where graphic representations on a CRT or other display device are desired. It is contemplated that many changes and modifications may be made, by one of ordinary skill in the art, without departing from the spirit and scope of the invention as disclosed above.

I claim:

1. A computer display system, comprising:

display means for providing a display including a plurality of display elements, each of said display elements being selectively enabled;

memory means for storing a plurality of inversion points, each of said inversion points having a coordinate corresponding to an element on said display, wherein the coordinates of each inversion point specify orthogonal lines extending in the direction of subsequently enabled display elements from said inversion point and forming two boundaries of a contrasting area;

processing means coupled to said memory means for enabling elements on said display which correspond to said stored inversion points, and generating said contrasting areas on said display, the contrast of an area being a function of the coordinates of previously displayed inversion points;

whereby a region which comprises a plurality of inversion points may be displayed by enabling said corresponding elements and generating said associated contrasting areas on said display means.

2. The display system of claim 1 wherein said display means includes a plurality of raster scan lines comprising said elements defining said display.

3. The display system of claim 2 wherein said processing means includes reading means for reading said inversion points from said memory in the order in which said elements are scanned by said display means.

4. The display system of claim 3, wherein said processing means includes sorting means for sorting said inversion points into an ordered list in accordance with a predetermined convention and storing said list in said memory means.

5. The display system of claim 4 further including input means coupled to said processing means for inputting a region to be displayed into said memory.

6. The display system of claim 5 wherein said processing means further includes inversion point locating means for determining the coordinates of inversion points comprising said inputted region.

7. The display system of claim 6 wherein said processing means further includes logic means for executing logic operations between ordered lists of inversion points defining at least two regions.

8. The display system of claim 7 wherein said logic operations include the functions of logical AND, OR, NOT, and exclusive -OR.

9. The display system of claim 7 wherein said reading means reads a destination bitmap within said memory means, said destination bitmap including a plurality of inversion points representing regions currently being displayed on said raster scan display.

10. The display system of claim 9 wherein said memory means further includes at least one source bitmap, said source bitmap including a plurality of inversion points representing regions at least some portion of which may be transferred to said destination bitmap.

11. The display system of claim 10 wherein at least one scan line buffer is defined within said memory means, said scan line buffer being sufficiently large such that it contains adequate bits to represent all elements disposed along a scan line of said raster scan display.

12. The display system of claim 11 wherein said reading means sequentially reads inversion points in said source bitmap and provides a representation of said region in said scan line buffer thereby providing a scan of said region in said source bitmap corresponding to each scan line of said display means.

13. The display system of claim 12 wherein at least one scan line mask buffer is provided within said memory means, said scan line mask sequentially providing a scan of said destination bitmap such that the contents of said scan line mask are representative of a region stored within said destination bitmap in the order in which it is scanned by said display means.

14. The display system of claim 13 further including comparison means for comparing the contents of said scan line mask and said scan line buffer, such that prior to the transfer of the contents of said scan line buffer from said source bitmap to said destination bitmap for display, the contents of said scan line buffer are compared to the contents of said mask buffer for each scan line position of said display means.

15. The display system of claim 14 further including precedence control means for providing a predetermined priority as defined by a user between the contents of said scan line mask buffer and said scan line buffer as compared by said comparison means, and for transferring portions of said scan line buffer which have precedence to said destination bitmap for display.

16. The display system of claim 15 wherein each region inputted into said memory means is defined by at

least two inversion points having the same coordinates in different bitmaps, each of said inversion points corresponding to a different color to be displayed on said display means.

17. A method for generating and manipulating graphic representations on a computer controlled display system, said display system including a plurality of display elements, each of said elements being selectively enabled, comprising the steps of;

providing memory means within said computer including storage for a plurality of inversion points, each of said inversion points having a coordinate corresponding to an element on said display system, wherein the coordinates of each inversion point specify orthogonal lines extending in the direction of subsequently enabled display elements from said inversion point and forming two boundaries of a contrasting area;

inputting a region comprising a plurality of inversion points into said memory means;

displaying said inversion points comprising said region by enabling said corrsponding elements on said display and generating said contrasting areas on said display, the contrast of a display being a function of the coordinates of previously displayed points;

whereby said region is displayed by displaying said inversion points comprising said region and generating said associated contrasting areas on said display.

18. The method as defined by claim 17 further including the step of identifying and storing in said memory means the inversion points defining said region.

19. The method as defined by claim 18 wherein said display system includes a plurality of raster scan lines comprising said elements of said display.

20. The method as defined by claim 19 further including the step of reading said inversion points defining said region from said memory in the order in which said elements are scanned by said display system.

21. The method as defined by claim 20 wherein said storing step includes sorting said inversion points into an ordered list in accordance with a predetermined convention.

22. The method as defined by claim 21 wherein said sorting convention comprises sorting said inversion points in accordance with their coordinates, such that said points are sorted left to right and top to bottom relative to one another.

23. The method as defined by claim 22 further including the step of providing a one scan line buffer defined within said memory means, said reading means sequentially providing a representation of said region in said scan line buffer corresponding to each scan line on said display.

24. The method as defined by claim 23 further including the step of providing a one scan line mask buffer within said memory means, said mask buffer sequentially providing a representation of a region being displayed on said display such that the contents of said mask buffer correspond to each scan line of said display.

25. The method as defined by claim 24 further including the step of comparing the contents of said scan line buffer with the contents of said scan line mask.

26. The method as defined by claim 25 further including applying a predetermined priority between the contents of said scan line buffer and said scan line mask,

such that cnly selected portions of said scan 'ine buffer contents are displayed on said display system.

27. A method for selectively transferring data from a first location in a computer memory to a second location in said memory, comprising the steps of:

defining a one scan line buffer in said memory, said scan line buffer sequentially representing said data in said first location;

defining a one scan line mask buffer in said memory, said scan line mask sequentially representing data in said second location;

sequentially comparing the contents of said scan line buffer with the contents of said scan line mask prior to the transfer of the contents of said scan line buffer to said second location;

providing a predetermined precedence as defined by a user between the contents of said scan line buffer and said scan line mask, such that only selected data comprising said scan line buffer having priority is transferred to said second location;

whereby data is selectively transferred from said first location to said second location.

28. The method as defined by claim 27 wherein said second location comprises a plurality of bits, each bit corresponding to an element on a display system.

29. The method as defined by claim 28 wherein data in said second location is displayed on said display system.

30. The method as defined by claim 29 wherein said scan line buffer sequentially represents said data in said first location in the order in which said data will be displayed on said display system.

31. The method as defined by claim 30 wherein said scan line mask sequentially represents data in said second location in the order in which said data is displayed.

32. The method as defined by claim 31 wherein said data within each of said locations is representative of at least one region, said region comprising a plurality of inversion points each of said points having a coordinate corresponding to an element on said display, wherein coordinates of each inversion point specify orthagonal lines extending in the direction of subsequently enabled display elements from said inversion point and forming two boundaries of a contrasting area.

33. The method as defined by claim 32 wherein the process of determining the location of inversion points defining said region includes the steps of:

detecting horizontal line segments comprising said region;

defining inversion points at coordinates corresponding to the end points of said line segments.

34. The method as defined by claim 33 further including the step of sorting said inversion points defining said region in accordance with a predetermined convention.

35. The method as defined by claim 34 further including a process to determine if a specified point lies within said region, said region being defined by an ordered list inversion points arranged such that said inversion points are sorted in accordance with their coordinates left to right and top to bottom relative to one another, comprising the steps of:

defining at least one flag bit in said memory, said flag bit initially set in a first state;

scanning through said ordered list and switching said flag bit to a second state in a event that an inversion point in said list has a vertical coordinate greater than or equal to the vertical coordinate of said specified point and a horizontal coordiate less than that of said specified point;

determining the state of said flag bit.

* * * * *

United States Patent [19]

Hochsprung et al.

[11] **Patent Number: 4,661,902**

[45] **Date of Patent: Apr. 28, 1987**

[54] **LOCAL AREA NETWORK WITH CARRIER SENSE COLLISION AVOIDANCE**

[75] Inventors: **Ronald R. Hochsprung**, Saratoga; **Lawrence A. Kenyon, Jr.**, Sunnyvale; **Alan B. Oppenheimer**, Cupertino; **Gursharan S. Sidhu**, Menlo Park, all of Calif.

[73] Assignee: **Apple Computer, Inc.**, Cupertino, Calif.

[21] Appl. No.: **715,065**

[22] Filed: **Mar. 21, 1985**

[51] **Int. Cl.⁴** **G06F 15/16**
[52] **U.S. Cl.** **364/200**
[58] **Field of Search** ... 364/200 MS File, 900 MS File

[56] **References Cited**

U.S. PATENT DOCUMENTS

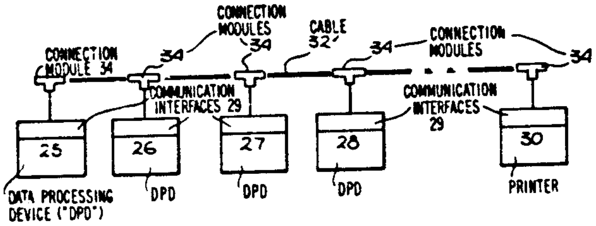
4,547,850 10/1985 Genma 364/200

Primary Examiner—Raulfe B. Zache
Attorney, Agent, or Firm—Blakely Sokoloff Taylor & Zafman

[57] **ABSTRACT**

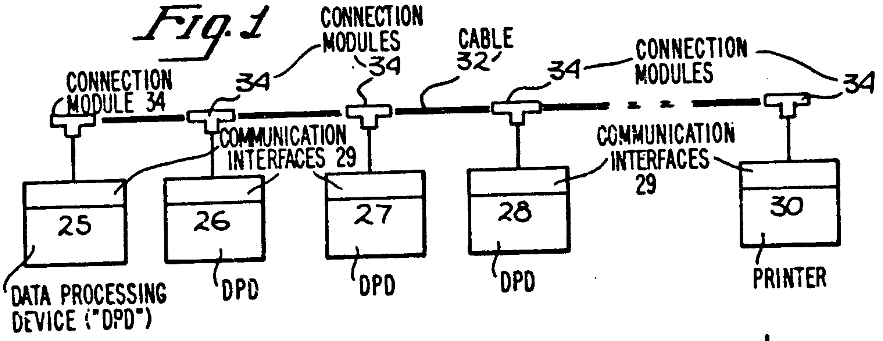
A local area network is disclosed including apparatus and methods for transferring data between a plurality of data processing resources ("agents") coupled to a cable. In the preferred embodiment, a plurality of agents are coupled to a common cable for data transmission and reception. An agent newly coupled to the cable dynamically assigns itself a unique address on the cable to which other agents may send data. Once an agent has been assigned a final address, it may then transmit to, and receive data from, other agents on the cable. An agent desiring to send data to a receiving agent follows a three step handshake, wherein the sending agent transmits an "RTS" signal and within a predetermined time must receive a "CTS"signal from the receiving agent. The sending agent then transmits a data frame within a predetermined time after the CTS signal is received. The failure to detect a return CTS signal within the predetermined time denotes a collision condition. Retransmission is attempted using a linear back off method which is adjusted based on previous cable traffic history.

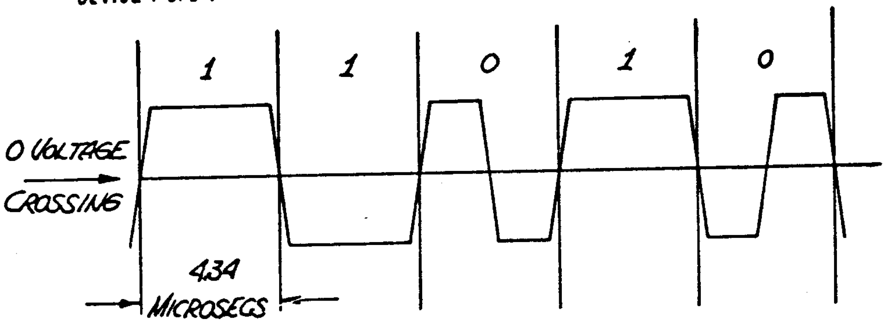
36 Claims, 15 Drawing Figures



AppleTalk Network

Fig. 1





FM-0 ENCODING

Fig. 2

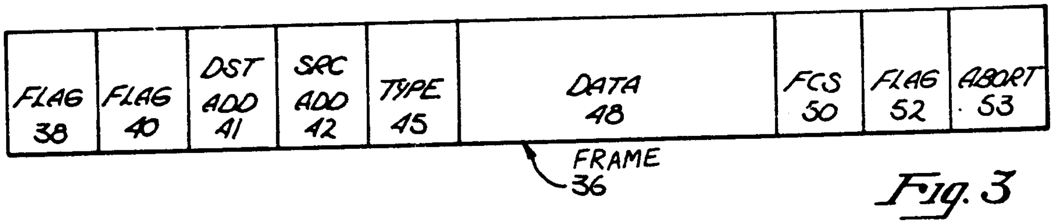


Fig. 3

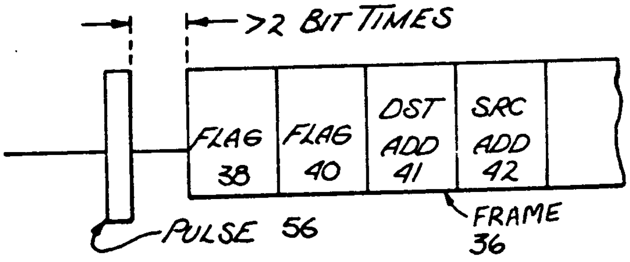


Fig. 4

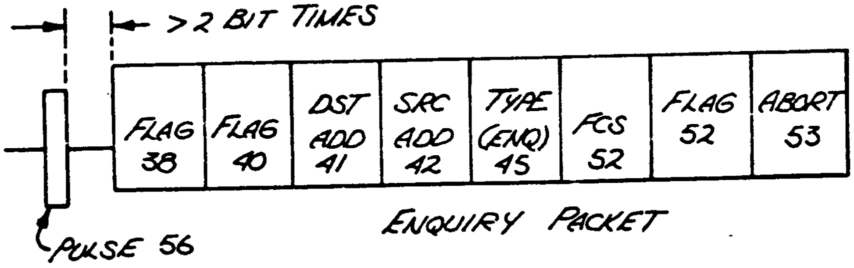
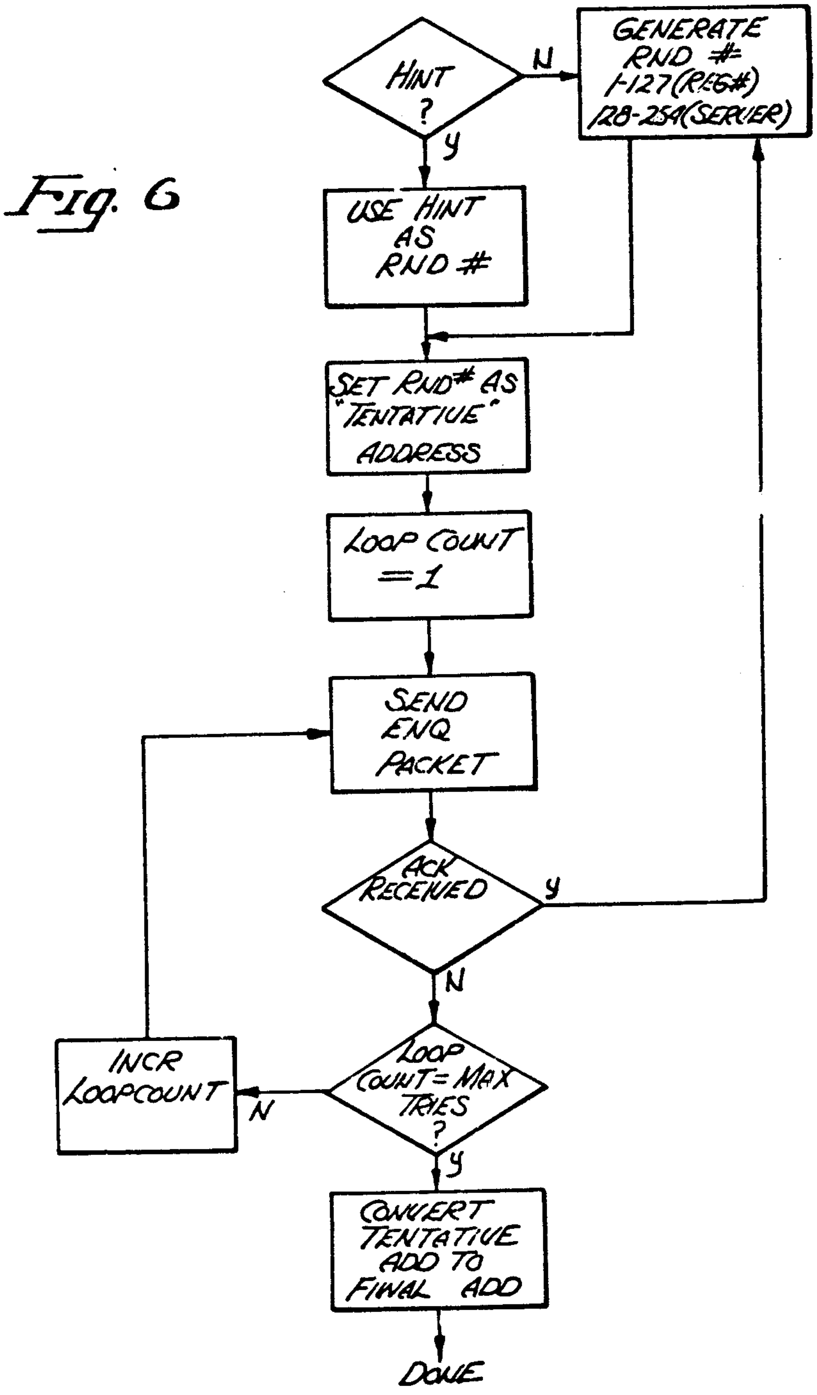


Fig. 5

ADDRESS ASSIGNMENT

Fig. 6



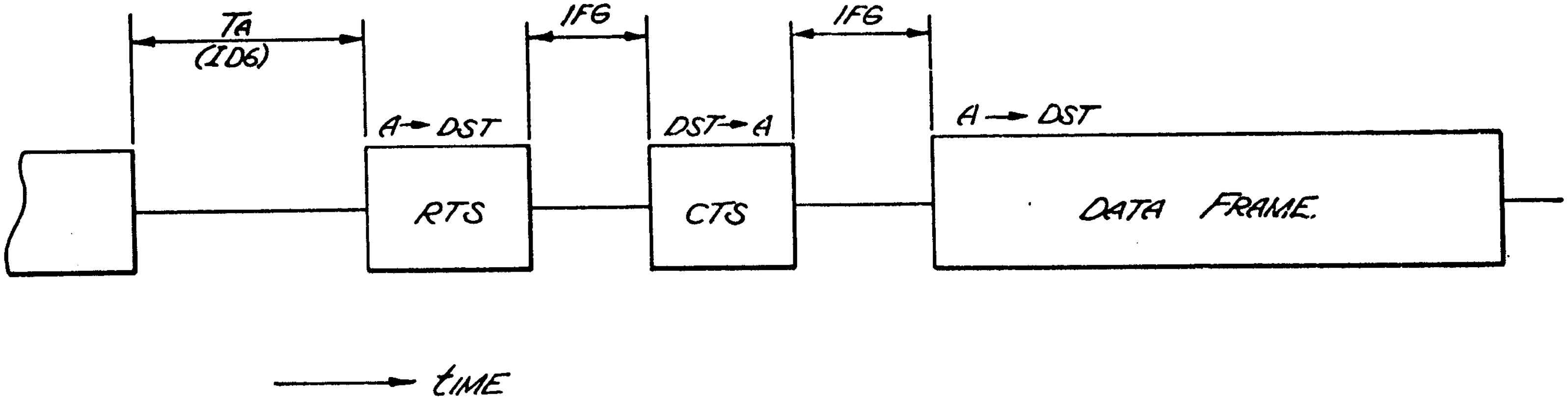


Fig. 7

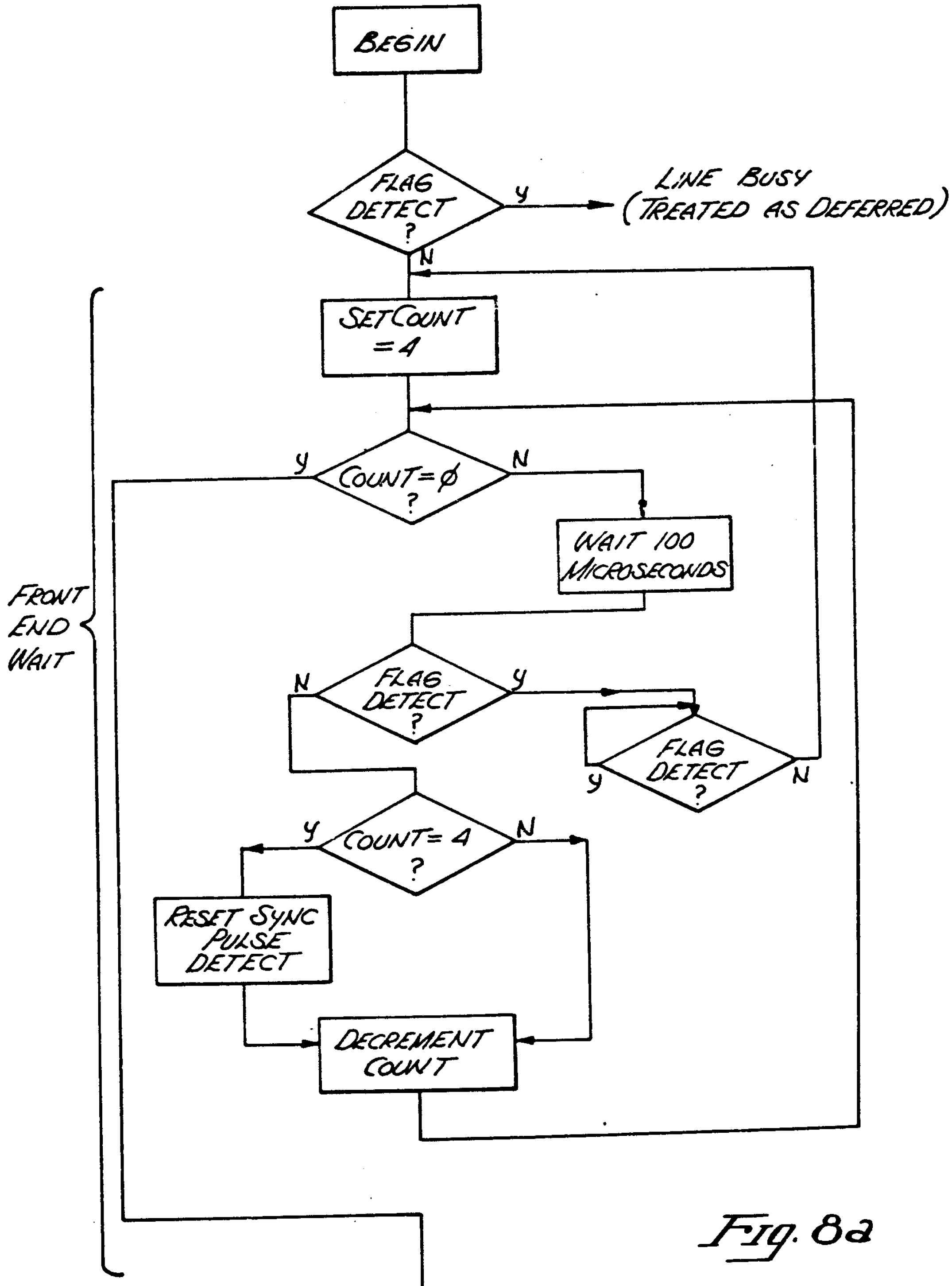


Fig. 8a

CONTINUED FIG 8b

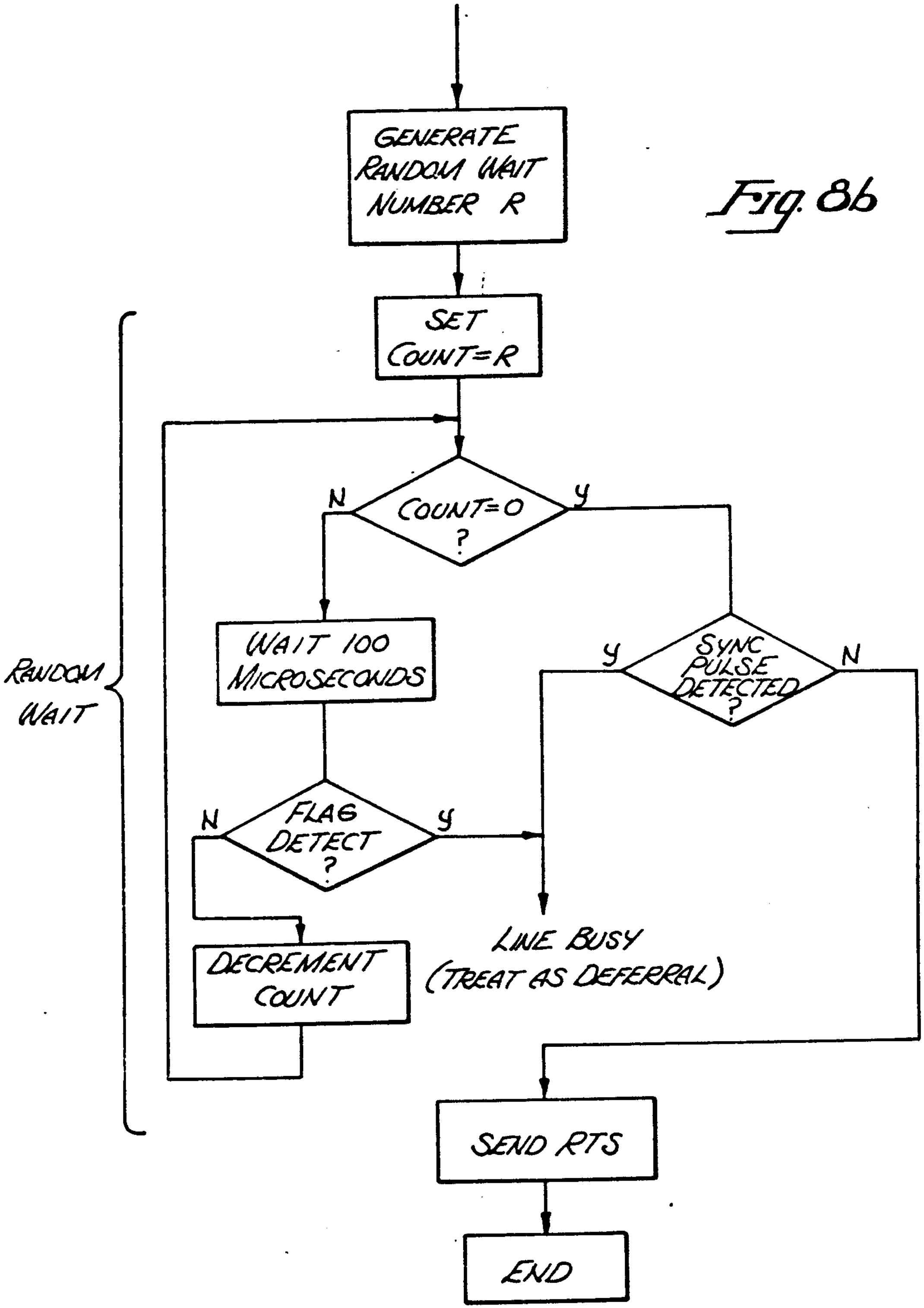


Fig. 8b

"APPLE_PAT_4_661_902_06" 87 KB 2000-02-21 dpi: 300h x 300v pix: 1876h x 2720v

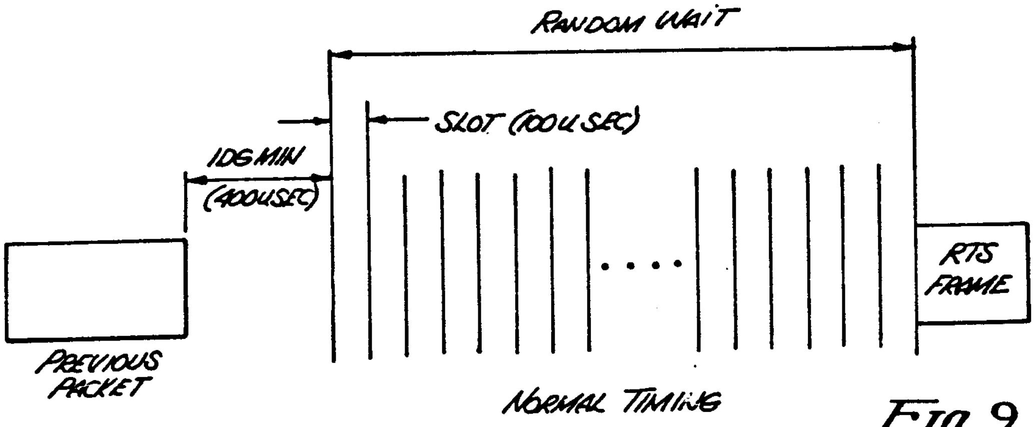


Fig. 9

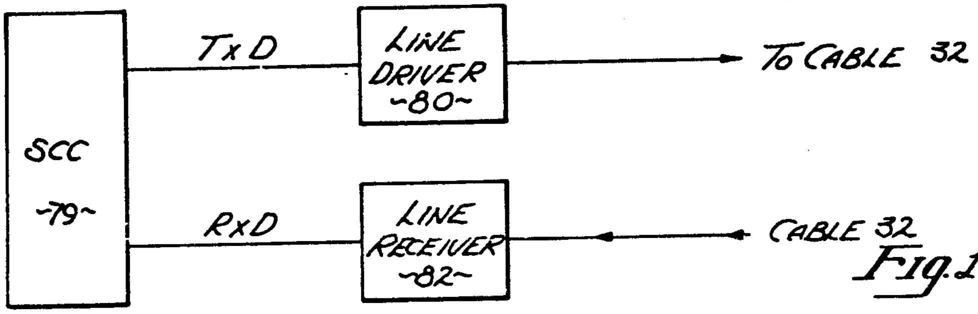


Fig. 10

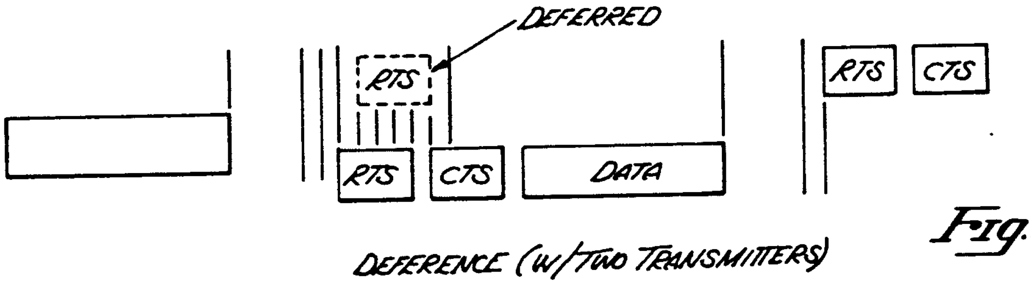


Fig. 11

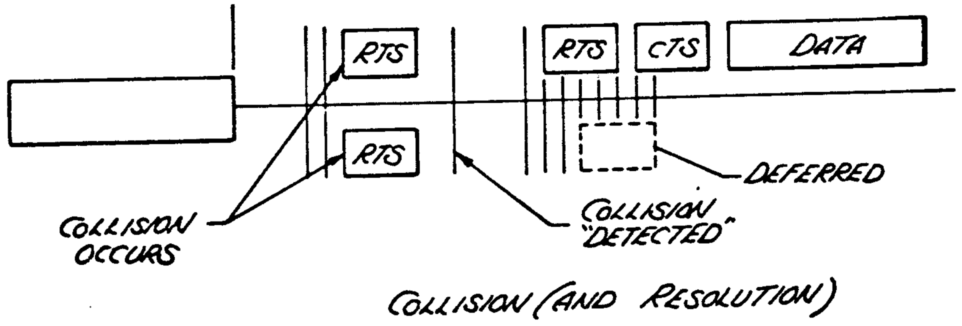
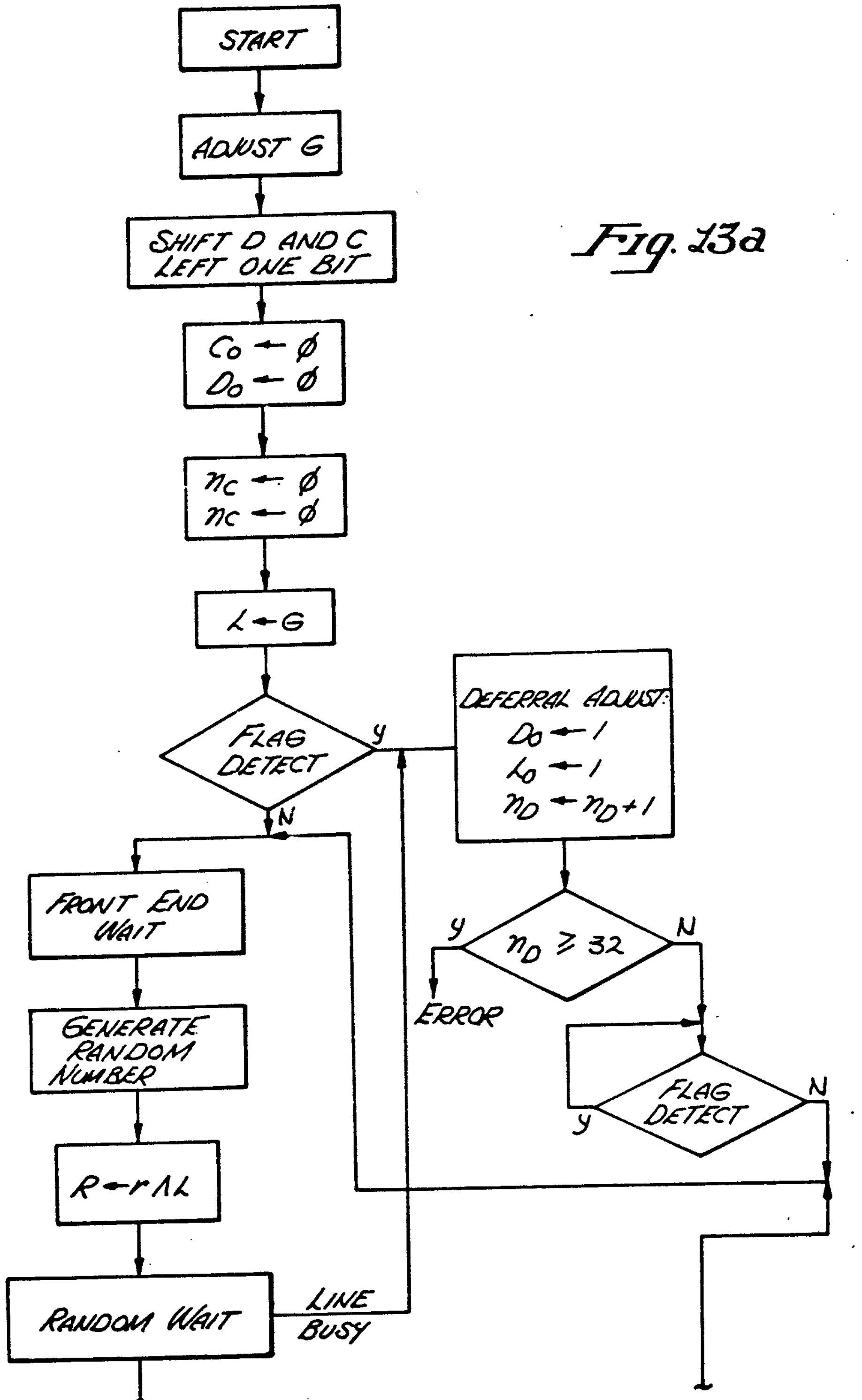


Fig. 12

TRANSMIT OPERATION



CONTINUE TO FIG 13b

Fig. 13a

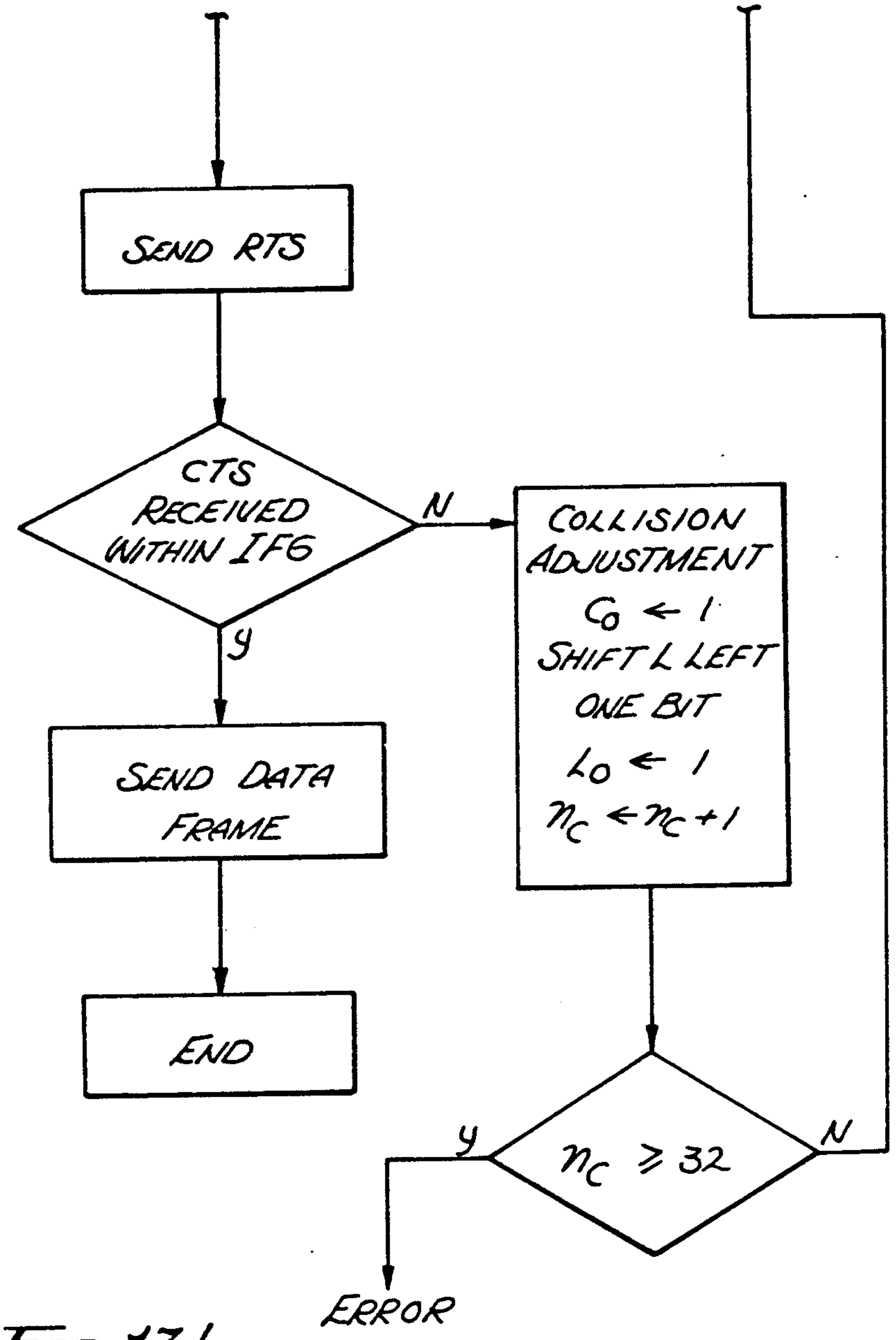
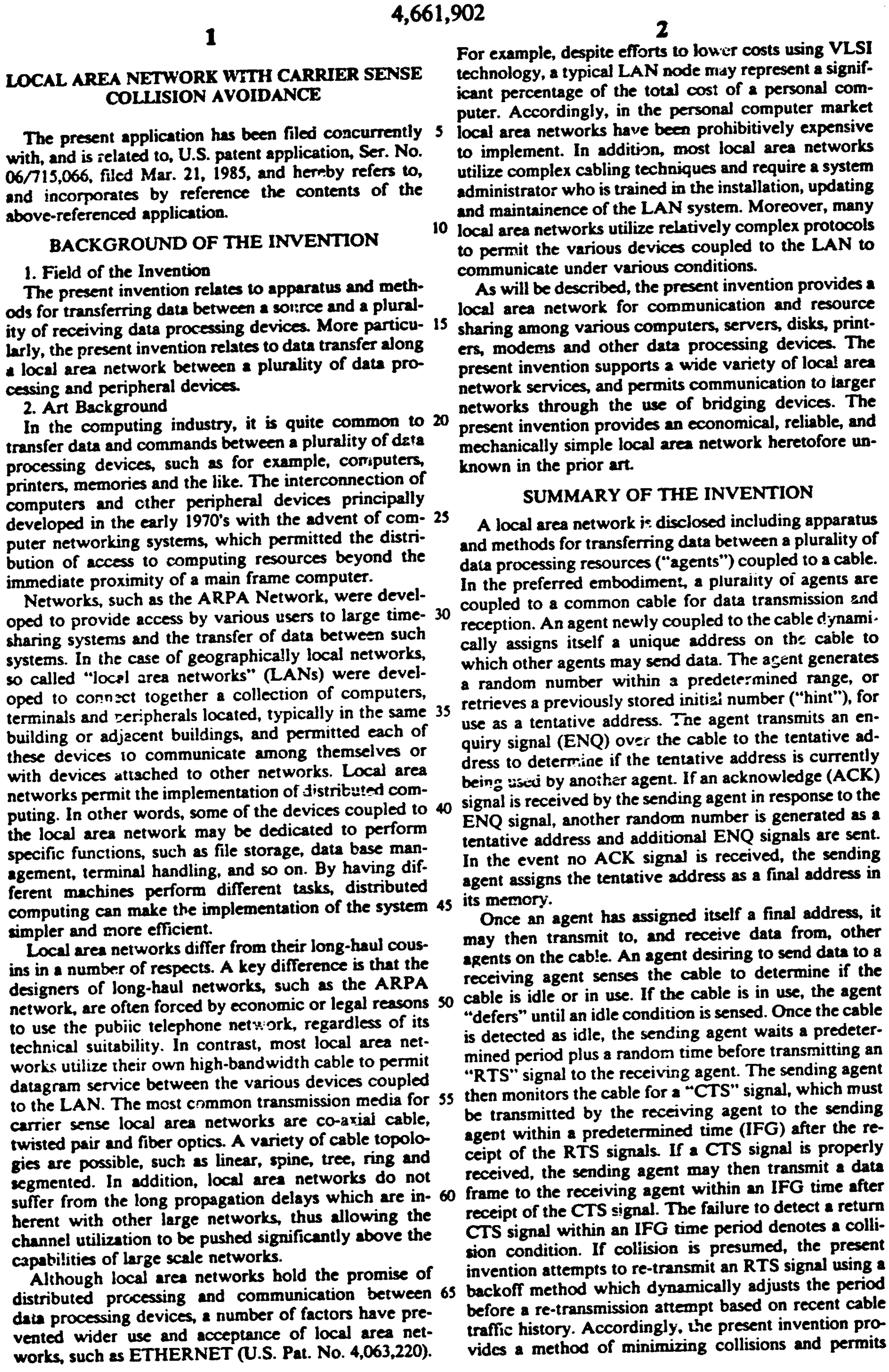

SEND RTS
CTS RECEIVED WITHIN IFG
N
Y
SEND DATA FRAME
END
COLLISION ADJUSTMENT
$C_0 \leftarrow 1$
SHIFT L LEFT ONE BIT
$L_0 \leftarrow 1$
$n_c \leftarrow n_c + 1$
$n_c \geq 32$
Y
N
ERROR

Fig. 13b



LOCAL AREA NETWORK WITH CARRIER SENSE COLLISION AVOIDANCE

The present application has been filed concurrently with, and is related to, U.S. patent application, Ser. No. 06/715,066, filed Mar. 21, 1985, and hereby refers to, and incorporates by reference the contents of the above-referenced application.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to apparatus and methods for transferring data between a source and a plurality of receiving data processing devices. More particularly, the present invention relates to data transfer along a local area network between a plurality of data processing and peripheral devices.

2. Art Background

In the computing industry, it is quite common to transfer data and commands between a plurality of data processing devices, such as for example, computers, printers, memories and the like. The interconnection of computers and other peripheral devices principally developed in the early 1970's with the advent of computer networking systems, which permitted the distribution of access to computing resources beyond the immediate proximity of a main frame computer.

Networks, such as the ARPA Network, were developed to provide access by various users to large timesharing systems and the transfer of data between such systems. In the case of geographically local networks, so called "local area networks" (LANs) were developed to connect together a collection of computers, terminals and peripherals located, typically in the same building or adjacent buildings, and permitted each of these devices to communicate among themselves or with devices attached to other networks. Local area networks permit the implementation of distributed computing. In other words, some of the devices coupled to the local area network may be dedicated to perform specific functions, such as file storage, data base management, terminal handling, and so on. By having different machines perform different tasks, distributed computing can make the implementation of the system simpler and more efficient.

Local area networks differ from their long-haul cousins in a number of respects. A key difference is that the designers of long-haul networks, such as the ARPA network, are often forced by economic or legal reasons to use the public telephone network, regardless of its technical suitability. In contrast, most local area networks utilize their own high-bandwidth cable to permit datagram service between the various devices coupled to the LAN. The most common transmission media for carrier sense local area networks are co-axial cable, twisted pair and fiber optics. A variety of cable topologies are possible, such as linear, spine, tree, ring and segmented. In addition, local area networks do not suffer from the long propagation delays which are inherent with other large networks, thus allowing the channel utilization to be pushed significantly above the capabilities of large scale networks.

Although local area networks hold the promise of distributed processing and communication between data processing devices, a number of factors have prevented wider use and acceptance of local area networks, such as ETHERNET (U.S. Pat. No. 4,063,220). For example, despite efforts to lower costs using VLSI technology, a typical LAN node may represent a significant percentage of the total cost of a personal computer. Accordingly, in the personal computer market local area networks have been prohibitively expensive to implement. In addition, most local area networks utilize complex cabling techniques and require a system administrator who is trained in the installation, updating and maintainence of the LAN system. Moreover, many local area networks utilize relatively complex protocols to permit the various devices coupled to the LAN to communicate under various conditions.

As will be described, the present invention provides a local area network for communication and resource sharing among various computers, servers, disks, printers, modems and other data processing devices. The present invention supports a wide variety of local area network services, and permits communication to larger networks through the use of bridging devices. The present invention provides an economical, reliable, and mechanically simple local area network heretofore unknown in the prior art.

SUMMARY OF THE INVENTION

A local area network is disclosed including apparatus and methods for transferring data between a plurality of data processing resources ("agents") coupled to a cable. In the preferred embodiment, a plurality of agents are coupled to a common cable for data transmission and reception. An agent newly coupled to the cable dynamically assigns itself a unique address on the cable to which other agents may send data. The agent generates a random number within a predetermined range, or retrieves a previously stored initial number ("hint"), for use as a tentative address. The agent transmits an enquiry signal (ENQ) over the cable to the tentative address to determine if the tentative address is currently being used by another agent. If an acknowledge (ACK) signal is received by the sending agent in response to the ENQ signal, another random number is generated as a tentative address and additional ENQ signals are sent. In the event no ACK signal is received, the sending agent assigns the tentative address as a final address in its memory.

Once an agent has assigned itself a final address, it may then transmit to, and receive data from, other agents on the cable. An agent desiring to send data to a receiving agent senses the cable to determine if the cable is idle or in use. If the cable is in use, the agent "defers" until an idle condition is sensed. Once the cable is detected as idle, the sending agent waits a predetermined period plus a random time before transmitting an "RTS" signal to the receiving agent. The sending agent then monitors the cable for a "CTS" signal, which must be transmitted by the receiving agent to the sending agent within a predetermined time (IFG) after the receipt of the RTS signals. If a CTS signal is properly received, the sending agent may then transmit a data frame to the receiving agent within an IFG time after receipt of the CTS signal. The failure to detect a return CTS signal within an IFG time period denotes a collision condition. If collision is presumed, the present invention attempts to re-transmit an RTS signal using a backoff method which dynamically adjusts the period before a re-transmission attempt based on recent cable traffic history. Accordingly, the present invention provides a method of minimizing collisions and permits reliable and economical data transfers between a plurality of agents coupled to the common cable.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 illustrates a local area network adapted to utilize the teachings of the present invention.

FIG. 2 is a timing diagram illustrating the present invention's use of frequency modulated (FM-0) encoding.

FIG. 3 illustrates the frame format utilized by the present invention to transfer data to various data processing devices coupled to the local area network.

FIG. 4 illustrates the present invention's use of a synchronization pulse prior to the transmission of a frame.

FIG. 5 illustrates an enquiry (ENQ) frame utilized by the present invention during dynamic address assignment.

FIG. 6 is a flow chart illustrating the sequence of operations utilized by a data processing device coupled to the present invention during dynamic address assignment.

FIG. 7 diagrammatically illustrates the present invention's use of handshake signals between sending and receiving data processing devices prior to the transmission of a data frame.

FIGS. 8(*a*) and 8(*b*) are a flow chart illustrating the sequence of operations of a sending device to obtain cable access.

FIG. 9 is a diagrammatical illustration of the transmission of an "RTS" frame by a sending device after sensing an idle cable.

FIG. 10 is a block diagram illustrating the present invention's use of a serial controller device coupled to the local area network.

FIG. 11 illustrates the present invention's collision avoidance method including deference.

FIG. 12 illustrates the collision and resolution mechanism of the present invention wherein two "RTS" signals collide along the local area network.

FIGS. 13(*a*) and 13(*b*) are a flow chart illustrating the generation of the random wait period R.

DETAILED DESCRIPTION OF THE INVENTION

A local area network including apparatus and methods for transferring data between a plurality of data processing resources coupled to a common cable is disclosed. In the following description for purposes of explanation, specific numbers, bytes, registers, addresses, times, signals, and formats, etc. are set forth in order to provide a thorough understanding of the present invention. However, it will be apparent to one skilled in the art that the present invention may be practiced without these specific details. In other instances, well known circuits and devices are shown in block diagram form in order not to obscure the present invention unnecessarily.

Referring to FIG. 1, the present invention may include a plurality of data processing devices ("DPD") identified generally by the numerals 25 through 28 as well as peripheral devices such as printer 30 (or other devices such as, for example, a global memory, a disk drive and the like). For purposes of this Specification, all data processing and peripheral devices which are coupled to the present invention's local area network are collectively referred to as "agents". As shown, processing devices 25, 26, 27, 28 and printer 30 are interconnected for data transfer to one another by a common cable 32. As shown in FIG. 1, each agent includes a communication interface 29 which is coupled to the agent and to common cable 32 through connection modules 34, connection modules 34, which in the presently preferred embodiment, contain a passive coupling transformer, resistive and capacitive circuits, and are known in the art for coupling each of the data processing and other devices to cable 32. Communication interface 29 comprises circuitry, logic and software, known in the art for sensing cable 32 to determine if it is clear of carrying data of another agent. In addition, communication interface 29 includes known circuitry and logic to transmit and receive data on cable 32, as well as provide timing, random number generation, and signal generation for implementing the teachings of the present invention as described herein. For purposes of clarity, the specific circuits, programming routines, and other logic comprising communication interface 29 will not be disclosed in the Specification, as means for accomplishing its functions as defined in this patent are known in the art. Cable 32 is appropriately terminated to eliminate signal reflections. In the preferred embodiment, cable 32 is terminated with 100 Ohm resistors, and is comprised of a twisted pair cable. It will be appreciated by one skilled in the art that cable 32 may comprise any shared media, such as coaxial cable, fiber optics, radio channel and the like. Since, in the present embodiment, the agents are passively coupled to cable 32, a failure of an agent or a connection module will not disrupt communication over cable 32.

As will be described, the present invention provides a local area network permitting synchronous serial communication and data transfer between data processing devices 25 through 28, and other peripheral devices such as printer 30, utilizing protocols and collision avoidance and detection methods and apparatus. The present invention's architecture and protocols minimize handshake and collision detection complexities common in prior art systems, and permit high speed serial communication along cable 32. The present invention permits access to various resources coupled to the network, such as data stored in local memories, or disk, and the common use of global printers, without the necessity of complex or active termination circuitry at the cable end, or the need to predefine addresses for each device coupled to cable 32. In the current embodiment, the present invention operates at approximately 230 kilobits per second through a shielded, twisted pair cable 32, and is driven in accordance with EIA standard RS-422 balanced voltage specifications.

Referring to FIG. 2, data is encoded and transmitted on cable 32 using a self-clocking technique known as FM-0 (bi-phase space), such that each bit cell, typically 4.34 microseconds in duration, contains a state transition at its end, thereby providing necessary timing information to the receiver. As illustrated, zeros are encoded by adding an additional zero-crossing transition at mid-cell, such that two zero crossings are detected for each 4.34 microsecond cell. Similarly, a logical one is provided in a particular cell by a zero-crossing transition only at its end. Accordingly, through the use of FM-0 encoding, clocking information is carried by the data signal itself, and permits the present invention to operate in a synchronous mode.

Referring now to FIG. 3, the present invention utilizes a basic unit of data transmission known as a "frame" 36. Frame 36 includes a preamble consisting of two or more synchronization ("flag" bytes 38 and 40. In the presently preferred embodiment, each synchronization byte comprises the bits 01111110. As is known, synchronization bytes 38 and 40 permit receiving data processing unit coupled to cable 32 to synchronize their receiving circuits, and to receive necessary clock information (through the use of FM-0 encoding). Following synchronization bytes 38 and 40 is an eight-bit destination address 41 which specifies the address of the data processing agent for which the frame is intended. A source address 42 comprises an eight-bit address of the data processing agent transmitting the frame of information. A "type" field 45 is provided to specify the type of frame which is transmitted through the use of various codes. For example, type field 45 may designate an acknowledge (ACK) frame, an enquiry (ENQ) frame, as well as an RTS and CTS frame, which will be described more fully below. The type field is followed by a multi-byte data field (possibly of zero length) which may contain raw data, messages, and the like to be transmitted between the agents coupled to cable 32. Following the data field 48 is a 16-bit frame check sequence which is computed as a function of the contents of the source address, destination address, type and data fields. In the present embodiment, the frame check sequence (FCS) is defined using the standard CRC-CCITT polynomial. The frame check sequence 50 is followed by an eight-bit synchronization ("flag") trailer 52 (consisting of the logical bits 01111110). and an abort sequence 53 which consists of eleven or more ones in a row. Abort sequence 53 is used to delineate the end of the frame 36 to agents coupled to cable 32. Frame 36, as will be described, is transmitted along line 32 in a serial synchronous fashion using a handshake sequence of control frames, which are in turn followed by the data frame 36 illustrated in FIG. 3.

As shown in FIG. 4, prior to the transmission of a frame, a transmitting agent coupled to cable 32 transmits a synchronization pulse 56 which, is followed by an idle period greater than two bit times and less than 10 bit times. Pulse 56 may comprise any signal containing a zero crossing. In the present embodiment, as illustrated in FIG. 10, each agent coupled to cable 32 utilizes a Zilog Z8530 SCC serial communication controller chip 79, accessing cable 32 through a line driver 80 and a line receiver 82. (See, Zilog Technical Manual, Z8030/Z8530 SCC Serial Communications Controller, January, 1983.) The Z8530 SCC device 79 includes circuitry which searches for synchronization (flag) bits when in a "hunt" mode. As previously discussed, the present invention utilizes a synchronization (flag) byte having the bit states 01111110. In addition, the serial communication controller chip has the capability of detecting a missing clock cycle, and setting a missing clock bit within the device if following a given zero crossing, a predefined period (greater than 1 bit time) elapses without a successive zero crossing of the incoming signal R_xD.

The pulse 56 provided by a transmitting agent coupled to cable 32 will be taken as a clock by all receiving agents. However, since it is followed by an idle period greater than two bit times, a missing clock is detected and the missing clock bit is set in the SCC device 79 of each agent connected to cable 32, thereby notifying agents that cable 32 is in use. In the preferred embodiment, the synchronization pulse 56 is obtained by momentarily enabling the line driver 80 for at least one bit time. This causes transmission, for the pulse duration

time, of the signal T_xD onto cable 32, thus ensuring at least one zero crossing in the synchronization pulse 56. In addition, the detection of synchronization (flag) bits (i.e., 38 and 40) clears the "hunt" bit in the Z8530 device, and permits each agent coupled to cable 32 to more efficiently detect whether or not cable 32 is currently in use prior to the transmission of a frame, as well as providing the necessary synchronization bits to allow the receiving agent to clock itself to the incoming data frame. It will be appreciated that although the present invention currently uses a Z8530 SCC device for detecting missing clock cycles and synchronization bytes, that other circuitry may equally be used for the same function.

Each agent coupled to cable 32 is identified by a unique binary address along the cable. One feature of the present invention is that an agent coupled to cable 32 does not require a predefined permanent address. Thus, for example, device 27 may be removed from cable 32 and then recoupled to another cable at a different location without need to configure an address. When an agent is newly coupled to cable 32, a unique protocol is followed such that an address is dynamically generated and assigned by the agent itself. In the presently preferred embodiment, the address of each agent is identified using an eight bit identifier (where no agent may have a zero address or an address of 255).

Referring briefly to FIG. 6, the sequence of operations which an agent utilizes in order to determine and assign itself an address is illustrated. It is apparent that to prevent disrupted service, no agent may acquire the same address as an already functioning agent. In practice, the address of agents may be allocated between general data processing devices and "servers" which may comprise main frame or other machines. In the present embodiment, addresses 1 through 127 are allocated for general purpose agents, and addresses 128 through 254 are allocated for use by servers. As shown in FIG. 6, upon being coupled to cable 32, each agent either generates an arbitrary random number within a predefined range or obtains a starting number from some long term, non-volatile memory (for example, read-only-memory or magnetic medium) referred to as a "hint". This random number (or "hint") is treated as a "tentative" address, and the agent then transmits an enquiry (ENQ) frame which utilizes the tentative address as a destination address. The enquiry frame transmitted is of the form illustrated in FIG. 5, and includes an initial pulse 56 separated by at least two bit times prior to the synchronization (flag) bytes 38 and 40, previously described with reference to FIG. 3. The destination address 41 of FIG. 5, as well as the source address 42, contains the tentative address generated randomly or through the hint. It will be noted that the type field 45 in FIG. 5 contains a binary code which identifies the frame of FIG. 5 as an "enquiry" (ENQ) frame for use in address assignment. This ENQ frame is transmitted over cable 32. In the event another agent has previously been assigned the tentative address, the agent already using the tentative address receives the ENQ frame, and in response transmits an acknowledge frame (ACK) back to the transmitting agent. In practice, the ACK frame is structured similarly to the ENQ frame disclosed in FIG. 5, except that the type byte contains a binary code identifying the packet as an ACK.

As illustrated in FIG. 6, in the event that an ACK frame is received by the transmitting agent, that agent must then generate another random number as a tentative address and repeat the transmission of this new tentative address along cable 32. In the event that no ACK frame is received, the agent newly coupled to the cable continues to send ENQ frames onto the cable until some predefined maximum number of tries has occurred. If, after a predefined number of attempts, no ACK frame has been received, the transmitting agent then assigns the tentative address as its final address for all future communication along cable 32. The repeated transmission of ENQ frames is used to avoid instances where a particular agent which may be using the tentative address may currently be busy, and thus miss the reception of an enquiry.

Once an agent has been assigned a final address, it may then communicate with other agents coupled to cable 32 utilizing a handshake protocol and collision avoidance mechanism described below. Referring to FIGS. 7, 8(*a*), 8(*b*) and 9, communication between agents coupled to cable 32 occurs through a three-way handshake process. The purpose of the handshake sequence is to control the access to the shared cable 32 in an orderly fashion that reduces the probability of a collision. Each transmission including the handshake (known as a "dialogue") must be separated by a minimum inter-dialogue gap (IDG), which in the present embodiment comprises 400 microseconds. In addition, the frames within a single transmission (dialogue) must follow one another within a maximum interframe gap (IFG) of, in the current embodiment, 200 microseconds. A collision is said to occur when two or more agents transmit at the same time on cable 32.

Referring to FIGS. 7 and 8(*a*) and 8(*b*), the sending agent, for example data processing agent 25, which desires to communicate with another agent coupled to cable 32, executes the operations set forth in the flow chart of FIGS. 8(*a*) and 8(*b*). A sending agent prior to transmission determines whether or not the "hunt" bit in the Z8530 SCC serial controller, or other appropriate hardware, has detected a synchronization (flag) byte passing along cable 32. If a synchronization (flag) byte has been detected, and no abort byte has followed, then cable 32 is currently in use and the agent wishing to transmit "defers" its transmission. In the event that no synchronization pulse 56 or synchronization (flag) bytes (38 and 40) are detected, the agent desiring to transmit data executes a front end wait operation, as best illustrated in FIGS. 8(*a*). The front end wait operation consists of a series of four waiting periods, in the present embodiment of 100 microseconds each, following each of which the flag detect ("hunt bit") is checked to see if a synchronization (flag) byte has been received on the cable 32. The detection of a flag byte denotes that some other agent is using the cable 32. In such event, the sending agent must wait for the flag detect (hunt bit) to clear, thus signalling the end of utilization of cable 32. At this point, the entire front end wait sequence illustrated in FIGS. 8(*a*) and 8(*b*) is repeated.

If, on the other hand, a flag byte is not detected, this indicates that during the front end wait sequence no other agent has attempted to use the cable, and a random wait operation is then executed. In addition, during the front end wait operation, the synchronization pulse detect is cleared after the first 100 microsecond wait.

Before proceeding to execute the random wait operation, illustrated in FIG. 8(*b*), a random wait number R is generated (the details regarding the generation of R will be discussed below). As shown,.the random wait operation cycles R times through a basic operation of waiting 100 microseconds before checking to see if flag has been detected (hunt bit cleared). If, at any point a flag is detected, then another agent is using the cable 32 and the sending agent must defer its transmission. If, however, at the end of the random wait sequence the cable is still quiet (not in use) then one last check is made to see if a synchronization pulse has been detected, before sending an RTS frame, as will be described.

If the cable 32 remains idle throughout this randomly generated waiting period R, the transmitting agent proceeds to transmit a synchronization pulse 56 followed by an "RTS" frame along cable 32 to the receiving agent. An RTS frame is structured substantially the same as the ENQ frame illustrated in FIG. 5, however, the type field contains a binary code identifying the frame as an RTS rather than an ENQ frame. The receiving agent, upon receiving the RTS frame from the transmitting agent, transmits a "CTS" frame back to the original transmitting agent within the maximum interframe gap (IFG) period. As in the case of the RTS frame, a CTS frame transmitted by a receiving agent is structured substantially the same as the ENQ packet illustrated in FIG. 5, except that the type field contains a code identifying the frame as CTS. Once the original transmitting agent, for example data processing unit 25, receives the CTS frame, a full data frame 36, as illustrated in FIG. 3, is transmitted to the receiving agent within one IFG of the receipt of the CTS frame. In the event that the transmission of a CTS or data frame does not occur within an IFG, then the transmitting agent assumes that a collision has occurred or the destination agent is inactive or otherwise unavailable.

If a general broadcast to all agents coupled to cable 32 is desired, the transmitting agent sends an RTS frame with a destination address of 255 to all agents on the line, and waits for an IFG period to elapse prior to sending a data frame 36 also having a destination address of 255. Accordingly, in the case of general broadcasts along cable 32, the transmitting agent does not wait for return CTS frames, but rather, immediately proceeds to conduct a general broadcast once the IFG period has elapsed after transmitting an RTS frame. In addition, by providing within the RTS frame a destination address field 41 having a particular value (255) corresponding to a broadcast address, only one RTS frame need be transmitted to all agents at the various addresses along cable 32.

It will be appreciated by one skilled in the art that the purpose of the three step handshake prototol described above is to avoid collisions by restricting the periods in which collisions are highly likely (typically during the RTS and CTS frame exchanges), and to spread out in time the cable access of transmitters waiting for the cable 32 to become idle prior to the beginning of a transmission. A successful RTS-CTS frame exchange signifies that a collision did not occur, and that all agents desiring to transmit have sensed the coming data frame transmission, and are waiting until the data exchange is complete prior to attempting to gain control of the cable.

In the event that another agent begins a transmission during the RTS-CTS frame exchange described above, it will be appreciated that the CTS frame will not be properly received (e.g. the frame check sequence is invalid), and that the sending agent may then assume that a collision has occurred. A collision will prevent a complete RTS an CTS frame exchange, and thereby prevent a proper handshake from occurring. Normally, if an agent desiring to transmit data on cable 32 senses that the cable is currently in use, it defers the transmission of its own RTS until the cable is idle (see FIGS. 11 and 12).

Referring now to FIGS. 9, 13(*a*) and 13(*b*), the sequence of operations executed by the present invention to obtain the value of random wait number R (as previously discussed with reference to FIG. 8) will be described in detail. As will be appreciated, the present invention dynamically modifies the random wait number R in response to recent cable traffic history. The method utilized by the present invention presumes that if collisions have been assumed for recently sent data frames, the cable 32 is currently the subject of heavy loading and high bus contention. A random waiting period R prior to retransmission attempt spreads out in time bus access for the various agents contending for cable use. Accordingly, the operations illustrated in FIGS. 13(*a*) and (*b*) are executed to generate and adjust the random wait number R utilized in accordance with the sequence of operations set forth in FIGS. 8(*a*) and (*b*). In the present invention, eight bit shift registers are provided in order to keep track of collision and deferral histories for each agent coupled to cable 32. For purposes of this Specification, the variable "C" denotes an eight bit shift register which is utilized to keep track of the collision history for the last eight data messages which an agent has attempted to send, and a variable "D" which denotes an eight bit shift register representing the deferral history for the last eight messages which have been attempted to be sent. As previously discussed, a collision is presumed if the RTS-CTS frame handshake protocol fails to occur within the IFG period, and a deferral is deemed to occur if an agent, prior to transmitting a message, detects a flag byte or synchronization pulse 56 thereby indicating that the cable is in use. A variable "G" is defined as a four bit global mask which represents a number signifying a modification factor representative of all previous messages which the the agent has attempted to transmit. A variable "L" is defined as a local mask which is representative of attempts to transmit the current message by an agent coupled to cable 32. In addition, N_C is defined as the number of collisions which have been assumed for a particular data packet, and N_D is defined as the number of deferrals which have occurred prior to the transmission of the current data packet.

As illustrated best in FIGS. 13(*a*) and (*b*) prior to the transmission of a new data packet, the variable G is adjusted, as follows:

If the number of bits set (i.e. equal to 1) in the eight bit register "C" is greater than 2, then all bits in the four bit shift register defining "G" are shifted to the left [least significant bit (LSB) toward most significant bit (MSB)] one bit. In addition, G_0 (the least significant bit of four bit shift register G) is set to 1 and the eight bits comprising C are set to 0.

If the number of bits set in the eight bit register "C" is less than or equal to 2, then D is examined, and if the number of bits set in "D" is less than 2, then the present invention shifts the contents of G right (MSB toward LSB) one bit; sets G_3 (MSB of G) equal to 0 and sets the value of D equal to 255.

Once G has been adjusted, the present invention then shifts the contents of registers D and C left one bit (toward MSB) and sets the least significant bit (LSB) of C and D equal to 0. Similarly, variables N_C and N_D which denote the number of collisions and deferrals for the particular message to be sent are also set equal to 0. Moreover, as shown in FIG. 13, the value of L is then set equal to the value of G.

Prior to beginning the front end wait sequence disclosed in FIG. 8(*a*), the present invention determines whether or not a flag detect (i.e. flag byte) has been detected along cable 32. In the event that no flag byte has been detected, the present invention then executes the sequence of operations illustrated in FIG. 8(*a*) for the front end waiting period. Subsequent to the fixed front end wait sequence, the present invention generates a random number "r" within a predetermined range, and then calculates the value of "R" by logically "AND"ing the value of r with the previously determined value of L (local mask variable). Once the value of R is determined, the present invention then follows the random wait cycle illustrated in FIG. 8(*b*), and upon completion of the random waiting period, transmits the RTS frame as shown in FIG. 13(*b*).

If the CTS frame is received by the sending agent within the IFG period, as previously discussed, then the data frame is transmitted, and the message dialogue has been completed. If, on the other hand, a flag is detected prior to the beginning of the front end wait sequence, a deferral adjustment is provided wherein D_0 (the LSB of register D) is set to 1 and L_0 is set to 1. In addition, the deferral adjustment includes the setting of N_D equal to N_D+1. The flag detect (hunt bit) is once again checked. As shown in FIG. 13, this deferral adjustment occurs in cases where it is determined that the line is busy prior to transmission.

In the event the RTS/CTS handshake is not successful, then a collision is presumed and a collision adjustment occurs. C_0 is set equal to 1 and the value of L is shifted left (LSB to MSB) one bit. In addition, L_0 is set equal to 1 and N_C is set equal to N_C+1, as illustrated in FIG. 13(*b*).

It has been found that the utilization of the steps illustrated in FIG. 13 dynamically adjust the randomly generated value of r such that the period of time (in 100 microsecond increments) which an agent waits in addition to the front end waiting period prior to attempting a transmission is modified in accordance with recent cable traffic history. This modification of the random waiting period significantly increases the probability of a successful RTS/CTS frame exchange, and thereby avoids collisions along cable 32.

Accordingly, apparatus and methods have been disclosed having particular utility when used in conjunction with a local area network. The present invention provides a network which permits any agent to be coupled to the cable at any point, and assign itself a unique address. In addition, the present invention's novel collision avoidance protocol minimizes the probability of collisions occurring on the cable, and if a collision occurs, provides a greater probability of success for subsequent re-transmissions.

Although the present invention has been described particularly with reference to FIGS. 1–13, it will be apparent to one skilled in the art that the present invention has utility far exceeding that disclosed in the Figures. It is contemplated that many changes and modifications may be made, by one of ordinary skill in the art, without departing from the spirit and scope of the invention as disclosed above.

We claim:

1. A communication medium for transferring data between a plurality of data processing devices ("agents") including a sending agent and a receiving agent, comprising:

sensing means coupled to said sending agent for therein if said medium is currently carrying data of another agent and is thereby in use;

timing means coupled to said sending agent for timing a first predetermined waiting period once said medium is idle and available for use;

first random number generating means coupled to said sending agent for generating a random number within a predefined range corresponding to a second waiting period prior to transmittng data on said communication medium to said receiving agent;

first signal generation means coupled to said sending agent for generating a first signal and transmitting said signal to said receiving agent;

signal receiving means coupled to said sending agent for receiving a signal transmitted from said receiving agent to said sending agent within a second predetermined time (IFG) after said sending agent has transmitted said first signal;

data transmission means coupled to said sending agent for transmitting data to said receiving agent within said IFG time after receiving said signal from said receiving agent;

whereby data is transferred between said sending and receiving agents coupled to said communication medium.

2. The communication system as defined by claim 1, further including collision assumption means coupled to said sending agent for producing a collision signal in the absence of the receipt of said signal from said receiving agent by said signal receiving means within said IFG time.

3. The communication system as defined by claim 2, wherein said random number (R) is generated by said random number generating means such that:

$$R=r\Lambda L$$

where:

r=a random number within said predefined range;

Λ denotes a logical AND operation;

L=a Local variable representative of the collision and deferral history of said sending agent for said data to be sent.

4. The communication system as defined by claim 3, further including a global mask variable G which is representative of the collision and deferral history of said sending agent for all previous data transferred, said variable G being adjusted prior to the initiation of each new data transfer.

5. The communication system as defined by claim 2, wherein said sensing means includes pulse detection means for detecting a synchronization pulse on said communication medium, said synchronization pulse preceeding other signals transmitted by agents coupled to said communication medium.

6. The communication system as defined by claim 5, wherein said synchronization pulse preceeds said other signals by at least 2 bit times.

7. The communication system as defined by claim 5, wherein said first signal comprises an RTS frame including at least one sychronization flag byte having a predefined bit sequence and a type field containing an RTS byte.

8. The communication system as defined by claim 7, wherein said sensing means includes means for sensing said flag byte, the sensing of said flag bytes denoting that said medium is in use.

9. The communication system as defined by claim 8, wherein said signal received from said receiving agent by said receiving means comprises a CTS frame including at least one synchronization flag byte and a type field containing a CTS byte.

10. The communication system as defined by claim 9, wherein said data transmitted by said sending agent is in the form of a data frame including a plurality of data bytes preceded by at least one synchronization flag byte, a destination address and a source address byte.

11. The communication system as defined by claim 10, wherein said data frame further includes a frame check sequence (FCS) and an abort sequence of bits following said plurality of data bytes.

12. The communication system as defined by claim 11, wherein said IFG time is generally 200 microseconds.

13. The communication system as defined by claim 12, wherein said sensing means includes a Z8530 SCC serial communication controller device.

14. The communication system as defined by claim 11, wherein said communication medium comprises a twisted pair cable.

15. The communication system as defined by claim 14, wherein said twisted pair cable is terminated by 100 Ohm resistors.

16. The communication system as defined by claim 11, wherein said data transfers on said communication medium are separated by a third predetermined time (IDG).

17. The communication system as defined by claim 16, wherein said third predetermined time is generally 400 microseconds.

18. The communication system as defined by claim 17, wherein said signals are transmitted on said communication medium using FM-0 encoding.

19. A method for transferring data on a data communication medium between a plurality of data processing devices ("agents"), including a sending agent and a receiving agent, comprising the steps of:

sensing said medium to determine if said medium is currently carrying data of another agent and is thereby in use;

waiting a first predetermined time after sensing that said medium is idle and available for use;

generating a random number within a predefined range corresponding to an additional waiting time prior to transmitting data on said communication medium to said receiving agent;

transmitting a first signal to said receiving agent coupled to said communication medium;

receiving a signal transmitted from said receiving agent to said sending agent within a second predetermined time (IFG) after said sending agent has transmitted said first signal;

transmitting data to said receiving agent within said IFG time after receiving said signal from said receiving agent;

whereby data is transferred between said sending and receiving agents coupled to said communication medium.

20. The method as defined by claim 19, further including the step of producing a collision signal in the absence of the receipt of said signal from said receiving agent within said IFG time.

21. The method as defined by claim 20 wherein said step of generating a random number (R) for said additional waiting time includes the calculation:

$$R = rAL$$

where:

r=a random number within said predetermined range;

A denotes a logical AND operation;

L=a local variable representative of the collision and deferral history of said sending agent for said data to be sent.

22. The method as defined by claim 21, wherein said random number generating step includes:

a global mask variable G which is representative of the collision and deferral history of said sending agent for all previous data transferred, said variable G being adjusted prior to the initiation of each new data transfer.

23. The method as defined by claim 22, wherein G is comprised of 4 bits and variables C and D are defined wherein C is composed of bits representative of the number of collisions assumed for prior attempts to transmit said data, and D is comprised of bits representative of the number of deferrals for prior attempts to send said data.

24. The method as defined by claim 23, wherein said variable G is adjusted such that if the number of bits in C equal to 1 is greater than 2, then:

(a) the bits comprising G are shifted 1 bit toward the most significant bit (MSB);

(b) set $G_0=1$;

(c) set all C bits=0.

25. The method as defined by claim 24, wherein G is adjusted such that if the number of bits set equal to 1 in D is less than 2 then:

(a) the bits of G are shifted one bit toward the least significant bit (LSB);

(b) set $G_3=0$;

(c) set D=255.

26. The method as defined by claim 25, wherein in the event said collision signal is generated:

(a) $C_0=1$ (LSB of C);

(b) bits comprising L are shifted 1 bit toward MSB;

(c) $L_0=1$

(d) $N_C=N_C+1$,

where N_C= number of collisions for prior attempts to transmit said data.

27. The method as defined by claim 26, wherein in the event of a deferral, D is adjusted such that:

(a) $D_0=1$;

(b) $L_0=1$;

(c) $N_D=N_D+1$,

where N_D=number of deferrals for prior attempts to transmit said data.

28. The method for transferring data as defined by claim 20, wherein said sensing step includes detecting a synchronization pulse on said communication medium, said synchronization pulse preceeding other signals transmitted by agents coupled to said bus.

29. The method for transferring data as defined by claim 28, wherein said synchronization pulse preceeds said other signals by at least 2 bit times.

30. The method for transferring data as defined by claim 28, wherein said first signal comprises a RTS frame including at least one synchronization flag byte having a predefined bit sequence and a type field containing an RTS byte.

31. The method for transferring data as defined by claim 30, wherein said sensing step senses said flag byte, the sensing of said flag byte denoting that said medium is in use.

32. The method as defined by claim 31. wherein said signal received from said receiving agent comprises a CTS frame including at least one synchronization flag byte and a type field containing a CTS byte.

33. The method as defined by claim 32, wherein said data transmitted by said sending agent is in the form of a data frame including a plurality of data bytes preceeded by at least one synchronization flag byte, a destination address and a source address byte.

34. The method as defined by claim 33, wherein said data frame further includes a frame check sequence (FCS) and abort sequence of bits following said plurality of data bytes.

35. The method as defined by claim 34, wherein said data IFG time is generally 200 microseconds.

36. The method as defined by claim 35, wherein said data transfers on said communication medium are separated by a third predetermined time (IDG).

37. The method as defined by claim 36, wherein said signals are transmitted on said communication medium using FM-0 encoding.

38. A data processing device coupled to a communications medium having a number of other devices coupled thereto, the devices each responding to an address, the medium for transferring data between the device and one of the other devices wherein the device is addressable by the other devices coupled to the medium in response to a unique self assigned address, the self assigned address determined by the device transmitting a first tentative address on the medium and, if no other device responds, assigning the tentative address as its address, if another device does respond transmitting other tentative addresses until one is not responded to and assigning that tentative address as its address.

* * * * *

“APPLE_PAT_4_661_902_16” 244 KB 2000-02-21 dpi: 300h x 300v pix: 1913h x 2805v

United States Patent [19]

Sidhu et al.

[11] **Patent Number:** **4,689,786**

[45] **Date of Patent:** **Aug. 25, 1987**

[54] **LOCAL AREA NETWORK WITH SELF ASSIGNED ADDRESS METHOD**

[75] Inventors: **Gursharan S. Sidhu**, Menlo Park; **Alan B. Oppenheimer**, Cupertino; **Lawrence A. Kenyon, Jr.**, Sunnyvale; **Ronald R. Hochsprung**, Saratoga, all of Calif.

[73] Assignee: **Apple Computer, Inc.**, Cupertino, Calif.

[21] Appl. No.: **715,066**

[22] Filed: **Mar. 21, 1985**

[51] **Int. Cl.**⁴ **H04J 3/24**

[52] **U.S. Cl.** **370/94**; 370/85; 370/92

[58] **Field of Search** 370/85, 86, 89, 94, 370/92, 60, 95; 340/825.5, 825.52

[56] **References Cited**

U.S. PATENT DOCUMENTS

3,787,627	1/1974	Abramson et al.	370/89
4,430,651	2/1984	Bryant et al.	340/825.52
4,602,366	7/1986	Takumi	370/85
4,626,846	12/1986	Parker et al.	340/825.52

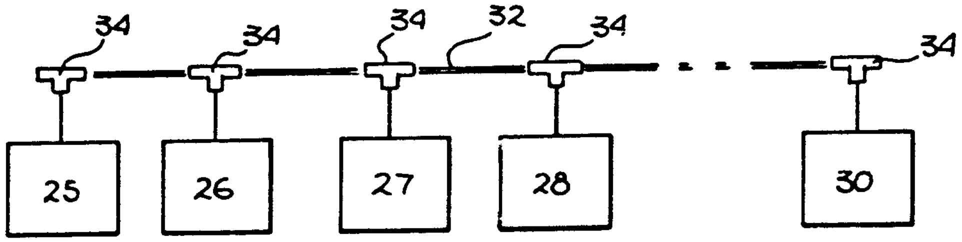
Primary Examiner—Douglas W. Olms

Attorney, Agent, or Firm—Blakely Sokoloff Taylor & Zafman

[57] **ABSTRACT**

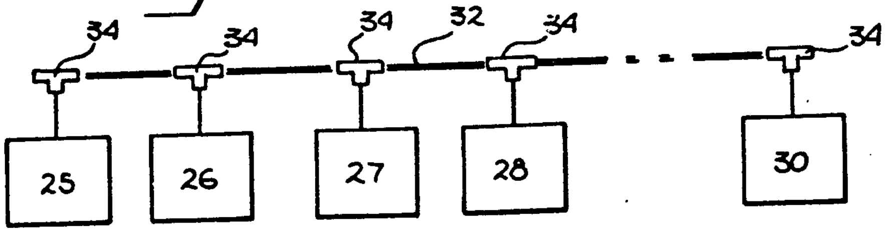
A local area network is disclosed including apparatus and methods for transferring data between a plurality of data processing resources ("agents") coupled to a cable. In the preferred embodiment, a plurality of agents are coupled to a common cable for data transmission and reception. An agent newly coupled to the cable dynamically assigns itself a unique address on the cable to which other agents may send data. Once an agent has been assigned a final address, it may then transmit to, and receive data from, other agents on the cable. An agent desiring to send data to a receiving agent follows a three step handshake, wherein the sending agent transmits an "RTS" signal and within a predetermined time must receive a "CTS" signal from the receiving agent. The sending agent then transmits a data frame within a predetermined time after the CTS signal is received. The failure to detect a return CTS signal within the predetermined time denotes a collision condition. Retransmission is attempted using a linear back off method which is adjusted based on previous cable traffic history.

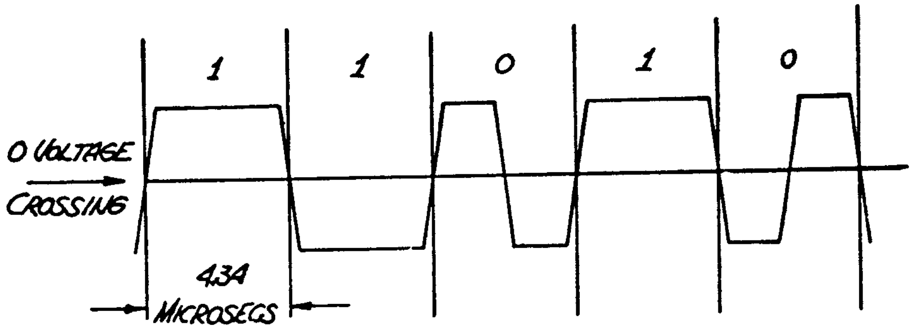
19 Claims, 15 Drawing Figures



AppleTalk Network

Fig. 1





FM-O ENCODING

Fig. 2

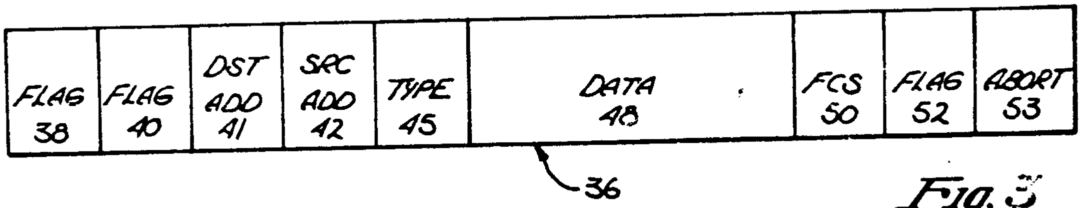


Fig. 3

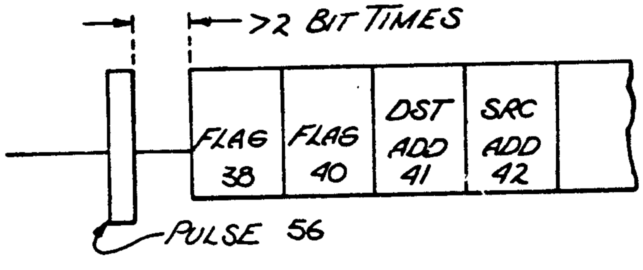


Fig. 4

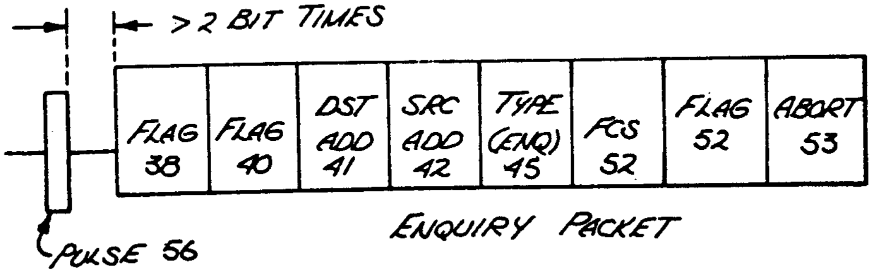


Fig. 5

ADDRESS ASSIGNMENT

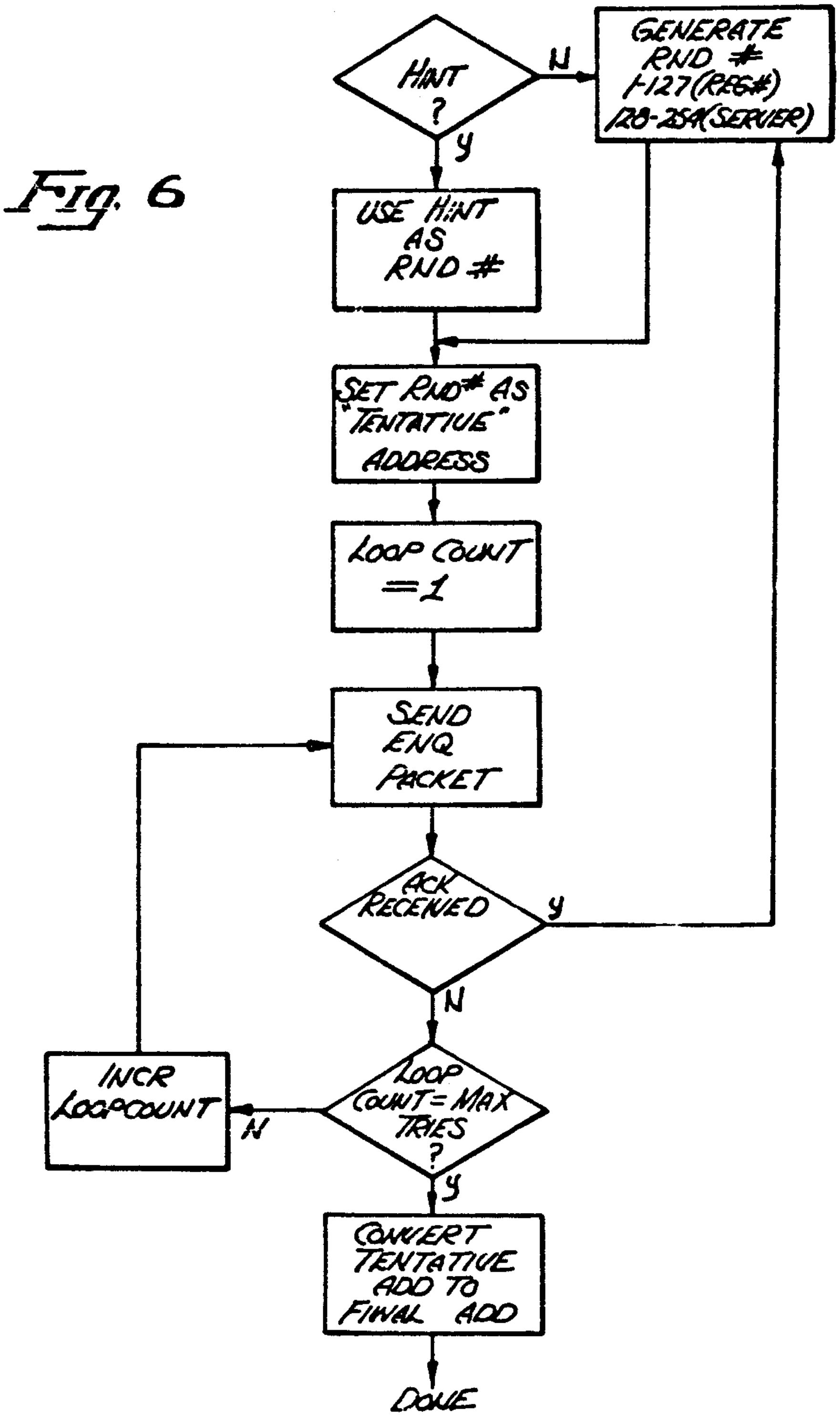


Fig. 6

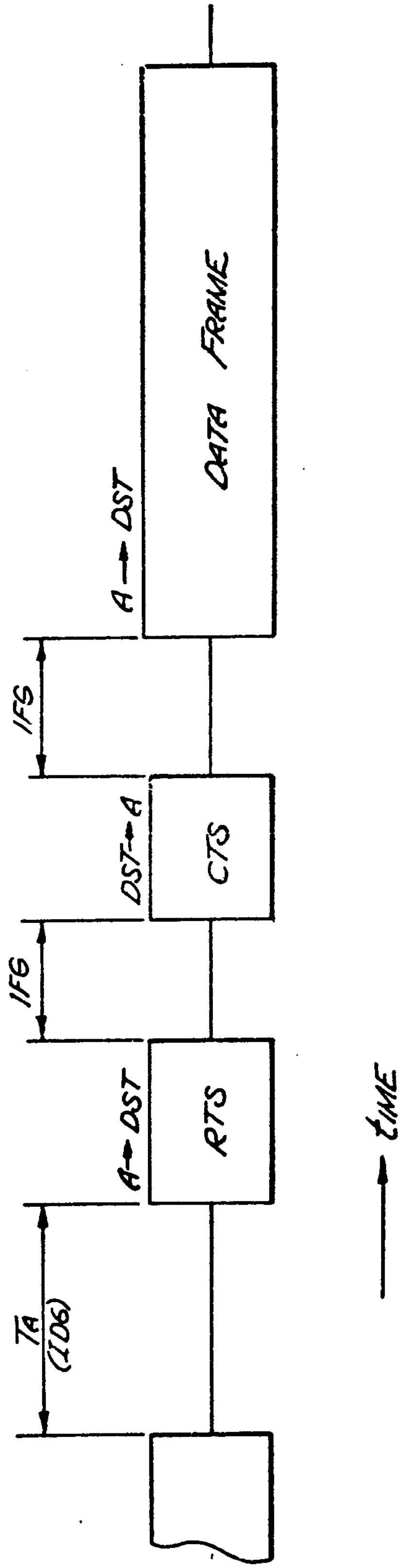


Fig. 7

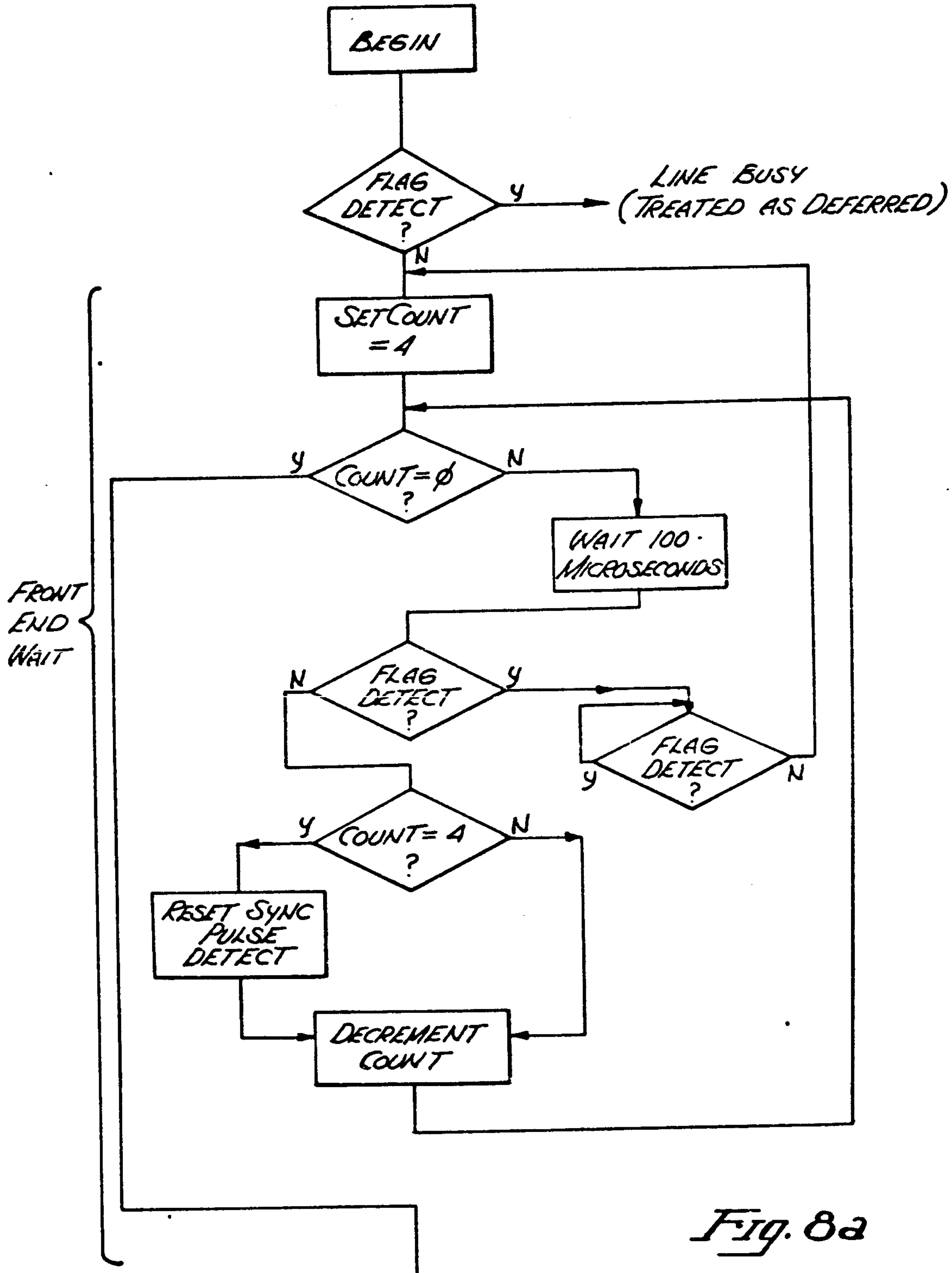


Fig. 8a

CONTINUED . FIG 8b

U.S. Patent Aug. 25, 1987 Sheet 5 of 8 4,689,786

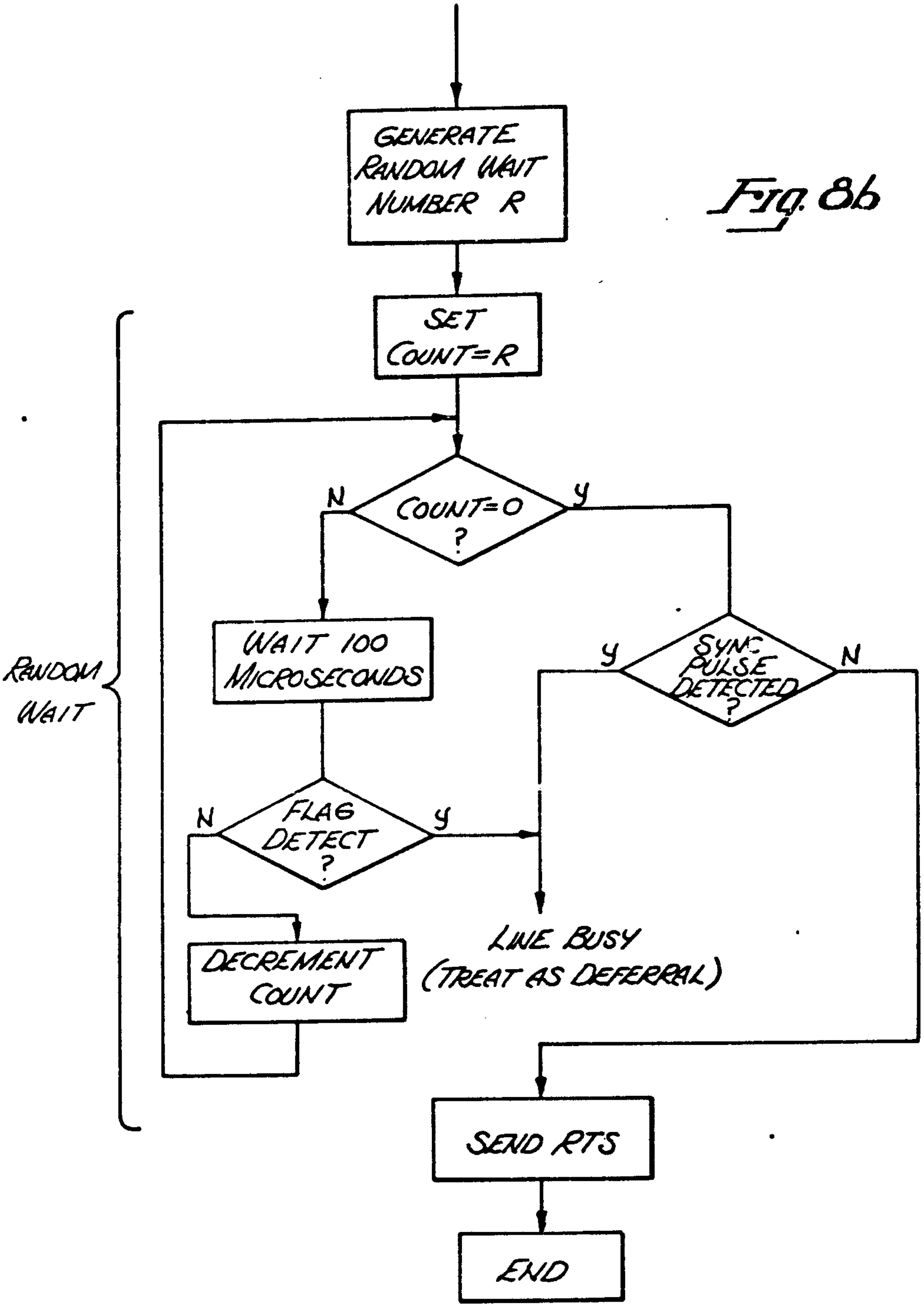
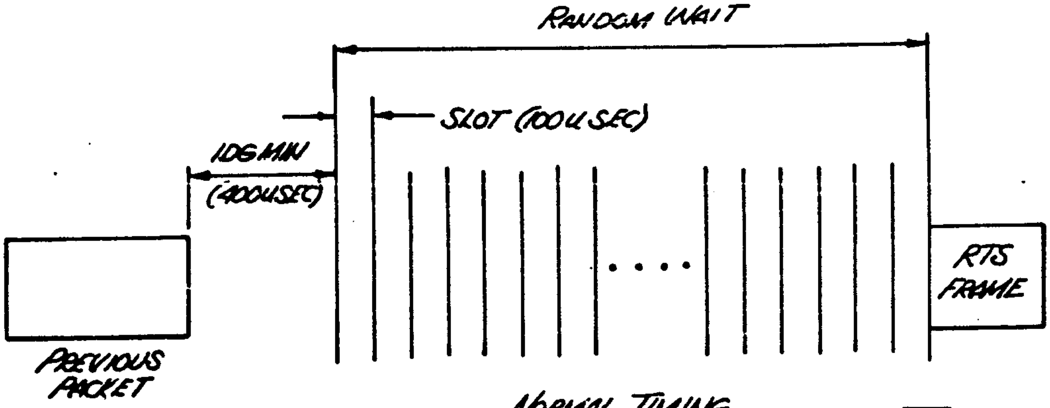


Fig. 8b

"APPLE_PAT_4_689_786_06" 78 KB 2000-02-22 dpi: 300h x 300v pix: 1757h x 2403v



NORMAL TIMING

Fig. 9

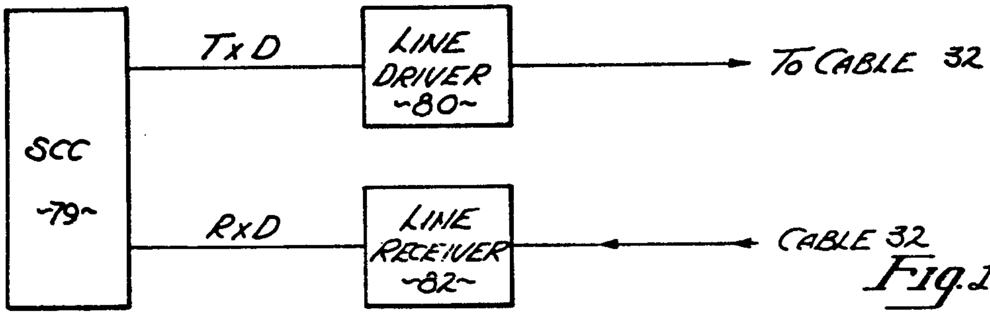
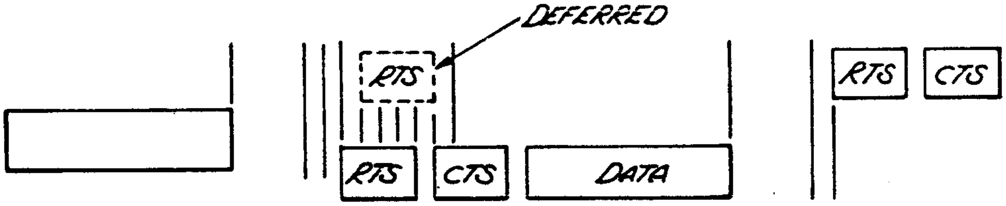
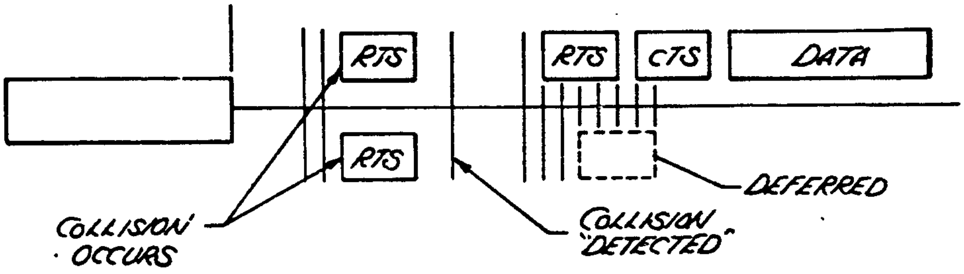


Fig. 10



DEFERENCE (W/TWO TRANSMITTERS)

Fig. 11



COLLISION (AND RESOLUTION)

Fig. 12

"APPLE_PAT_4_689_786_07" 112 KB 2000-02-22 dpi: 300h x 300v pix: 1806h x 2650v

TRANSMIT OPERATION

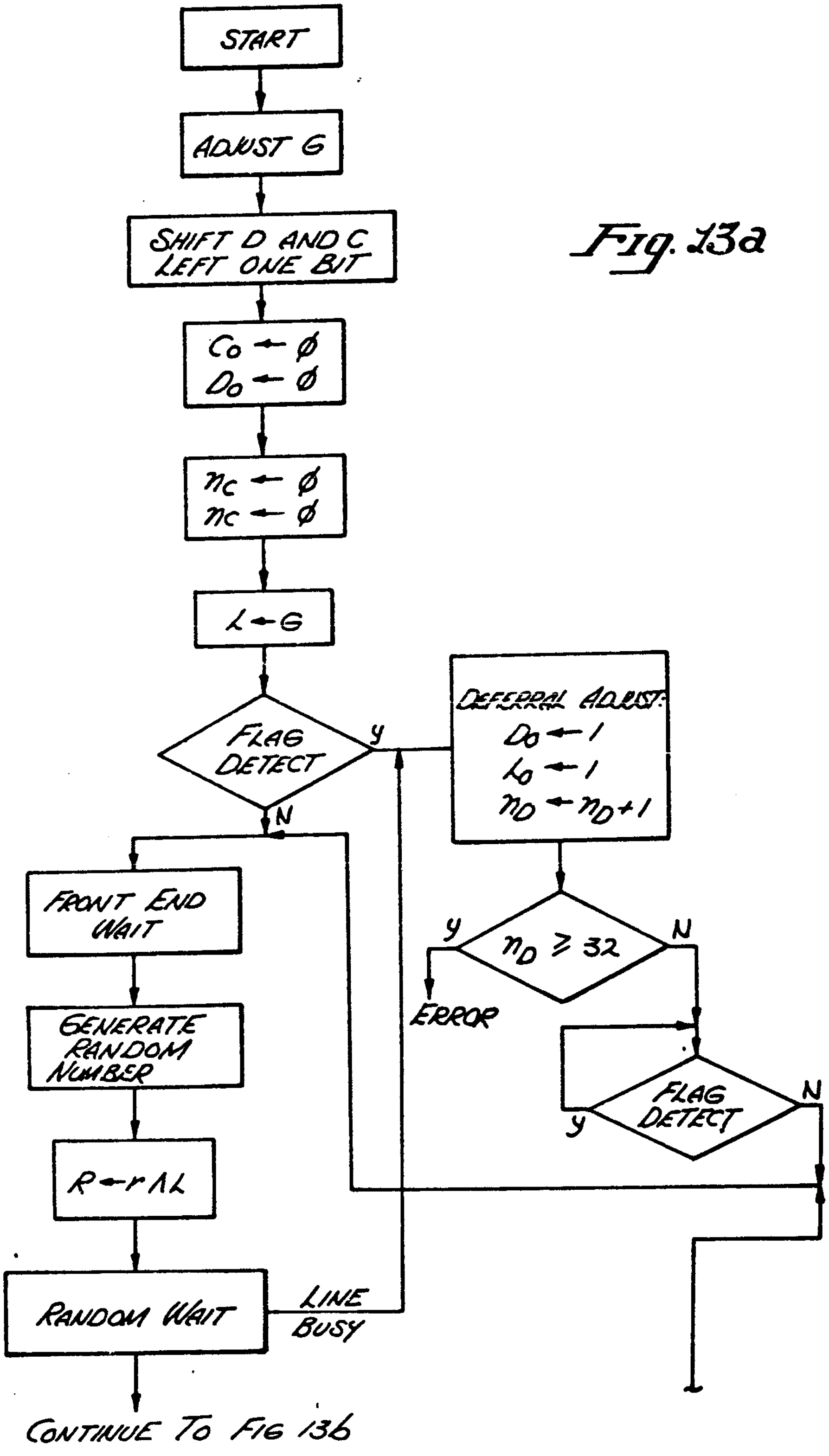


Fig. 13a

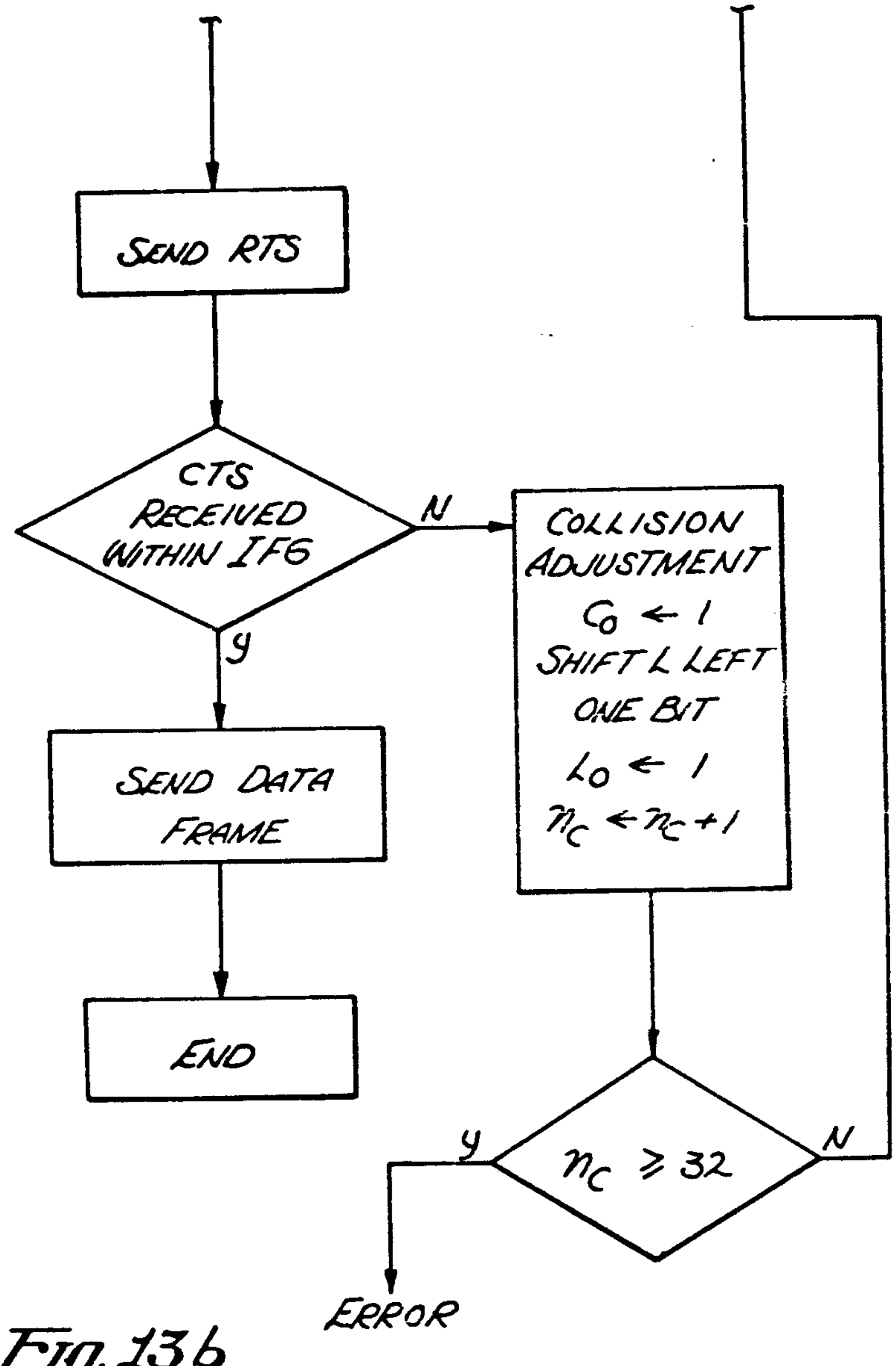


Fig. 13b

“APPLE_PAT_4_689_786_09” 63 KB 2000-02-22 dpi: 300h x 300v pix: 1769h x 2250v

LOCAL AREA NETWORK WITH SELF ASSIGNED ADDRESS METHOD

The present application has been filed concurrently with, and is related to, U.S. patent application, Ser. No. 06/715,065, filed Mar. 21, 1985, and hereby refers to, and incorporates by reference the contents of the above-referenced application.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to apparatus and methods for transferring data between a source and a plurality of receiving data processing devices. More particularly, the present invention relates to data transfer along a local area network between a plurality of data processing and peripheral devices.

2. Art Background

In the computing industry, it is quite common to transfer data and commands between a plurality of data processing devices, such as for example, computers, printers, memories and the like. The interconnection of computers and other peripheral devices principally developed in the early 1970's with the advent of computer networking systems, which permitted the distribution of access to computing resources beyond the immediate proximity of a main frame computer.

Networks, such as the ARPA Network, were developed to provide access by various users to large time-sharing systems and the transfer of data between such systems. In the case of geographically local networks, so called "local area networks" (LANs) were developed to connect together a collection of computers, terminals and peripherals located, typically in the same building or adjacent buildings, and permitted each of these devices to communicate among themselves or with devices attached to other networks. Local area networks permit the implementation of distributed computing. In other words, some of the devices coupled to the local area network may be dedicated to perform specific functions, such as file storage, data base management, terminal handling, and so on. By having different machines perform different tasks, distributed computing can make the implementation of the system simpler and more efficient.

Local area networks differ from their long-haul cousins in a number of respects. A key difference is that the designers of long-haul networks, such as the ARPA network, are often forced by economic or legal reasons to use the public telephone network, regardless of its technical suitability. In contrast, most local area networks utilize their own high-bandwidth cable to permit datagram service between the various devices coupled to the LAN. The most common transmission media for carrier sense local area networks are co-axial cable, twisted pair and fiber optics. A variety of cable topologies are possible, such as linear, spine, tree, ring and segmented. In addition, local area networks do not suffer from the long propagation delays which are inherent with other large networks, thus allowing the channel utilization to be pushed significantly above the capabilities of large scale networks.

Although local area networks hold the promise of distributed processing and communication between data processing devices, a number of factors have prevented wider use and acceptance of local area networks, such as ETHERNET (U.S. Pat. No. 4,063,220).

For example, despite efforts to lower costs using VLSI technology, a typical LAN node may represent a significant percentage of the total cost of a personal computer. Accordingly, in the personal computer market local area networks have been prohibitively expensive to implement. In addition, most local area networks utilize complex cabling techniques and require a system administrator who is trained in the installation, updating and maintainence of the LAN system. Moreover, many local area networks utilize relatively complex protocols to permit the various devices coupled to the LAN to communicate under various conditions.

As will be decribed, the present invention provides a local area network for communication and resource sharing among various computers, servers, disks, printers, modems and other data processing devices. The present invention supports a wide variety of local area network services, and permits communication to larger networks through the use of bridging devices. The present invention provides an economical, reliable, and mechanically simple local area network heretofore unknown in the prior art.

SUMMARY OF THE INVENTION

A local area network is disclosed including apparatus and methods for transferring data between a plurality of data processing resources ("agents") coupled to a cable. In the preferred embodiment, a plurality of agents are coupled to a common cable for data transmission and reception. An agent newly coupled to the cable dynamically assigns itself a unique address on the cable to which other agents may send data. The agent generates a random number within a predetermined range, or retrieves a previously stored initial number ("hint"), for use as a tentative address. The agent transmits an enquiry signal (ENQ) over the cable [illegible]e tentative address to determine if the tentative address is currently being used by another agent. If an acknowledge (ACK) signal is received by the sending agent in response to the ENQ signal, another random number is generated as a tentative address and additional ENQ signals are sent. In the event no ACK signal is received, the sending agent assigns the tentative address as a final address in its memory.

Once an agent has assigned itself a final address, it may then transmit to, and receive data from, other agents on the cable. An agent desiring to send data to a receiving agent senses the cable to determine if the cable is idle or in use. If the cable is in use, the agent "defers" until an idle condition is sensed. Once the cable is detected as idle, the sending agent waits a predetermined period plus a random time before transmitting an "RTS" signal to the receiving agent. The sending agent then monitors the cable for a "CTS" signal, which must be transmitted by the receiving agent to the sending agent within a predetermined time (IFG) after the receipt of the RTS signals. If a CTS signal is properly received, the sending agent may then transmit a data frame to the receiving agent within an IFG time after receipt of the CTS signal. The failure to detect a return CTS signal within an IFG time period denotes a collision condition. If a collision is presumed, the present invention attempts to re-transmit an RTS signal using a backoff method which dynamically adjusts the period before a re-transmission attempt based on recent cable traffic history. Accordingly, the present invention provides a method of minimizing collisions and permits reliable and economical data transfers between a plurality of agents coupled to the common cable.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 illustrates a local area network adapted to utilize the teachings of the, present invention.

FIG. 2 is a timing diagram illustrating the present invention's use of frequency modulated (FM-0) encoding.

FIG. 3 illustrates the frame format utilized by the present invention to transfer data to various data processing devices coupled to the local area network.

FIG. 4 illustrates the present invention's use of a synchronization pulse prior to the transmission of a frame.

FIG. 5 illustrates an enquiry (ENQ) frame utilized by the present invention during dynamic address assignment.

FIG. 6 is a flow chart illustrating the sequence of operations utilized by a data processing device coupled to the present invention during dynamic address assignment.

FIG. 7 diagrammatically illustrates the present invention's use of handshake signals between sending and receiving data processing devices prior to the transmission of a data frame.

FIGS. 8(*a*) and 8(*b*) are a flow chart illustrating the sequence of operations of a sending device to obtain cable access.

FIG. 9 is a diagrammatical illustration of the transmission of an "RTS" frame by a sending device after sensing an idle cable.

FIG. 10 is a block diagram illustrating the present invention's use of a serial controller device coupled to the local area network.

FIG. 11 illustrates the present invention's collision avoidance method including deference.

FIG. 12 illustrates the collision and resolution mechanism of the present invention wherein two "RTS" signals collide along the local area network.

FIGS. 13(*a*) and 13(*b*) are a flow chart illustrating the generation of the random wait period R.

DETAILED DESCRIPTION OF THE INVENTION

A local area network including apparatus and methods for transferring data between a plurality of data processing resources coupled to a common cable is disclosed. In the following description for purposes of explanation, specific numbers, bytes, registers, addresses, times, signals, and formats, etc. are set forth in order to provide a thorough understanding of the present invention. However, it will be apparent to one skilled in the art that the present invention may be practiced without these specific details. In other instances, well known circuits and devices are shown in block diagram form in order not to obscure the present invention unnecessarily.

Referring to FIG. 1, the present invention may include a plurality of data processing devices identified generally by the numerals 25 through 28 as well as peripheral devices such as printer 30 (or other devices such as, for example, a global memory, a disk drive and the like). For purposes of this Specification, all data processing and peripheral devices which are coupled to the present invention's local area network are collectively referred to as "agents". As shown, processing devices 25, 26, 27, 28 and printer 30 are interconnected

for data transfer to one another by a common cable 32. The various devices are coupled to cable 32 by connection modules 34, which in the presently preferred embodiment, contain a passive coupling transformer, resistive and capacitive circuits, and are known in the art for coupling each of the data processing and other devices to cable 32. Cable 32 is appropriately terminated to eliminate signal reflections. In the preferred embodiment, cable 32 is terminated with 100 Ohm resistors, and is comprised of a twisted pair cable. It will be appreciated by one skilled in the art that cable 32 may comprise any shared media, such as coaxial cable, fiber optics, radio channel and the like. Since, in the present embodiment, the agents are passively coupled to cable 32, a failure of an agent or a connection module will not disrupt communication over cable 32.

As will be described, the present invention provides a local area network permitting synchronous serial communication and data transfer between data processing devices 25 through 28, and other peripheral devices such as printer 30, utilizing protocols and collision avoidance and detection methods and apparatus. The present invention's architecture and protocols minimize handshake and collision detection complexities common in prior art systems, and permit high speed serial communication along cable 32. The present invention permits access to various resources coupled to the network, such as data stored in local memories, or disk, and the common use of global printers, without the necessity of complex or active termination circuitry at the cable end, or the need to predefine addresses for each device coupled to cable 32. In the current embodiment, the present invention operates at approximately 230 kilobits per second through a shielded, twisted pair cable 32, and is driven in accordance with EIA standard RS0-422 balanced voltage specifications.

Referring to FIG. 2, data is encoded and transmitted on cable 32 using a self-clocking technique known as FM-0 (bi-phase space), such that each bit cell, typically 4.34 microseconds in duration, contains a state transition at its end, thereby providing necessary timing information to the receiver. As illustrated, zeros are encoded by adding an additional zero-crossing transition at mid-cell, such that two zero crossings are detected for each 4.34 microsecond cell. Similarly, a logical one is provided in a particular cell by a zero-crossing transition only at its end. Accordingly, through the use of FM-0 encoding, clocking information is carried by the data signal itself, and permits the present invention to operate in a synchronous mode.

Referring now to FIG. 3, the present invention utilizes a basic unit of data transmission known as a "frame" 36. Frame 36 includes a preamble consisting of two or more synchronization ("flag") bytes 38 and 40. In the presently preferred embodiment, each synchronization byte comprises the bits 01111110. As is known, synchronization bytes 38 and 40 permit receiving data processing units coupled to cable 32 to synchronize their receiving circuits, and to receive necessary clock information (through the use of FM-0 encoding). Following synchronization bytes 38 and 40 is an eight-bit destination address 41 which specifies the address of the data processing agent for which the frame is intended. A source address 42 comprises an eight-bit address of the data processing agent transmitting the frame of information. A "type" field 45 is provided to specify the type of frame which is transmitted through the use of various codes. For example, type field 45 may designate

"APPLE_PAT_4_689_786_11" 336 KB 2000-02-22 dpi: 300h x 300v pix: 1947h x 2915v

an acknowledge (ACK) frame, an enquiry (ENQ) frame, as well as an RTS and CTS frame, which will be described more fully below. The type field is followed by a multi-byte data field (possibly of zero length) which may contain raw data, messages, and the like to be transmitted between the agents coupled to cable 32. Following the data field 48 is a 16-bit frame check sequence which is computed as a function of the contents of the source address, destination address, type and data fields. In the present embodiment, the frame check sequence (FCS) is defined using the standard CRC-CCITT polynomial. The frame check sequence 50 is followed by an eight-bit synchronization ("flag") trailer 52 (consisting of the logical bits 01111110), and an abort sequence 53 which consists of eleven or more ones in a row. Abort sequence 53 is used to delineate the end of the frame 36 to agents coupled to cable 32. Frame 36, as will be described, is transmitted along line 32 in a serial synchronous fashion using a handshake sequence of control frames which are in turn followed by the data frame 36 illustrated in FIG. 3.

As shown in FIG. 4, prior to the transmission of a frame, a transmitting agent coupled to cable 32 transmits a synchronization pulse 56 which, is followed by an idle period greater than two bit times and less than 10 bit times. Pulse 56 may comprise any signal containing a zero crossing. In the present embodiment, as illustrated in FIG. 10, each agent coupled to cable 32 utilizes a Zilog Z8530 SCC serial communication controller chip 79, accessing cable 32 through a line driver 80 and a line receiver 82. (See, Zilog Technical Manual, Z8030/Z8530 SCC Serial Communications Controller, January, 1983.) The Z8530 SCC device 79 includes circuitry which searches for synchronization (flag) bits when in a "hunt" mode. As previously discussed, the present invention utilizes a synchronization (flag) byte having the bit states 01111110. In addition, the serial communication controller chip has the capability of detecting a missing clock cycle, and setting a missing clock bit within the device if following a given zero crossing, a predefined period (greater than 1 bit time) elapses without a successive zero crossing of the incoming signal R_xD.

The pulse 56 provided by a transmitting agent coupled to cable 32 will be taken as a clock by all receiving agents. However, since it is followed by an idle period greater than two bit times, a missing clock is detected and the missing clock bit is set in the SCC device 79 of each agent connected to cable 32, thereby notifying agents that cable 32 is in use. In the preferred embodiment, the synchronization pulse 56 is obtained by momentarily enabling the line driver 80 for at least one bit time. This causes transmission, for the pulse duration time, of the signal T_xD onto cable 32, thus ensuring at least one zero crossing in the synchronization pulse 56. In addition, the detection of synchronization (flag) bits (i.e., 38 and 40) clears the "hunt" bit in the Z8530 device, and permits each agent coupled to cable 32 to more efficiently detect whether or not cable 32 is currently in use prior to the transmission of a frame, as well as providing the necessary synchronization bits to allow the receiving agent to clock itself to the incoming data frame. It will be appreciated that although the present invention currently uses a Z8530 SCC device for detecting missing clock cycles and synchronization bytes, that other circuitry may equally be used for the same function.

Each agent coupled to cable 32 is identified by a unique binary address along the cable. One feature of the present invention is that an agent coupled to cable 32 does not require a predefined permanent address. Thus, for example, device 27 may be removed from cable 32 and then recoupled to another cable at a different location without need to configure an address. When an agent is newly coupled to cable 32, a unique protocol is followed such that an address is dynamically generated and assigned by the agent itself. In the presently preferred embodiment, the address of each agent is identified using an eight bit identifier (where no agent may have a zero address or an address of 255).

Referring briefly to FIG. 6, the sequence of operations which an agent utilizes in order to determine and assign itself an address is illustrated. It is apparent that to prevent disrupted service, no agent may acquire the same address as an already functioning agent. In practice, the address of agents may be allocated between general data processing devices and "servers" which may comprise main frame or other machines. In the present embodiment, addresses 1 through 127 are allocated for general purpose agents, and addresses 128 through 254 are allocated for use by servers. As shown in FIG. 6, upon being coupled to cable 32, each agent either generates an arbitrary random number within a predefined range or obtains a starting number from some long term, non-volatile memory (for example, read-only-memory or magnetic medium) referred to as a "hint". This random number (or "hint") is treated as a "tentative" address, and the agent then transmits an enquiry (ENQ) frame which utilizes the tentative address as a destination address. The enquiry frame transmitted is of the form illustrated in FIG. 5, and includes an initial pulse 56 separated by at least two bit times prior to the synchronization (flag) bytes 38 and 40, previously described with reference to FIG. 3. The destination address 41 of FIG. 5, as well as the source address 42, contains the tentative address generated randomly or through the hint. It will be noted that the type field 45 in FIG. 5 contains a binary code which identifies the frame of FIG. 5 as an "enquiry" (ENQ) frame for use in address assignment. This ENQ frame is transmitted over cable 32. In the event another agent has previously been assigned the tentative address, the agent already using the tentative address receives the ENQ frame, and in response transmits an acknowledge frame (ACK) back to the transmitting agent. In practice, the ACK frame is structured similarly to the ENQ frame disclosed in FIG. 5, except that the type byte contains a binary code identifying the packet as an ACK.

As illustrated in FIG. 6, in the event that an ACK frame is received by the transmitting agent, that agent must then generate another random number as a tentative address and repeat the transmission of this new tentative address along cable 32. In the event that no ACK frame is received, the agent newly coupled to the cable continues to send ENQ frames onto the cable until some predefined maximum number of tries has occurred. If, after a predefined number of attempts, no ACK frame has been received, the transmitting agent then assigns the tentative address as its final address for all future communication along cable 32. The repeated transmission of ENQ frames is used to avoid instances where a particular agent which may be using the tentative address may currently be busy, and thus miss the reception of an enquiry.

Once an agent has been assigned a final address, it may then communicate with other agents coupled to cable 32 utilizing a handshake protocol and collision avoidance mechanism described below. Referring to FIGS. 6, 8(*a*), 8(*b*) and 9, communication between agents coupled to cable 32 occurs through a three-way handshake process. The purpose of the handshake sequence is to control the access to the shared cable 32 in an orderly fashion that reduces the probability of a collision. Each transmission including the handshake (known as a "dialogue") must be separated by a minimum inter-dialogue gap (IDG), which in the present embodiment comprises 400 microseconds. In addition, the frames within a single transmission (dialogue) must follow one another within a maximum interframe gap (IFG) of, in the current embodiment, 200 microseconds. A collision is said to occur when two or more agents transmit at the same time on cable 32.

Referring to FIGS. 7 and 8(*a*) and 8(*b*), the sending agent, for example data processing agent 25, which desires to communicate with another agent coupled to cable 32, executes the operations set forth in the flow chart of FIGS. 8(*a*) and 8(*b*). A sending agent prior to transmission determines whether or not the "hunt" bit in the Z 8530 SCC serial controller, or other appropriate hardware, has detected a synchronization (flag) byte passing along cable 32. If a synchronization (flag) byte has been detected, and no abort byte has followed, then cable 32 is currently in use and the agent wishing to transmit "defers" its transmission. In the event that no synchronization pulse 56 or synchronization (flag) bytes (38 and 40) are detected, the agent desiring to transmit data executes a front end wait operation, as best illustrated in FIG. 8(*a*). The front end wait operation consists of a series of four waiting periods, in the present embodiment of 100 microseconds each, following each of which the flag detect ("hunt bit") is checked to see if a synchronization (flag) byte has been received on the cable 32. The detection of a flag byte denotes that some other agent is using the cable 32. In such event, the sending agent must wait for the flag detect (hunt bit) to clear, thus signalling the end of utilization of cable 32. At this point, the entire front end wait sequence illustrated in FIGS. 8(*a*) and 8(*b*) is repeated.

If, on the other hand, a flag byte is not detected, this indicates that during the front end wait sequence no other agent has empted to use the cable, and a random wait operation is then executed. In addition, during the front end wait operation, the synchronization pulse detect is cleared after the first 100 microsecond wait.

Before proceeding to execute the random wait operation, illustrated in FIG. 8(*b*), a random wait number R is generated (the details regarding the generation of R will be discussed below). As shown, the random wait operation cycles R times through a basic operation of waiting 100 microseconds before checking to see if flag has been detected (hunt bit cleared). If, at any point a flag is detected, then another agent is using the cable 32 and the sending agent must defer its transmission. If, however, at the end of the random wait sequence the cable is still quiet (not in use) then one last check is made to see if a synchronization pulse has been detected, before sending an RTS frame, as will be described.

If the cable 32 remains idle throughout this randomly generated waiting period R, the transmitting agent proceeds to transmit a synchronization pulse 56 followed by an "RTS" frame along cable 32 to the receiving agent. An RTS frame is structured substantially the same as the ENQ frame illustrated in FIG. 5, however, the type field contains a binary code identifying the frame as an RTS rather than an ENQ frame. The receiving agent, upon receiving the RTS frame from the transmitting agent, transmit a "CTS" frame back to the orginal transmitting agent within the maximum interframe gap (IFG) period. As in the case of the RTS frame, a CTS frame transmitted by a receiving agent is structured substantially the same as the ENQ packet illustrated in FIG. 5, except that the type field contains a code identifying the frame as CTS. Once the original transmitting agent, for example data processing unit 25, receives the CTS frame, a full data frame 36, as illustrated in FIG. 3, is transmitted to the receiving agent within one IFG of the receipt of the CTS frame. In the event that the transmission of a CTS or data frame does not occur within an IFG, then the transmitting agent assumes that a collision has occurred or the destination agent is inactive or otherwise unavailable.

If a general broadcast to all agents coupled to cable 32 is desired, the transmitting agent sends an RTS frame with a destination address of 255 to all agents on the line, and waits for an IFG period to elapse prior to sending a data frame 36 also having a destination address of 255. Accordingly, in the case of general broadcasts along cable 32, the transmitting agent does not wait for return CTS frames, but rather, immediately proceeds to conduct a general broadcast once the IFG period has elapsed after transmitting an RTS frame. In addition, by providing within the RTS frame a destination address field 41 having a particular value (255) corresponding to a broadcast address, only one RTS frame need be transmitted to all agents at the various addresses along cable 32.

It will be appreciated by one skilled in the art that the purpose of the three step handshake prototol described above is to avoid collisions by restricting the periods in which collisions are highly likely (typically during the RTS and CTS frame exchanges), and to spread out in time the cable access of transmitters waiting for the cable 32 to become idle prior to the beginning of a transmission. A successful RTS-CTS frame exchange signifies that a collision did not occur, and that all agents desiring to transmit have sensed the coming data frame transmission, and are waiting until the data exchange is complete prior to attempting to gain control of the cable.

In the event that another agent begins a transmission during the RTS-CTS frame exchange described above, it will be appreciated that the CTS frame will not be properly received (e.g. the frame check sequence is invalid), and that the sending agent may then assume that a collision has occurred. A collision will prevent a complete RTS and CTS frame exchange, and thereby prevent a proper handshake from occurring. Normally, if an agent desiring to transmit data on cable 32 senses that the cable is currently in use, it defers the transmission of its own RTS until the cable is idle (see FIGS. 11 and 12).

Referring now to FIGS. 9, 13(*a*) and 13 (*b*), the sequence of operations executed by the present invention to obtain the value of random wait number R (as previously discussed with reference to FIG. 8) will be described in detail. As will be appreciated, the present invention dynamically modifies the random wait number R in response to recent cable traffic history. The method utilized by the present invention presumes that if collisions have been assumed for recently sent data frames, the cable 32 is currently the subject of heavy loading and high bus contention. A random waiting period R prior to a retransmission attempt spreads out in time bus access for the various agents contending for cable use. Accordingly, the operations illustrated in FIGS. 13(*a*) and (*b*) are executed to generate and adjust the random wait number R utilized in accordance with the sequence of operations set forth in FIGS. 8(*a*) and (*b*). In the present invention, eight bit shift registers are provided in order to keep track of collision and deferral histories for each agent coupled to cable 32. For purposes of this Specification, the variable "C" denotes an eight bit shift register which is utilized to keep track of the collision history for the last eight data messages which an agent has attempted to send, and a variable "D" which denotes an eight bit shift register representing the deferral history for the last eight messages which have been attempted to be sent. As previously discussed, a collision is presumed if the RTS-CTS frame handshake protocol fails to occur within the IFG period, and a deferral is deemed to occur if an agent, prior to transmitting a message, detects a flag byte or synchronization pulse 56 thereby indicating that the cable is in use. A variable "G" is defined as a four bit global mask which represents a number signifying a modification factor representative of all previous messages which the the agent has attempted to transmit. A variable "L" is defined as a local mask which is representative of attempts to transmit the current message by an agent coupled to cable 32. In addition, N_C is defined as the number of collisions which have been assumed for a particular data packet, and N_D is defined as the number of deferrals which have occurred prior to the transmission of the current data packet.

As illustrated best in FIGS. 13(*a*) and (*b*) prior to the transmission of a new data packet, the variable G is adjusted, as follows:

If the number of bits sets (i.e. equal to 1) in the eight bit register "C" is greater than 2, then all bits in the four bit shift register defining "G" are shifted to the left [least significant bit (LSB) toward most significant bit (MSB)] one bit. In addition, G_0 (the least significant bit of four bit shift register G) is set to 1 and the eight bits comprising C are set to 0.

If the number of bits set in the eight bit register "C " is less than or equal to 2, then D is examined, and if the number of bits set in "D" is less than, 2, then the present invention shifts the contents of G right (MSB toward LSB) one bit; sets G_3 (MSB of G) equal to 0 and sets the value of D equal to 255.

Once G has been adjusted, the present invention then shifts the contents of registers D and C left one bit (toward MSB) and sets the least significant bit (LSB) of C and D equal to 0. Similarly, variables N_C and N_D which denote the number of collisions and deferrals for the particular message to be sent are also set equal to 0. Moreover, as shown in FIG. 13, the value of L is then set equal to the value of G.

Prior to beginning the front end wait sequence disclosed in FIG. 8(*a*), the present invention determines whether or not a flag detect (i.e. flag byte) has been detected along cable 32. In the event that no flag byte has been detected, the present invention then executes the sequence of operations illustrated in FIG. 8(*a*) for the front end waiting period. Subsequent to the fixed front end wait sequence, the present invention generates a random number "r" within a predetermined range, and then calculates the value of "R" by logically "AND" ing the value of r with the previously determined value of L (local mask variable). Once the value of R is determined, the present invention then follows the random wait cycle illustrated in FIG. 8(*b*), and upon completion of the random waiting period, transmits the RTS frame as shown in FIG. 13(*b*).

If the CTS frame is received by the sending agent within the IFG period, as previously discussed, then the data frame is transmitted, and the message dialogue has been completed. If, on the other hand, a flag is detected prior to the beginning of the front end wait sequence, a deferral adjustment is provided wherein D_0 (the LSB of register D) is set to 1 and L_0 is set to 1. In addition, the deferral adjustment includes the setting of N_D equal to N_D+1. The flag detect (hunt bit) is once again checked. As shown in FIG. 13, this deferral adjustment occurs in cases where it is determined that the line is busy prior to transmission.

In the event the RTS/CTS handshake is not successful, then a collision is presumed and a collision adjustment occurs. C_O is set equal to 1 and the value of L is shifted left (LSB to MSB) one bit. In addition, L_O is set equal to 1 and N_C is set equal to N_C+1, as illustrated in FIG. 13(*b*).

It has been found that the utilization of the steps illustrated in FIGS. 13 dynamically adjust the randomly generated value of r such that the period of time (in 100 microsecond increments) which an agent waits in addition to the front end waiting period prior to attempting a transmission is modified in accordance with recent cable traffic history. This modification of the random waiting period significantly increases the probability of a successful RTS/CTS frame exchange, and thereby avoids collisions along cable 32.

Accordingly, apparatus and methods have been disclosed having particular utility when used in conjunction with a local area network. The present invention provides a network which permits any agent to be coupled to the cable at any point, and assign itself a unique address. In addition, the present invention's novel collision avoidance protocol minimizes the probability of collisions occurring on the cable, and if a collision occurs, provides a greater probability of success for subsequent re-transmissions.

Although the present invention has been described particularly with reference to FIGS. 1-13, it will be apparant to one skilled in the art that the present invention has utility far exceeding that isclosed in the Figures. It is contemplated that many changes and modifications may be made, by one of ordinary skill in the art, without departing from the spirit and scope of the invention as disclosed above.

We claim:

1. An apparatus for assigning a unique address to a data processing device coupled to a communication medium to permit the transfer of data between a plurality of said agents on said medium, comprising:

transceiver means coupled to each of said agents for transmitting signals onto said medium and receiving signals transmitted on said medium by another agent;

address assignment means coupled to each of said agents to permit each of said agents to assign itself a unique address on said communication medium, said address assignment means including:

random number generating means for generating a random number within a predefined range for use as a tentative address;

address storage means coupled to said random number generating means for storing said tentative address;

first signal generation means coupled to said address storage means and said transceiver means for generating an enquiry (ENQ) signal and transmitting at least one said ENQ signal to a device having said tentative address on said communication medium;

acknowledge signal receiving means coupled to said transceiver means for receiving an acknowledge (ACK) signal transmitted by an agent other than said agent being assigned a unique address in response to its receipt of said ENQ signal, said acknowledge receiving means signalling said random number generating means to generate another random number as a tentative address upon the receipt of said ACK signal;

timing means coupled to said acknowledge signal receiving means for storing said tentative address as a final address in said address storage means in the absence of the receipt of said ACK signal within a predetermined time (IFG) after the last ENQ signal has been transmitted;

whereby said agent is assigned a unique address on said communication medium.

2. The system as defined by claim 1, wherein said first signal generation means generates and transmits a plurality of ENQ signals, each of said ENQ signals being transmitted after said IFG time has elapsed.

3. The system as defined by claim 2, wherein said timing means stores said tentative address as a final address in the absence of said ACK signal once said IFG time has elapsed after the last of said plurality of said ENQ signals have been transmitted.

4. The system as defined by claim 3, wherein said ENQ signal includes an ENQ frame having at least one synchronization flag byte having a predefined bit sequence and an ENQ byte.

5. The system as defined by claim 4, wherein said ACK signal includes an ACK frame having at least one synchronization flag byte having a predefined bit sequence and an ACK byte.

6. The system as defined by claim 5, wherein said random number generation means generates a random number in the range of 1 to 254.

7. The system as defined by claim 5, wherein said first signal generation means generates a synchronization pulse and transmits seid synchronization pulse on said communication medium, said synchronization pulse transmitted prior to other signals on said medium.

8. The system as defined by claim 7, wherein said synchronization pulse preceeds said other signals by at least 2 bit times.

9. The sytem as defined by claim 7, wherein said IFG time is generally 200 microseconds.

10. The system as defined by claim 7, wherein said communicaiton medium is a twisted pair cable.

11. The system as defined by claim 10, wherein said twisted pair cable is terminated by 100 Ohm resistors.

12. The system as defined by claim 7, wherein said signals on said communication medium are transmitted using FM-0 encoding.

13. In a communication system for transferring data between a plurality of devices, a method used by each of said agents for assigning itself a unique address on said communication system, comprising the steps of:

generating a random number within a predetermined range for use as a tentative address;

storing said tentative address in storage means;

generating an enquiry (ENQ) signal and transmitting said ENQ signal to a device having said tentative address on said communication system;

sensing said communication medium for the reception of an acknowledge (ACK) signal transmitted by an agent other than said agent being assigned a unique address in response to its receipt of said ENQ signal, and upon sensing said ACK signal generating another random number for use as an alternate tentative address;

storing said tentative address as a final address in said storage means in the absence of the receipt of said ACK signal within a predetermined time (IFG) after the transmission of said ENQ signal;

whereby an agent is assigned a unique address on said communication system.

14. The method as defined by claim 13, wherein said generating step generates and transmits a plurality of ENQ signals, each of said ENQ signals being transmitted after said IFG time has elapsed in the absence of the receipt of an ACK signal.

15. The method as defined by claim 14, wherein said tentative address is stored as a final address in the absence of an ACK signal once said IFG time has elapsed after the last of said plurality ENQ signals have been transmitted.

16. The method as defined by claim 15, further including the step of generating a synchronization pulse and transmitting said synchronization pulse on said communication medium, said synchronization pulse being transmitted prior to other signals on said medium.

17. The method as defined by claim 16, wherein said ENQ signal includes an ENQ frame having at least one synchronization flag byte having a predefined bit sequence and an ENQ byte.

18. The method an defined by claim 17, wherein said ACK signal includes an ACK frame having at least one synchronization flag byte having a predefined bit sequence and an ACK byte.

19. The method as defined by claim 18, wherein said random number is in the range of 1 to 256.

* * * * *

United States Patent [19]

Sander et al.

[11] **Patent Number: 4,742,448**

[45] **Date of Patent: May 3, 1988**

[54] **INTEGRATED FLOPPY DISK DRIVE CONTROLLER**

[75] Inventors: **Wendell B. Sander**, Los Gatos; **Robert Bailey**, San Jose, both of Calif.

[73] Assignee: **Apple Computer, Inc.**, Cupertino, Calif.

[21] Appl. No.: **943,839**

[22] Filed: **Dec. 18, 1986**

Related U.S. Application Data

[63] Continuation of Ser. No. 573,067, Jan. 24, 1984, abandoned.

[51] **Int. Cl.**⁴ **G06F 13/12; G06F 3/06**

[52] **U.S. Cl.** **364/200**

[58] **Field of Search** ... 364/200 MS File, 900 MS File

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,100,601	7/1978	Kaufman et al.	364/200
4,148,098	4/1979	McCreight et al.	364/200
4,210,959	7/1980	Wozniak	364/200
4,357,657	11/1982	Fellinger	364/200
4,423,480	12/1983	Bauer et al.	364/200
4,433,378	2/1984	Leger	364/200
4,494,196	1/1985	Greer	364/200
4,549,262	10/1985	Chung et al.	364/200

OTHER PUBLICATIONS

Western Digital; May 1980; FD179X-02 Floppy Disk Formatter/Controller Family; 24 pages.

Zaks; 1979; Microprocessor Interfacing Techniques; pp. 198-207.

Intel, 8271/2 Data Sheets, 1979.

Primary Examiner—Gary V. Harkcom
Assistant Examiner—C. H. Lynt
Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman

[57] **ABSTRACT**

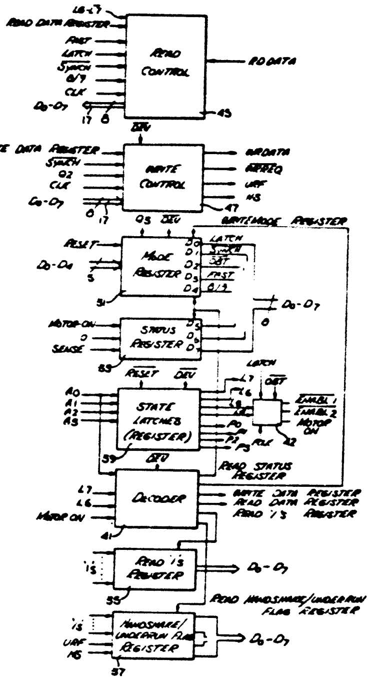
A floppy disk drive controller interface implemented as an integrated circuit on a single semi-conductor chip. The controller connects to a host computer data bus and one or more floppy disk drives. Based upon clocking and control signals received from a digital computer, the controller generates serial encoded data for recording on a floppy disk and receives serial encoded data previously recorded on a floppy disk. The controller comprises a read control circuit including a read data register, write control means including a write data register, a mode register, a status register, state latches, a decoder and special function registers. The controller operates by the setting and clearing of the state latches and reading or writing the mode register, the status register, the special function registers, the read data register and the write data register. The setting of a state latch and accessing of a register is done simultaneously. The controller, under software control, operates in a synchronous or asynchronous read/write mode, and slow or fast read/write mode.

8 Claims, 4 Drawing Sheets

IWM chip

(Integrated
Woz *
Machine)

*Woz ≡ Steve
Wozniak
(Apple 2
designer)



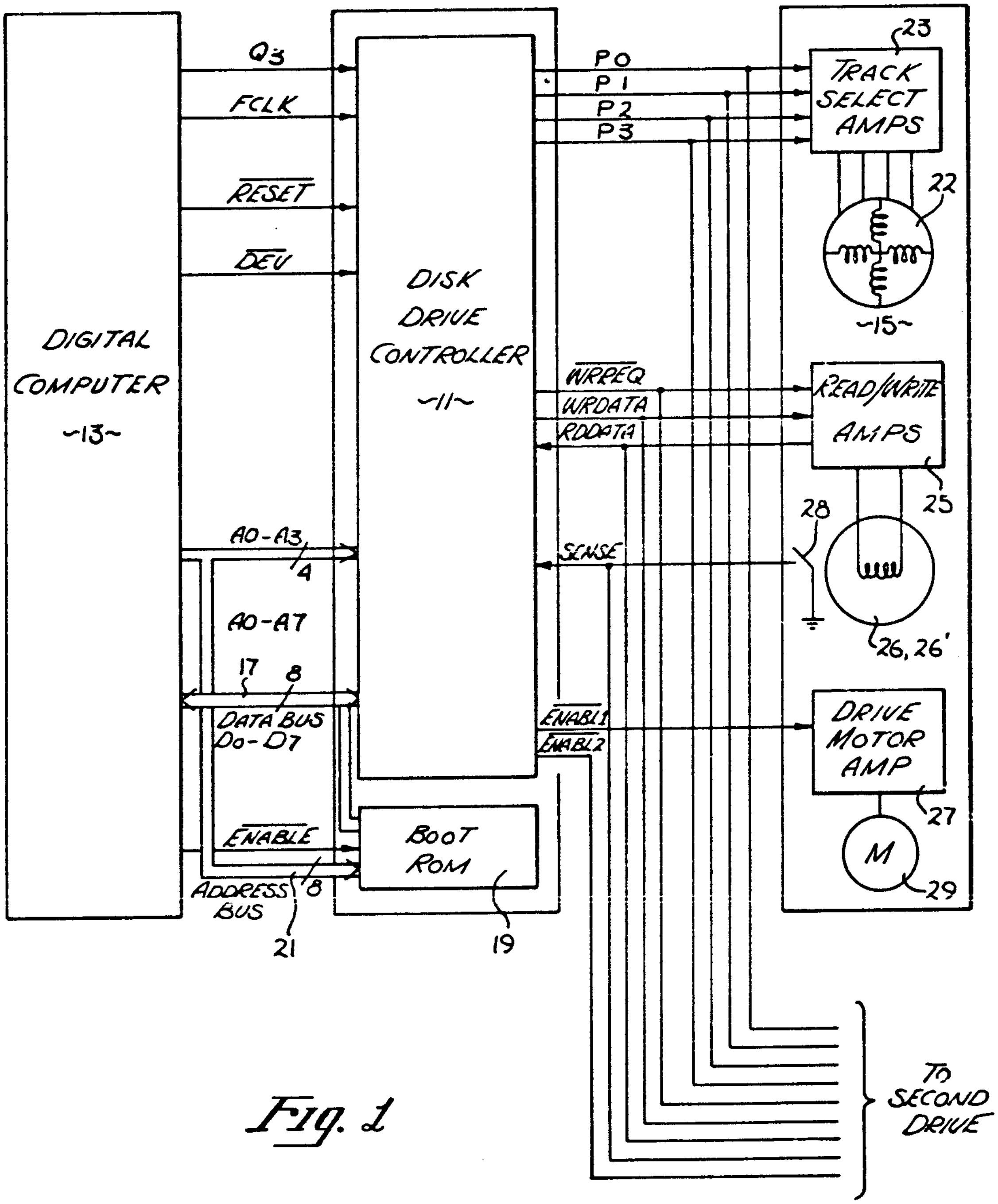

DIGITAL COMPUTER
~13~
Q3
FCLK
RESET
DEV
A0-A3
4
A0-A7
17
8
DATA BUS
D0-D7
ENABLE
ADDRESS BUS
8
21
DISK DRIVE CONTROLLER
~11~
BOOT ROM
19
P0
P1
P2
P3
WRREQ
WRDATA
RDDATA
SENSE
ENABL1
ENABL2
TRACK SELECT AMPS
23
22
~15~
READ/WRITE AMPS
25
28
26, 26'
DRIVE MOTOR AMP
27
M
29
TO SECOND DRIVE

Fig. 1

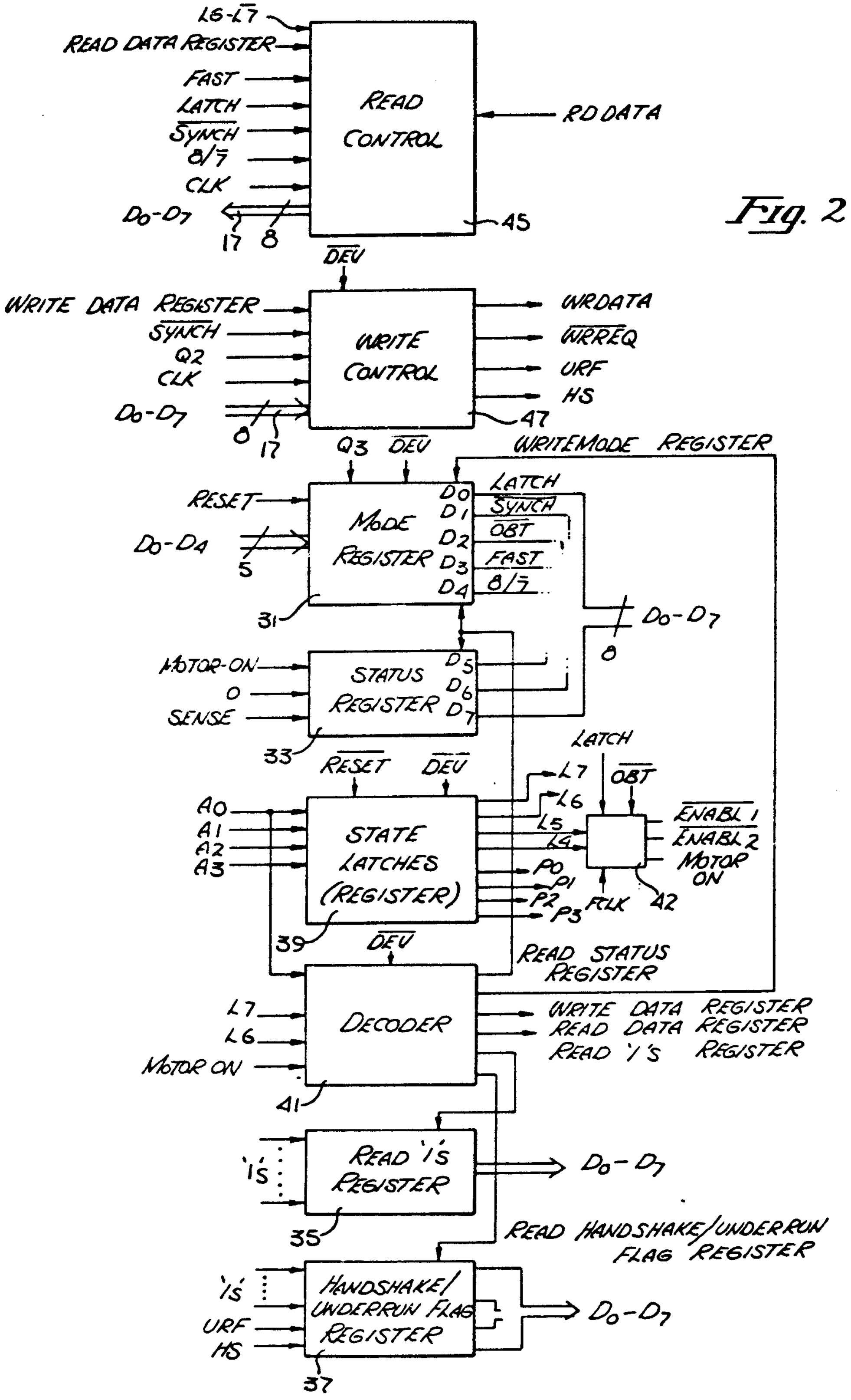


Fig. 2

"APPLE_PAT_4_742_448_03" 123 KB 2000-02-21 dpi: 300h x 300v pix: 1827h x 2934v

U.S. Patent May 3, 1988 Sheet 3 of 4 4,742,448

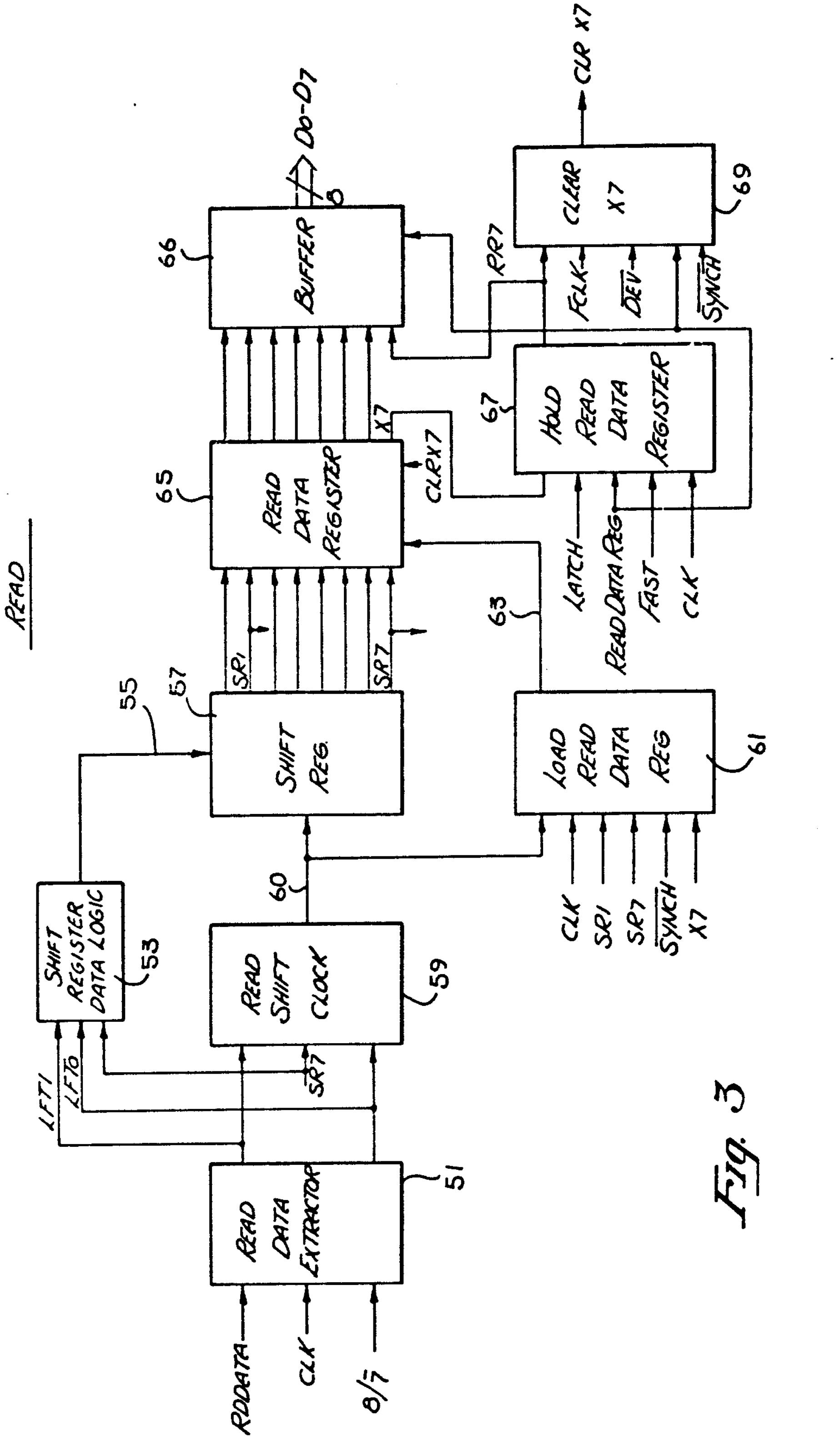


Fig. 3

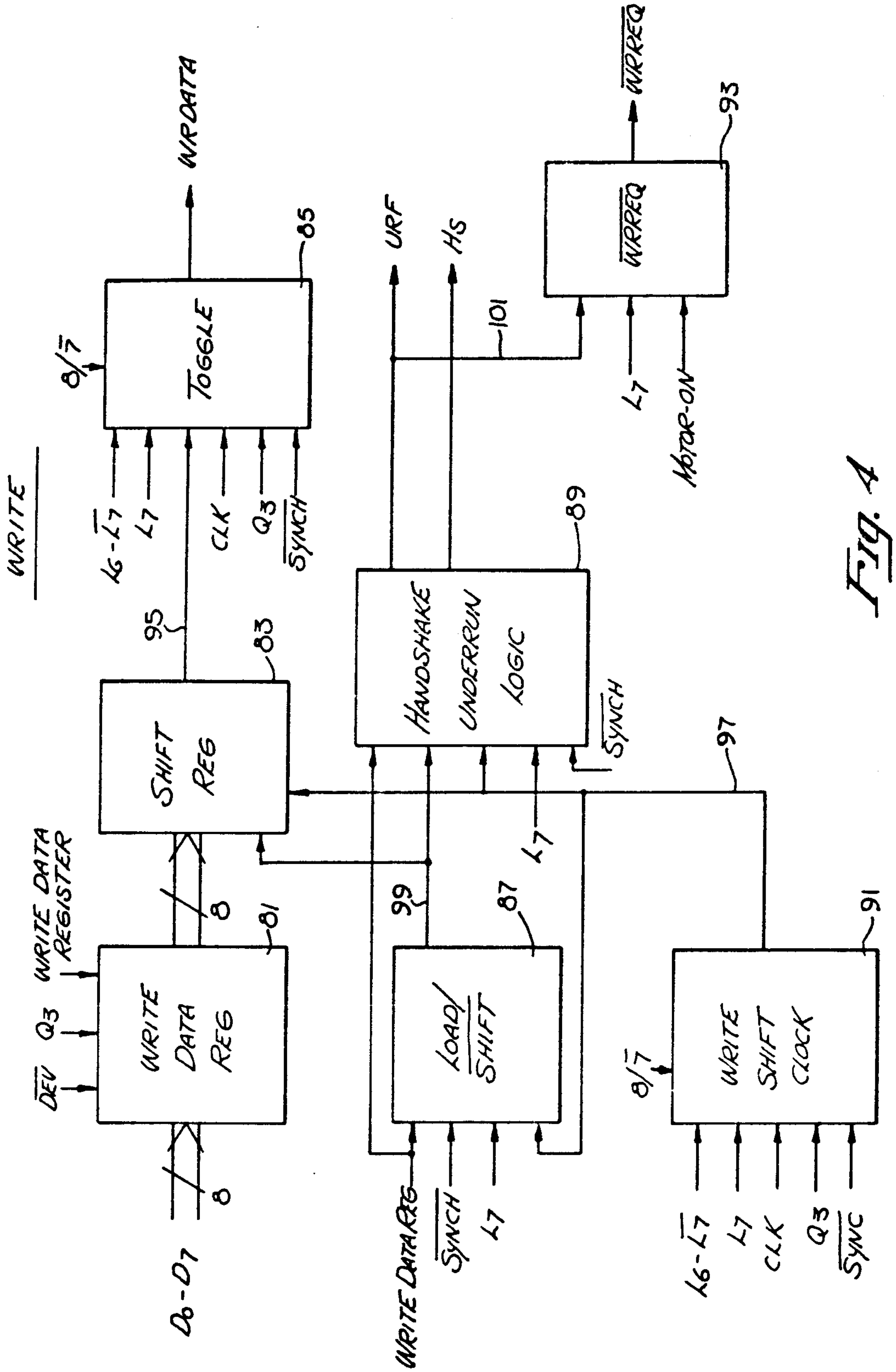

WRITE
WRDATA
85
TOGGLE
8/$\overline{7}$
L6-$\overline{L7}$
L7
CLK
Q3
$\overline{SYNCH}$
95
83
SHIFT REG
WRITE DATA REGISTER
Q3
$\overline{DEV}$
81
WRITE DATA REG
8
8
D0-D7
99
87
LOAD/SHIFT
WRITE DATA REG
$\overline{SYNCH}$
L7
89
HANDSHAKE UNDERRUN LOGIC
$\overline{SYNCH}$
L7
97
URF
HS
101
93
$\overline{WRREQ}$
$\overline{WRREQ}$
L7
MOTOR-ON
91
WRITE SHIFT CLOCK
8/$\overline{7}$
L6-$\overline{L7}$
L7
CLK
Q3
$\overline{SYNC}$

Fig. 4

INTEGRATED FLOPPY DISK DRIVE CONTROLLER

This is a continuation of application Ser. No. 573,067 filed Jan. 24, 1984, now abandoned.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention relates to the field of controllers for interfacing between a digital computer and a floppy disk drive. The disclosed invention is particularly suited for implementation as an integrated circuit.

2. Prior Art

Floppy disk controllers for interfacing between digital computers and floppy disk drives are well known. Such disk drives include a motor for rotating a floppy disk, a floppy disk being comprised of a flexible material shaped such that it is flat and circular and onto which is bonded a magnetic medium; a write head for recording data on the magnetic medium and a read head for reading data from the magnetic medium; a stepper motor for moving the read and write heads along the surface of the floppy disk; and electronic and logic circuitry for receiving binary signals which turn the disk drive motor on and off, move the read and write heads and cause electrical signals to be sent to the write head for recording data or receive electrical signals generated by the read head as the magnetic medium rotates past it. Disk drive controllers generate the necessary binary signals to turn the disk drive motor on and off, move the read and write heads and send appropriate signals to the electronic and logic circuitry of the disk drive to cause the read and write heads to read from or write to the magnetic medium of the rotating floppy disk. Disk drive controllers generate the appropriate signals to control the operation of disk drives by appropriate control, data and clock signals received from a digital computer.

In U.S. Pat. No. 4,210,959, a floppy disk drive controller is disclosed comprised of a serial/parallel shift register, controller logic and timing means and latches. The serial/parallel shift register is used to transfer data to and from the computer on a data bus. The controller logic and timing means receives signals from the latches to place the controller logic means in one of four possible modes of operation namely, read, sense write protect/write initialize, write record and write load. All reading and writing is done in a synchronous manner based upon a clock signal CLK. The aforesaid invention is directed to a relatively simple, inexpensive controller suitable for consumer and small business applications. The present invention is an integration of the controller disclosed in U.S. Pat. No. 4,210,959 with extensions and improvements including the capability of multiple modes of operation.

BRIEF SUMMARY OF THE INVENTION

A floppy disk drive controller interface is disclosed which is implemented in an integrated circuit. The controller connects to a host computer data bus and one or more floppy disk drives. Based upon clocking and control signals received from a digital computer, the controller generates serial encoded data for recording on a floppy disk and receives serial encoded data previously recorded on a floppy disk. The controller comprises read control means including a read data register, write control means including a write data register, a mode register, a status register, state latches, a decoder and special function registers. The controller operates by the setting and clearing of the state latches and reading or writing the mode register, the status register, the special function registers, the read data register and the write data register. The setting of a state latch and accessing of a register is done simultaneously. The controller, under software control, operates in a synchronous or asynchronous read/write mode, and slow or fast read/write mode.

Control signals received by the controller from the computer set or reset one of eight state latches. Two of the latches select one of two disk drives and turn the drive motor of the selected disk drive on or off. Four of the latches control a stepper motor in the disk drive which cause the read and write heads to move from track to track of the floppy disk. The remaining two latches are coupled to the decoder which decodes clocking and control signals received from the computer and generates signals to the various registers of the controller and to the read control means and write control means for controlling the function to be performed by the disk drive.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 illustrates the controller of the present invention interfacing between a digital computer and a floppy disk drive.

FIG. 2 is a block diagram of the controller of the present invention.

FIG. 3 is a detailed block diagram of the read control means of the present invention.

FIG. 4 is a detailed block diagram of the write control means of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

A floppy disk drive controller, implemented as an integrated circuit, is disclosed for providing an interface between a digital computer and a floppy disk drive. In the following description, numerous specific details are set forth such as specific word or byte lengths, etc., to provide a thorough understanding of the present invention. However, it will be obvious to one skilled in the art that the present invention may be practiced without such specific details. In other instances, well known circuits have been shown in block diagram form in order not to obscure the present invention in unnecessary detail. Unless otherwise stated, for convenience, positive logic will be used to describe the invention. Thus, the terms set, "1", high and true are equivalent as are the terms reset, "0", low and false.

The presently preferred embodiment of the controller provides an interface between microcomputers manufactured by Apple Computer, Inc. of Cupertino, Calif. such as its Apple-II computer and successors thereto, and floppy disk drives such as Disk II manufactured by Apple Computer, Inc., and successors thereto.

Referring first to FIG. 1, the controller 11 of the present invention is shown as an interface between a digital computer 13 and a floppy disk drive 15. The digital computer 13 is coupled to the controller 11 through a bidirectional data bus 17 (D0–D7), control lines A0–A3, device select line $\overline{DEV}$, reset line $\overline{RESET}$ and clock lines Q3 and FCLK. Although not part of the present invention, also shown in FIG. 1 is boot ROM or PROM 19 which is coupled to the digital computer through data bus 17, address bus 21 (A0–A7) and an enable line $\overline{\text{ENABLE}}$. When the computer is first turned on, or whenever it is necessary to reinitialize the computer operating system, a program stored in boot ROM 19 is utilized to instruct the controller 11 to read a program recorded on a floppy disk in disk drive 15 and transfer it over data bus 17 to computer 13. Such boot or boot strap programs are well known in the art and will not further discussed herein.

Data on data bus 17, depending upon signals which have been placed on control lines A0–A3, comprises a byte of data which has been received from the disk drive, which is to be sent to the disk drive or which is to be loaded into or read from registers within the controller 11. The controller 11 is selected by the computer by a "0" on line $\overline{\text{DEV}}$ and is placed in an initial state by a "0" on line $\overline{\text{RESET}}$. Clock signals generated by the computer on lines Q3 and FCLK are used by the controller as timing signals. Clock signals Q3 and FCLK are generated with periods which depend on the speed of the processor in the computer. In a preferred embodiment, Q3 is a 2 MHz clock and FCLK is a 7 MHz clock. Additionally, Q3 may be left at "0" (if only asynchronous mode is used) and/or FCLK may be 8 MHz.

Data/control lines between the controller 11 and disk drive 15 are as follows. Signals on lines P0 through P3 control a stepper motor 22 which rotates a unit turn in either a forward or backward direction depending upon the signals on lines P0 through P3. In a typical floppy disk drive, a unit turn is a one quarter turn, a one eighth turn, or a one sixteenth turn, however, this value is strictly drive dependent. Each unit turn of the stepper motor causes the read and write heads to move a unit distance in a forward or backward direction. The unit distance the heads move is also drive dependent, but typical unit distances are one-half or one-quarter track. The binary signals on lines P0 through P3 are input to track select amplifiers 23 which convert the binary signals into a voltage which rotates the stepper motor 22.

Signals on WRDATA are binary signals generated by the controller and are input into read/write amplifiers 25 of disk drive 15. Signals on WRDATA cause read/write amplifiers 25 to energize or de-energize the write head coil 26 to cause data to be written on the magnetic medium as it spins under the write head. Signals on $\overline{\text{WRREQ}}$ enable to disable write head coil 26 to allow or prevent the writing of data based on WRDATA. Similarly, as the magnetic medium passes under the read head, the read head coil 26' is energized or de-energized and the detected data is converted by the read/write amplifiers 25 into a binary signal which is placed on line RDDATA.

A write protect sense signal is generated by the disk drive 15 and placed on the SENSE line when a switch 28 in the disk drive is closed to indicate that the disk drive has been placed in a write protect state. Such switch may be a mechanical switch operated by a user and/or a switch which detects whether a floppy disk jacket has a write protect notch, such as, for example, a photocell which causes a transistor switch to close when light to it is blocked by the floppy disk jacket.

Lastly, drive select signals are generated by the controller and placed on lines $\overline{\text{ENABL1}}$ or $\overline{\text{ENABL2}}$. $\overline{\text{ENABL1}}$ is input to a first disk drive and $\overline{\text{ENABL2}}$ is input to a second disk drive. Each of these $\overline{\text{ENABL1}}$ or $\overline{\text{ENABL2}}$ inputs is coupled to a drive motor amplifier 27 which converts the binary signal into a voltage to cause a motor 29 in the disk drive to rotate thereby

spinning a floppy disk which has been inserted into the disk drive. In the disclosed embodiment, a single bit in the controller is used to generate a signal on $\overline{\text{ENABL1}}$ or $\overline{\text{ENABL2}}$ and, therefore, only one of two drives can be selected at any given point in time. Of course, with additional hardware, additional drives can be connected to the controller. It should be noted that although only one set of lines is shown as being coupled to controller 11, with respect to lines such as SENSE, which may be set for one drive and reset for the other, appropriate logic circuits are employed to ensure that only signals from the selected drive are input to the controller logic.

Referring now to FIG. 2, the main components of the controller 11 will now be described. The invented controller comprises mode register 31; status register 33; read 1's register 35; handshake/underrun flag register 37; state latches 39; decoder 41; read control means 45 and write control means 47. Read control means 45 and write control means 47 will be discussed more fully below with respect to FIGS. 3 and 4 respectively.

Once the controller 11 has been selected by the computer 13 by a signal on $\overline{\text{DEV}}$ and the controller has been initialized by a signal on $\overline{\text{RESET}}$ (which sets the state latches to their default values), the controller is instructed by the computer to perform a particular function by signals on A0–A3 which set or reset one of eight state latches 39 (P0 through P3 and L4 through L7). It should be understood that regardless of the states of the latches P0 through P3 and L4 through L7, unless the controller has been selected by a signal on $\overline{\text{DEV}}$, no operations will be performed by the controller. $\overline{\text{DEV}}$ enables the controller when it is low. The falling edge of $\overline{\text{DEV}}$ latches information on A0 through A3. One of the aforesaid eight latches is set by a "1" on A0 and reset by a "0" on A0. The particular latch to be set or reset based upon A0 is determined by the address set on A1 through A3. Table 1 shows the addresses on A1 through A3 which correspond to latches P0 through P3 and L4 through L7.

TABLE 1

A3	A2	A1	LATCH
0	0	0	P0
0	0	1	P1
0	1	0	P2
0	1	1	P3
1	0	0	L4
1	0	1	L5
1	1	0	L6
1	1	1	L7

Signals on P0 through P3 cause the stepper motor 22 to operate as follows. Setting P0 causes the stepper motor to be placed in an initial state readying it for a one unit turn in either a forward or backward direction depending upon the next signal received. If the next signal received is P1 (i.e., when latch P1 is set), the stepper motor turns one unit which causes the read and write heads to move a unit distance forward. If P3 is set after P0, then the stepper motor turns one unit in the opposite direction and the read and write heads step one unit distance backwards. At this point, both P0 and P1 are set (or P0 and P3 if the heads are being moved backwards) and P0 is cleared. After P0 is cleared, assuming additional forward head travel is desired, P2 is set which causes the stepper motor to turn an additional unit in the forward direction stepping the read and write heads another unit distance forward. If additional head movement in the forward direction is necessary, P1 is cleared and P3 is set causing an additional unit turn of the stepper motor. In a similar manner, if backwards movement of the read and write heads are necessary, and P0 has been set followed by P3, P0 is cleared and P2 is set followed by the clearing of P3 and the setting of P1, each of which causes the stepper motor to rotate a unit turn in the opposite direction and step the read and write heads a unit distance in a backwards direction. Further cycles of P0, P1, P2, P3 (for forward motion) or P0, P3, P2, P1 (for backwards head travel), may be issued by the computer 13 by addresses on A0 through A3, as appropriate, to cause the read and write heads to move to any desired track.

The setting and clearing of L4 through L7 determine other functions to be performed by the controller 11 as described below.

After the controller has been selected by $\overline{\text{DEV}}$ and initialized by $\overline{\text{RESET}}$, and WRITE MODE REGISTER is set as described below, D0 through D4 on the data bus 17 are loaded into the mode register 31 to select a particular mode of operation for subsequent reads and writes. The data on D0 through D4 correspond respectively to the signals LATCH, $\overline{\text{SYNCH}}$, $\overline{\text{OBT}}$, FAST and 8/$\overline{7}$ of the mode register. LATCH will be discussed more fully below with respect to the read control means 45 and FIG. 3. $\overline{\text{SYNCH}}$, when cleared, places the controller in a synchronous mode for subsequent reads and writes. When $\overline{\text{SYNCH}}$ is set, subsequent reads and writes are performed in an asynchronous mode. Both synchronous and asynchronous modes of operation will be discussed more fully below with respect to FIGS. 3 and 4.

$\overline{\text{OBT}}$ when cleared enables a one second on board timer. When $\overline{\text{OBT}}$ is set, the timer is disabled. The on board timer will be discussed more fully below with respect to $\overline{\text{ENABL1}}$ and $\overline{\text{ENABL2}}$ which select one of two disk drives which are coupled to the controller.

When FAST is cleared, the controller operates in slow mode. Normally, internal timing of the controller is based upon the clock signal CLK which is equal to the clock signal FCLK generated by the computer. When FAST is cleared, internal timing, i.e. CLK period, is equal to twice the period of FCLK.

8/$\overline{7}$ also relates to timing the FCLK. When an 8 MHz clock is in use, 8/$\overline{7}$ is set. If FCLK is running at 7 MHz, 8/$\overline{7}$ is cleared. The value of 8/$\overline{7}$ is used by the controller to determine how many FCLK periods are required for a given unit of time. For example, if FCLK is 8 MHz, one microsecond will be eight clock periods; if FCLK is 7 MHz, one microsecond will be seven clock periods. This allows computers with 7 MHz clocks and computers with 8 MHz clocks to read and write equivalently, that is, data written by a computer with a 7 MHz clock can be read by a computer with an 8 MHz clock and visa versa.

After the mode register has been loaded to set up particular modes of operation, one of the two drives is selected by latch L5 as follows. When latch L5 is cleared, drive 1 is selected. When latch L5 is set, drive 2 is selected. After a drive has been selected, setting latch L4 will cause line MOTOR-ON to go to "1". When latch L4 is set, if latch L5 is "0", drive 1 is enabled by $\overline{\text{ENABL1}}$; if L5 is "1", drive 2 is enabled by $\overline{\text{ENABL2}}$.

$\overline{\text{OBT}}$ mentioned above can now be described. When $\overline{\text{OBT}}$ is set, if L4 is cleared, $\overline{\text{ENABL1}}$ or $\overline{\text{ENABL2}}$ is disabled by logic circuit 42, which includes the onboard timer, depending upon the setting of L5, thereby shutting down drive motor 29. However, if $\overline{\text{OBT}}$ is cleared, then the clearing of L4 will not cause logic circuit 42 to disable $\overline{\text{ENABL1}}$ or $\overline{\text{ENABL2}}$ until a one second timer has elapsed (if LATCH is reset or until a one-half millisecond timer has elapsed if LATCH is set). Generally, it is preferable that there be a delay before turning off a drive motor because subsequent disk operations frequently occur in a very short time frame after prior disk operations. Thus, without the delay before disabling $\overline{\text{ENABL1}}$ or $\overline{\text{ENABL2}}$, subsequent disk operations would be subjected to waiting for the motor to achieve proper speed. Of course, the operation system or other program in the computer should include appropriate waits or timing loops, when necessary, to ensure that no disk reads or writes are requested until the drive motor is up to speed. Additional functions performed by the controller are determined by the settings of L6, L7, and MOTOR-ON. L6, L7 and MOTOR-ON select which register is to be read or written as described below. Registers are read during any operation in which A0 is being cleared. Registers are written to when A0 is being set. L6, L7 MOTOR-ON, A0 and $\overline{\text{DEV}}$ are input to decoder 41 which decodes the inputs and, as described below, places a "1" on one of the lines READ STATUS REGISTER, WRITE MODE REGISTER, WRITE DATA REGISTER, READ DATA REGISTER, READ 1's REGISTER or READ HANDSHAKE/UNDERRUN FLAG REGISTER. Each of the following operations take place as the falling edge of $\overline{\text{DEV}}$ is input to decoder 41.

When L6, L7 and MOTOR-ON are "0", the decoder 41 places a "1" on READ 1's REGISTER which causes the read 1's register 35 to place a byte of binary 1's on the data bus 17, lines D0 through D7. The 1's on the data bus are read into the memory of the computer for use by the operating system or other program.

When L6, L7 are "0" and MOTER-ON is "1", the decoder 41 places a "1" on READ DATA REGISTER. The function performed when READ DATA REGISTER is set will be discussed below with reference to the read control means 45 and FIG. 3.

When L6 is "1", L7 is "0" and MOTOR-ON is "0" or "1" (i.e. don't care), the decoder 41 places a "1" on READ STATUS REGISTER which causes the contents of the mode register 31 and status register 33 to be placed on data bus 17, such that the bus takes on the following values; LATCH is placed on D0, $\overline{\text{SYNCH}}$ is placed on D1, $\overline{\text{OBT}}$ is placed on D2, FAST is placed on D3, 8/$\overline{7}$ is placed on D4, MOTOR-ON is placed on D5, a 0 is placed on D6 and SENSE, from the disk drive, is placed on D7. The operating system or other program in the computer 13 is then able to determine the status of controller 11.

When L6 is "0", L7 is "1" and MOTOR-ON is "0" or "1", the decoder 41 places a "1" on READ HANDSHAKE/UNDERRUN FLAG REGISTER which causes the handshake/underrun flag register 37 to place "1"'s on DO through D5, an underrun flag URF on D6 and a handshake flag HS on D7. The underrun flag URF and the handshake flag HS will be discussed with respect to the write control means 47 and FIG. 4.

When L6 is "1", L7 is "1" and MOTOR-ON is "0", the decoder 41 places a "1" on WRITE MODE REGISTER and the data on D0 through D4 of the data bus 17 is written into the mode register 31 with D0 corresponding to LATCH, D1 corresponding to $\overline{\text{SYNCH}}$, D2 corresponding to $\overline{\text{OBT}}$, D3 corresponding to FAST and D4 corresponding to 8/$\overline{7}$. This occurs during

WRITE MODE REGISTER at the rising edge of the logical function Q3 or $\overline{\text{DEV}}$.

When L6, L7 and MOTOR-ON are "1", the decoder 41 places a "1" on WRITE DATA REGISTER. The function performed when WRITE DATA REGISTER is set will be discussed below with reference to write control means 47 and FIG. 4.

The read control means 45 will now be discussed with reference to FIG. 3. As noted above, with L6 and L7 equal to "0" and MOTOR-ON equal to "1", the decoder 41 places a "1" on READ DATA REGISTER. Of course, prior to reading, the read head is moved to the desired track of the floppy disk by rotating the stepper motor 22 according to control signals on P0 through P3 as described above. As the floppy disk rotates under the read head, data recorded the track causes the coil in the read head to be energized and de-energized causing fluctuations on RDDATA corresponding to set bits and cleared bits on the magnetic medium. At this time, neither the controller nor the computer can determine which portion of a track is under the read head. Therefore, a method for determining where data reading should be started is necessary. A method for providing proper synchronization for such purpose is described in U.S. Pat. No. 4,210,959.

Once synchronization has been obtained, reading proceeds as follows. The read data extractor 51 detects negative transitions of RDDATA synchronized to the CLK clock signal. Each time a negative transition of RDDATA occurs, it resets an interval counter. When $8/\overline{7}$ is set, the interval is 16 CLKs. When $8/\overline{7}$ is reset, the interval is 14 CLKs. The information on RDDATA is spaced at these intervals or "around" these intervals. A "1" is a negative transition at the expected time, i.e. interval. A "0" is no transition at the expected time. The expected time is widened by approximately one-half an interval before and after the expected time since the data is not precisely spaced when read due to variations in drive speed and other external factors.

A negative transition of RDDATA is detected as a "1" and the read data extractor 51 causes the signal LFT1 to pulse to a "1" for one CLK cycle. The next expected data is nominally at 16 CLKs when $8/\overline{7}$ is set. This may range between $16-8=8$ CLKs and $16+7=23$ CLKS. Thus, if another negative transition of RDDATA occurs between 8 and 23 CLKs, another "1" is detected and LFT1 pulses to a "1" for one CLK cycle. If no negative transition occurs on RDDATA between 8 and 23 CLKs a "0" is detected and LFT0 pulses to "1" for one CLK cycle.

If a LFT1 has occurred within the expected time, the interval counter is reset, otherwise the next expected data is nominally at 32 CLKs. This may range between $32-8=24$ CLKS and $32+7=39$ CLKs. If a negative transition of RDDATA occurs between 24 and 39 CLKs, a "1" is detected and LFT1 will pulse to "1" for one CLK cycle. If no negative transition of RDDATA occurs a "0" is detected and LFT0 will pulse to "1". Similarly, subsequent intervals are widened from the nominal number of CLKs by minus 8 CLKs and plus 7 CLKs with LFT1 being pulsed if a negative transition of RDDATA occurs within the widened interval and LFT0 being pulsed if there is no negative transition of RDDATA. When $8/\overline{7}$ is reset, LFT0 and LFT1 are pulsed as described above, except intervals are nominally 14 CLKs and are widened minus 7 CLKs and plus 6 CLKs.

LFT0 and LFT1 are input to shift register data logic circuitry 53 which sets line 55 if LFT1 is "1" or clears line 55 if LFT0 is "1" unless SR7 is "1" (as described below), the data on line 55 being the data input to shift register 57.

The data on line 55, when shift register 57 is signaled by shift clock 59 by a signal on line 60, is input to the shift register one bit at a time. Shift clock 59 sets line 60 at the end of each LFT1 pulse or LFT0 pulse except when SR7 is set. SR7 is set after a full byte of data has been shifted into the shift register. This occurs because the initial bit received by the shift register 57 from the data stored on the disk is always a "1" according to the group code coding scheme utilized for storing data on the diskette. Wherein the leading bit of a byte is always a "1".

Once SR7 is set, load read data register logic 61 generates a signal on line 63 which causes the data in shift register 57 to be parallel loaded into the read data register 65. The shift register 57 is cleared one half a read shift clock after SR7 is set so that it is ready to accept the next byte of data.

The signal on line 63 is set by load read data register logic 61 as follows.

In synchronous mode, i.e. when $\overline{\text{SYNCH}}$ is "0", when X7 is reset, the read data register 65 is loaded with the data in the shift register 57 each time the shift register 57 shifts by the setting of line 63 by load read data register logic 61. However, when X7 is set, i.e., when the first bit of the byte being read arrives at the far end of the shift register and is parallel loaded into the read data register 65, the load read data logic 61 will hold line 63 low for four CLKs after SR1 (corresponding to bit 1 of shift register 57) becomes "1" due to the first bit of the next byte being shifted through shift register 57. This delay is to ensure that the byte in the read data register 65 is there, and therefore available to be routed to buffer 66 and on data bus 17 D0 through D7, long enough to be seen by the computer 13, but not long enough to be seen as a valid byte twice. The rising edge of D7 is delayed by hold read data register logic 67 so that if D7 is read by the computer 13 as "1", it is guaranteed that the data on D0 through D6 will have been correctly written into a register in the computer 13. This delay is created by the hold read data register logic 67 as follows. When LATCH is cleared, which it should be during synchronous mode operation, and X7, corresponding to bit 7 of read data register 65, is set, output RR7 from hold read data logic 67, which corresponds to input bit 7 of buffer 66, is not set until 1 CLK period, when FAST is "1" (fast mode), and a ½ CLK period when FAST is "0" (slow mode) after X7 is set.

In asynchronous mode, i.e. when $\overline{\text{SYNCH}}$ is set, read data register 65 is parallel loaded from shift register 57. This ocurs by the load read data register logic 61 setting line 63 when SR7 is set. To ensure that the data in read data register 65 is properly loaded into a register in computer 13, in asynchronous mode, LATCH should always be set. When LATCH is set, the data on X7 is placed on RR7 by hold read data register logic 67 at the rising edge of READ DATA REGISTER. This ensures that D7 will meet the set up and hold requirements of the computer 13. If D7 is read by the conputer 13 as "1", D0 through D7 are correctly written into a register of the computer 13. X7 will be reset by clear X7 logic 69 fourteen FCLK's after READ DATA REGISTER is set and D7 is "1" (i.e., the byte has been read by the computer) so that X7 will be clear and the computer 13 will not re-read the byte as valid during subsequent polling, i.e., setting of READ DATA REGISTER.

Write control means 47 will now be described with reference to FIG. 4. Write control means comprises write data register 81 for receiving a byte of data to be written on the disk, shift register 83 for converting the parallel data in write data register 81 to serial form, and toggle 85 for generating the bitstream which is to be written onto the disk. Write control means 47 further comprises load/$\overline{\text{shift}}$ logic 87, handshake/underrun logic 89, write shift clock 91 and $\overline{\text{WRREQ}}$ logic 93, all of which control the timing of the write control means.

To initiate a write, L6 is set, L7 is cleared to set up a pre-write state. The pre-write state initializes the write shift clock 91 and load/$\overline{\text{shift}}$ logic circuit 87 setting line 99, sets WRDATA and $\overline{\text{WRREQ}}$, resets underrun flag URF in handshake/underrun flag register 37 and initializes a toggle clock in toggle 85. Prior to actual writing, L4 and L5 should be placed in appropriate states to select the desired drive and set MOTOR-ON. When L6, L7 and MOTOR-ON are "1", the decoder places a "1" on WRITE DATA REGISTER which loads data from data bus 17, D0 through D7, to the write data register 81 at the rising edge of the logical function Q3 or $\overline{\text{DEV}}$. This register is in turn parallel loaded into shift register 83 as follows. As noted above, when load/$\overline{\text{shift}}$ control logic 87 is initialized, line 99 is set. When line 99 is set, a pulse from the write shift clock 91 on line 97 causes data in write data register 81 to be latched into shift register 83. In asynchronous mode ($\overline{\text{SYNCH}}$ is set), the load will be completed approximately eight CLK's after WRITE DATA REGISTER has been set. In synchronous mode, the load will be completed between four and five Q3 periods after WRITE DATA REGISTER has been set.

In synchronous mode, ($\overline{\text{SYNCH}}$ is reset) writing continues as follows. Once the data has been loaded into shift register 83, the most significant bit in the shift register will be shifted onto line 95 which will cause (after two Q3 periods) the WRDATA to toggle from "1" to "0" since WRDATA is initialized at "1" and, according to the group code coding scheme used, the first bit of a byte must be a "1". Shift register 83 will shift every eight Q3 periods after it has been loaded, followed two Q3 periods later with a toggle, if the date on line 95 is a "1", and will continue such shift and toggle until the byte has been written. Thus, a byte of data is shifted out and written in 64 Q3 periods and a new byte of data can then be parallel loaded into shift register 83. With this timing, a "1" must be placed on WRITE DATA REGISTER every 64 Q3 periods, otherwise 0's will be shifted out of shift register 83. During synchronous mode URF is always reset so that URF does not prevent writing data on disk by causing $\overline{\text{WRREQ}}$ to be set.

When the controller is in asynchronous mode ($\overline{\text{SYNCH}}$ is set), the timing constraints of synchronous writes are relaxed. When in asynchronous mode, write control means 47 operates as follows. After shift register 83 has been parallel loaded with the data from write data register 81, the most significant bit in shift register 83 will be shifted onto line 95 and after eight more CLK periods, toggle 85 will cause WRDATA to toggle from "1" to "0" since, as noted above, the most significant bit must be a "1". Subsequent shifts and toggles are separated by eight CLKs. After all eight bits have been shifted out of shift register 83, load/$\overline{\text{shift}}$ logic 87 places a "1" on line 99 which parallel loads shift register 83, with data from write data register 81. When 8/$\overline{7}$ is set, shifts and toggles are separated by 8 CLKs. When 8/$\overline{7}$ is reset, toggles occur 6 CLKs after shifts, and shifts occur 8 CLKs after toggles.

Due to the relaxed timing which occurs during asynchronous writes as compared to synchronous writes, the following additional operations are needed to ensure that data is being properly written. Handshake flag HS is set by handshake/underrun logic 89 upon the completion of a parallel loading of shift register 83, as determined by signals on lines 97 and 99 and reset by the handshake/underrun logic 89 when WRITE DATA REGISTER is enabled. Since computer 13 can issue a command to clear L6 which will cause the decoder to enable READ HANDSHAKE/UNDERRUN FLAG REGISTER, the status of the handshake flag HS can be determined by the computer. That is the computer can poll the handshake/underrun flag register 37 until the HS flag is "1" indicating that the write data register 81 has been parallel loaded into the shift register 83 and the write data register is available for another byte of data. Once the computer detects that the write data register 81 is available, it may issue a command to set L6 which will enable WRITE DATA REGISTER which will cause the byte on data bus 17 to be written into write data register 81.

To ensure that a new byte of data has in fact been loaded into the write data register 81 prior to loading the shift register 83, the underrun flag URF in handshake/underrun flag register 37 is employed as follows. As noted above, during the pre-write state when writing is initiated, underrun flag URF is reset, i.e. when L7 is "0". The underrun flag URF is set by handshake/underrun logic 89 when the parallel load of the shift register 83 ends, if the handshake flag is set, indicating a new byte has not been written into the write data register 81. Since the current state of underrun flag URF is input to $\overline{\text{WRREQ}}$ logic 93 through line 101, if URF is set then no new data has been loaded into write data register 81 before loading the shift register 83, and $\overline{\text{WRREQ}}$ logic 93 will enable $\overline{\text{WRREQ}}$ before the next transition of WRDATA occurs. When $\overline{\text{WRREQ}}$ is "1", the write head is disabled preventing the same byte of data from being rewritten. URF can only be reset by exiting from writing, i.e., when L7 is "0".

For an example showing how latches L4 through L7 are set by the computer during asynchronous writes, see Table 2. For an example showing how latches L4 through L7 are set by the computer during synchronous writes, see Table 3.

TABLE 2

(Asynchronous Writes)

L4	L5	L6	L7	MOTOR-ON	Action
0	0	0	0	0	initial state
0	0	1	0	0	set L6
0	0	1	1	0	set L7; write data on bus into the mode register
0	0	1	0	0	clear L7
0	0	0	0	0	clear L6
1	0	0	0	1	set L4; select drive 1, set MOTOR-ON
1	0	1	0	1	set L6; pre-write state; initialize write shift clock 91; initialize load/$\overline{\text{shift}}$ control; set WRDATA; set $\overline{\text{WRREQ}}$; reset URF
1	0	1	1	1	set L7; enable WRITE DATA REGISTER
1	0	0	1	1	clear L6; read HS and URF flags

TABLE 2-continued

(Asynchronous Writes)

L4	L5	L6	L7	MOTOR-ON	Action
1	0	0	1	1	continue polling HS flag until it has been set
1	0	1	1	1	set L6; enable WRITE DATA REGISTER
1	0	0	1	1	clear L6; read HS and URF flags
1	0	0	1	1	continue polling HS flag until it has been set
1	0	1	1	1	set L6; enable WRITE DATA REGISTER
1	0	1	0	1	clear L7; exit write mode
1	0	0	0	1	clear L6
0	0	0	0	1	clear L4;
0	0	0	0	0	MOTOR-ON clears after timer counts down

TABLE 3

(Synchronous Writes)

L4	L5	L6	L7	MOTOR-ON	Action
0	0	0	0	0	initial state
0	0	1	0	0	set L6
0	0	1	1	0	set L7; write data on bus into mode register
0	0	1	0	0	clear L7
0	0	0	0	0	clear L6
1	0	0	0	1	set L4; select drive 1, set MOTOR-ON
1	0	1	0	1	set L6; pre-write state; initialize write shift clock; initialize load/shift control; set $\overline{\text{WRDATA}}$; set $\overline{\text{WRREQ}}$
1	0	1	1	1	set L7; place a byte of data on data bus 17 every 64 Q3 clocks
1	0	1	0	1	clear L7; exit write mode when done
1	0	0	0	1	clear L6
0	0	0	0	1	clear L4
0	0	0	0	0	MOTOR-ON clears after timer counts down

The diclosed controller may be packaged in a standard 28 pin, 600 mil plastic DIP using well known prior art methods. All of the pinouts are shown in FIG. 1, except for voltage source Vcc and ground.

Thus, a disk controller for interfacing between a digital computer and a floppy disk drive which may be implemented as an integrated circuit has been described. The controller is capable of performing multiple modes of operation, including fast and slow clocking and synchronous and asynchronous reading and writing.

We claim:

1. An integrated circuit floppy disk drive controller formed in a single semiconductor device for interfacing between a digital computer having an address bus and a data bus, and at least one floppy disk drive, said disk drive controller and said computer being coupled by said data bus, said computer generating a clock signal which is input to said controller, said controller comprising:

state storage means for coupling to said computer by said address bus for storing state commands sent by said computer;

decoder means coupled to said state storage means for decoding state commands stored in said state storage means and generating control signals for controlling the operation of a status register means,

a read control means and a write control means based upon said decoded commands;

mode storage means coupled to said decoder means and for coupling to said computer, said mode storage means for storing data sent by said computer indicating modes of operation selected by said computer, said modes of operation including at least one of synchronous/asynchronous reading and writing and fast/slow clock;

said status register means coupled to said decoder means, and for coupling to said floppy disk drive and said computer for storing information regarding the status of said at least one disk drive and the controller for interrogation by said computer, said status being determined by the contents of said mode storage means and said status register means;

said read control means coupled to said mode storage means, and for coupling to said computer and said at least one disk drive for receiving data from said disk drive and sending said data to said computer in a mode of operation as determined by said mode storage means; and

said write control means coupled to said mode storage means, for coupling to said at least one disk drive for receiving data from said computer and sending said data to said disk drive in a mode of operation as determined by said mode storage means.

2. The controller defined by claim 1 wherein the state commands stored in said state storage means control positioning of a stepper motor in said at least one disk drive, enable and disable a drive motor in said at least one disk drive, select one of said at least one disk drives to write to or read from, and cause said decoder means to generate said control signals as determined by said state commands.

3. The controller defined by claim 2 further comprising a delay timer wherein said modes of operation are asynchronous reading and writing, synchronous reading and writing, timing based on said clock signal running at a first speed, timing based on said clock signal running at a second speed, enabling said delay timer for turning off a drive motor in said at least one disk drive, and disabling said delay timer for turning off said disk drive motor.

4. The controller defined by claim 1 wherein the information stored in said status register means is used to inform said computer when said at least one disk drive is in a write protect state and when a drive motor in said at least one disk drive is activated.

5. The controller defined by claim 1 wherein said read control means comprises:

a read data extractor means for converting serial signals received from said disk drive into a plurality of serial pulses representing binary '1's and binary '0's;

a shift register means coupled to said read data extractor means for converting said plurality of serial pulses into parallel data;

a register means coupled to said shift register means for storing parallel data from said shift register means until said parallel data can be placed on said data bus for transfer to said computer; and

a read data control means coupled to said read data extractor means, said shift register means and said register means, said timing signal from said computer being input to said read control means, said read data controls means for controlling the loading of data into said shift register means, said register means and onto said data bus, and using said timing signal to ensure that data sent to said computer is not lost and is not duplicated.

6. The controller defined by claim 5 wherein said read data control means comprises:

a read shift clock coupled to said read extractor means and said shift register means for generating a signal to cause said shift register means to shift so as to be loaded with data based on said plurality of serial pulses;

a load read data register logic circuit, coupled to said shift clock, said shift register means and said register means, which sends a signal to said register means when prior data in said register means had been received by said computer as determined by a bit in said register means;

a hold read data register logic circuit coupled to said register means and a buffer means, said buffer means also being coupled to said register means, said hold read data register logic circuit sending a signal to said buffer means after a predetermined period of time which is long enough to ensure that data in said buffer means has been properly transferred to said computer, said predetermined period of time being based upon the timing of said computer as determined by said clock signal from said computer.

7. The controller defined by claim 1 wherein said write control means comprises:

register means for storing parallel data from said computer to be sent to said disk drive;

shift register means coupled to said register means for converting said parallel data into a serial bit stream;

toggle means coupled to said shift register means for generating pulses representing binary '1's and binary '0's which are sent to said disk drive; and

write data control means for controlling the loading of data from said computer into said register means, said shift register means, and said toggle means, to ensure that data sent to said disk drive is not lost and is not duplicated.

8. The controller defined by claim 7 wherein said write data control means comprises:

a write shift clock coupled to said shift register means;

a load and shift register logic circuit, coupled to said shift register means and said write shift clock, which sends a signal to said shift register means causing the shift register means to load data from said register means and shift data which has been previously loaded; and

a handshake/underrun logic circuit coupled to said load and shift register logic circuit, and said write shift clock for generating signals to inform said computer when said register means is ready to receive additional data from said computer.

* * * * *

United States Patent [19]

Dhuey et al.

[11] **Patent Number: 4,774,652**

[45] **Date of Patent: Sep. 27, 1988**

[54] **MEMORY MAPPING UNIT FOR DECODING ADDRESS SIGNALS**

[75] Inventors: **Michael J. Dhuey**, Cupertino; **Ronald R. Hochsprung**, Saratoga, both of Calif.

[73] Assignee: **Apple Computer, Inc.**, Cupertino, Calif.

[21] Appl. No.: **15,907**

[22] Filed: **Feb. 18, 1987**

[51] **Int. Cl.**⁴ **G06F 12/00**
[52] **U.S. Cl.** **364/200**
[58] **Field of Search** 364/200, 900

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,484,263	11/1984	Olson et al.	364/200

FOREIGN PATENT DOCUMENTS

0132129	7/1984	European Pat. Off. .
0205692	6/1985	European Pat. Off. .
0223551	11/1986	European Pat. Off. .

OTHER PUBLICATIONS

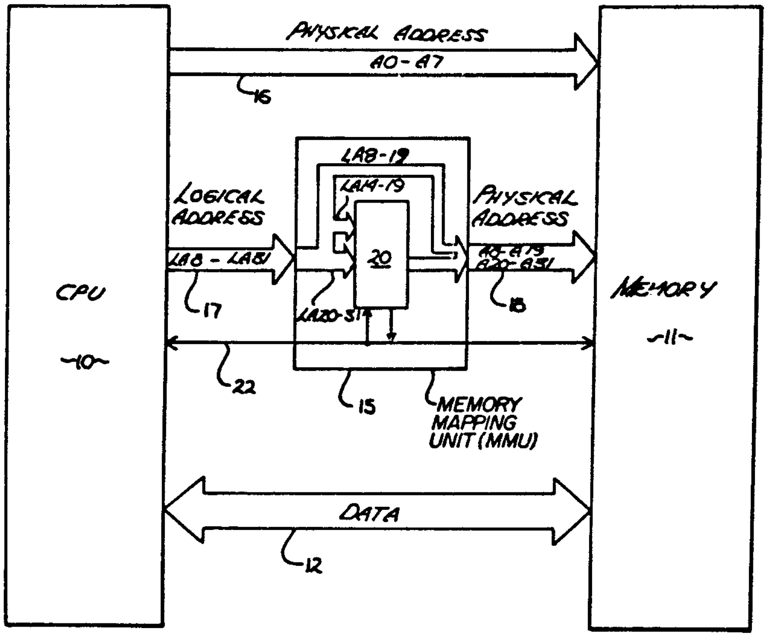
"MC68851 Page Memory Management Unit User's Manual"; Motorola; Prentice-Hall; 1986.

Primary Examiner—Raulfe B. Zache
Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman

[57] **ABSTRACT**

A memory mapping unit which permits a computer to run programs designed to provide 32-bit or 24-bit address signals to address a 32-bit addressable memory. When a CPU generates a 32-bit address, that address is passed through to provide a 32-bit physical address. However, when the CPU generates a 24-bit address, the most significant bits are processed by the memory mapping unit to provide a remapped 32-bit physical address. The memory mapping unit is implemented on a single semiconductor chip using gate-array technology.

16 Claims, 5 Drawing Sheets



Macintosh 2 HMMU

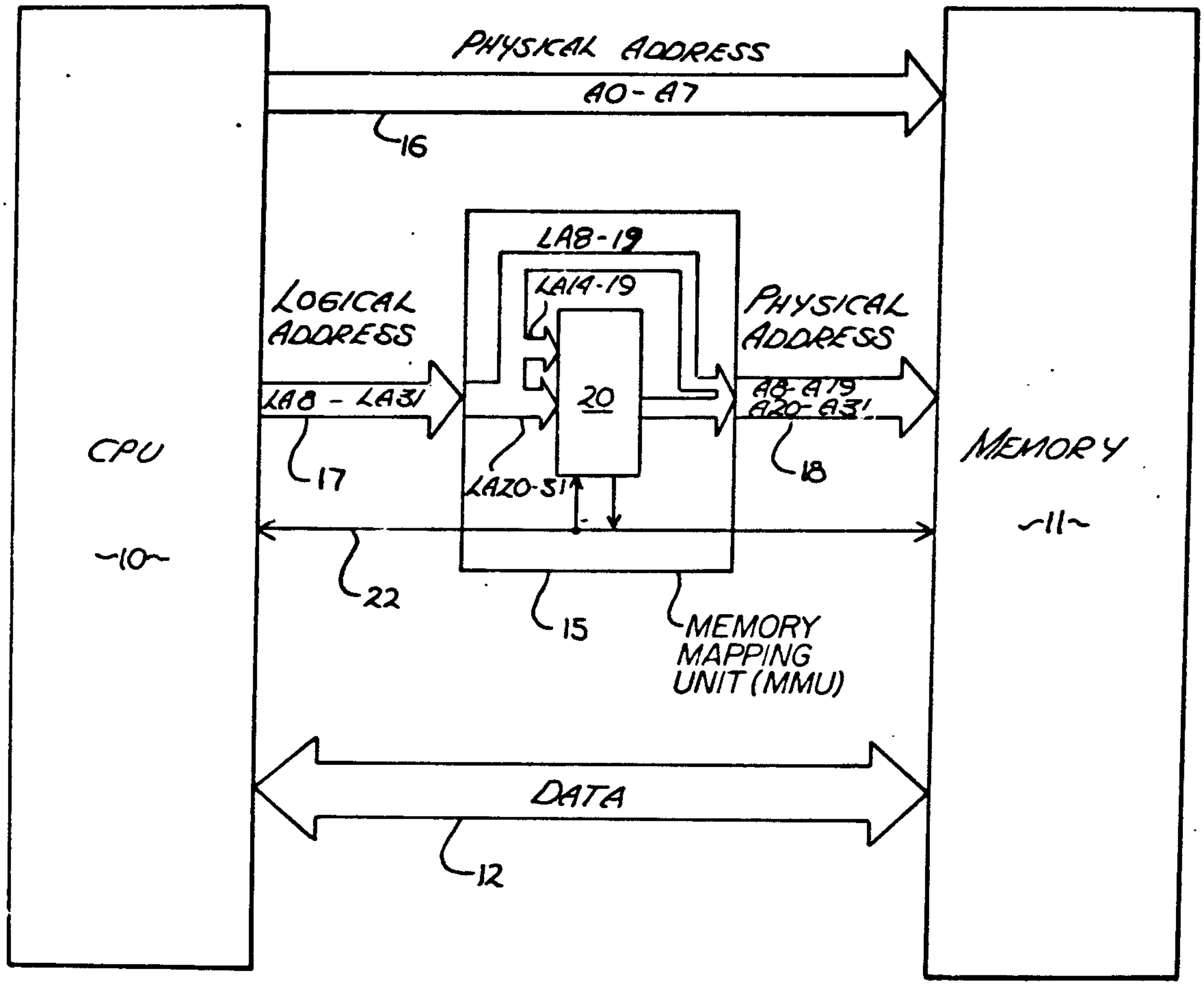
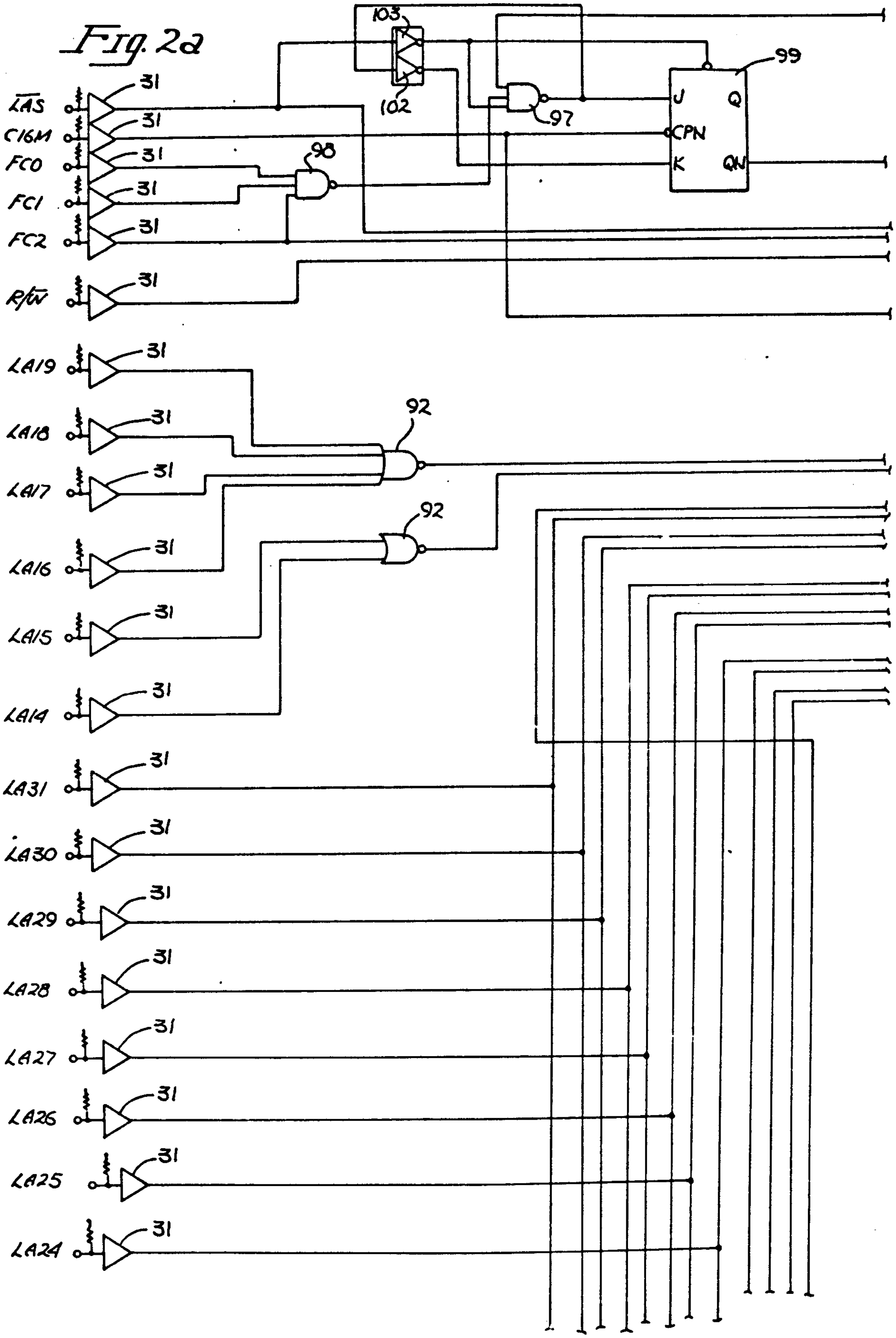


Fig. 1

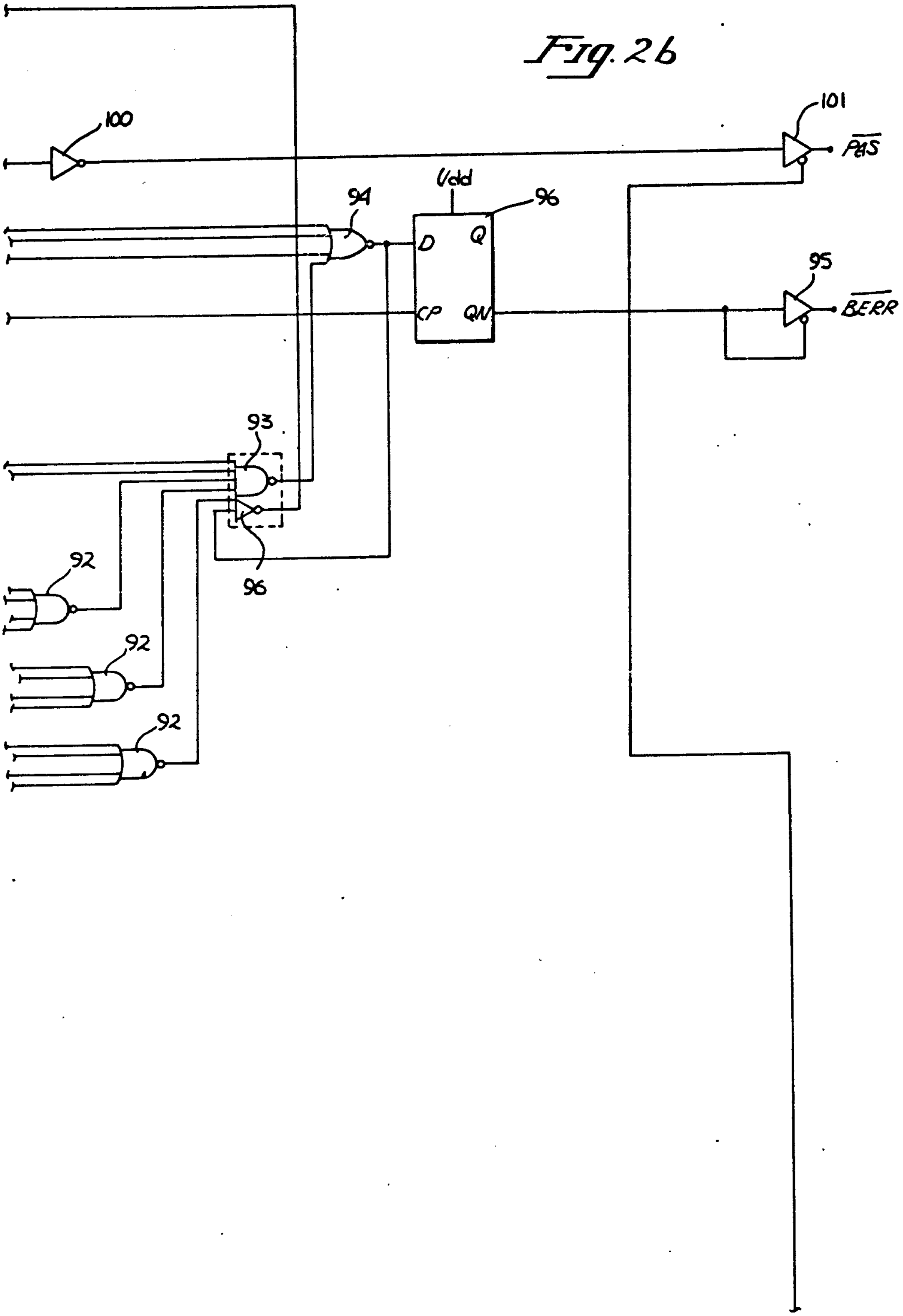
“APPLE_PAT_4_774_652_02” 84 KB 2000-02-22 dpi: 300h x 300v pix: 1911h x 2311v

Fig. 2a



“APPLE_PAT_4_774_652_03” 159 KB 2000-02-22 dpi: 300h x 300v pix: 1948h x 3036v

Fig. 2b


100
101
$\overline{PAS}$
Vdd
94
96
D
Q
CP
QN
95
$\overline{BERR}$
93
96
92
92
92

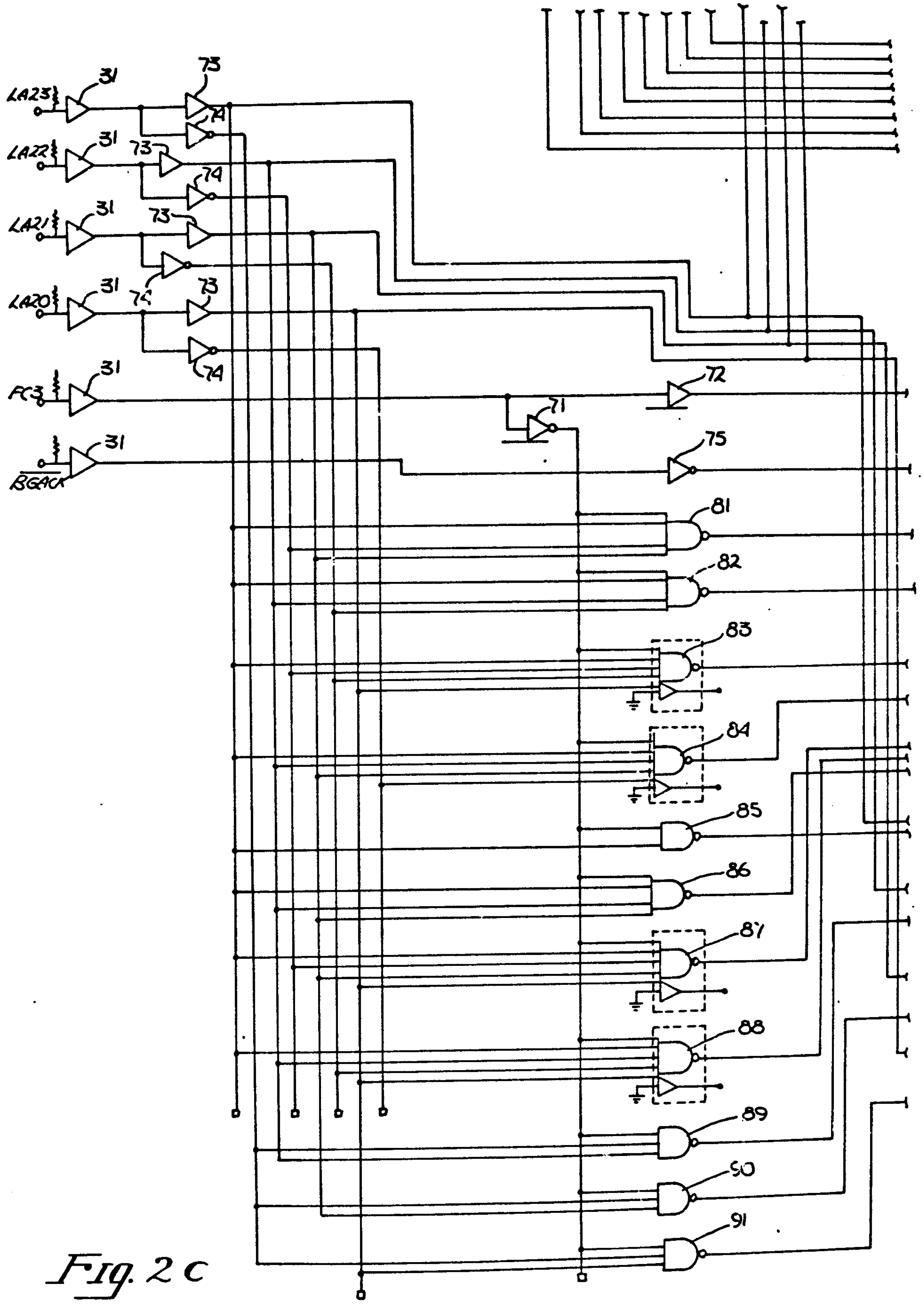


Fig. 2c

“APPLE_PAT_4_774_652_05” 194 KB 2000-02-22 dpi: 300h x 300v pix: 2009h x 2999v

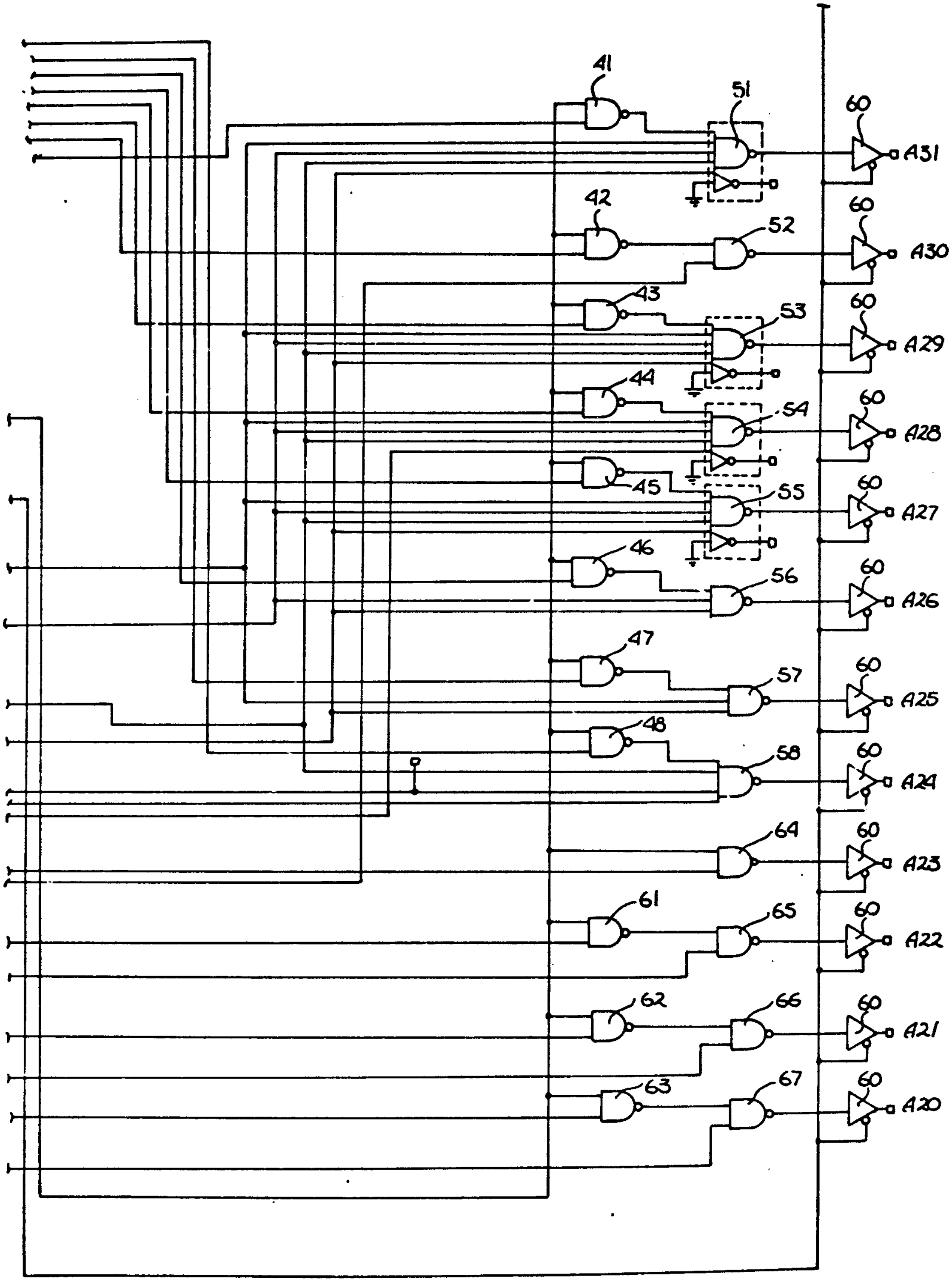

41
51
60
A31
42
52
60
A30
43
53
60
A29
44
54
60
A28
45
55
60
A27
46
56
60
A26
47
57
60
A25
48
58
60
A24
64
60
A23
61
65
60
A22
62
66
60
A21
63
67
60
A20

Fig. 2d

MEMORY MAPPING UNIT FOR DECODING ADDRESS SIGNALS

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to the field of computer memory management units, and more specifically, to mapping less than 32 bits onto a 32-bit memory address bus.

2. Prior Art

In most computers, a central processing unit (CPU) communicates directly with both an address bus and a data bus. These buses are coupled to a memory system in addition to numerous other items, such as input/output ports, specialized processors, DMA units, etc. The new generation of microcomputers of today utilized single chip CPUs such as the 8086, 80386, 68000 and 68020. More recent chips, such as the 80386 and the 68020 utilize 32 bit address signals to access various locations within the memory.

Memory management units are well-known in the prior art and are used to provide efficient utilization of the computer's main memory. These units perform housekeeping functions, such as remapping, and often include a memory which stores data containing relocation of an address base and providing paging functions. Because of the complexity of present day CPU chips, more complex memory management units are provided to perform extensive and complicated memory management functions. One such chip is the 68851 paged memory management unit by Motorola Inc. to support the 68020 chip.

However, until the advent of the 32-bit microprocessor chips the earlier CPUs operated on 16-bit and 24-bit addressing schemes. Considerable software, including operating systems, have been written to run on these prior art 16-bit and 24-bit computers. Prior art memory management units operating in conjuction with these earlier CPUs are not able to provide the extended addressing bit capability of the new 32-bit processors. Although 32-bit memory management units are available, such as the aforementioned 68851, such units are very complex, costly and provide significantly more complex functions than the basic memory remapping which is required to convert the prior art 16-bit and 24-bit address ranges into a 32-bit address map.

For example, the Macintosh™ computer sold by Apple Computer Inc. of Cupertino, Calif., provided a 24-bit address scheme, wherein 24 bits physically addressed the memory. A newer computer operating on the 68020 CPU is now capable of addressing considerable more memory space due to its 32-bit address configuration. However, to run the earlier software written for the 24-bit machine, the 32-bit system must be capable of converting the 24-bit address range of the older system to a 32-bit address range of the newer computer system. Although other memory management units are capable of performing this function, the memory remapping of 24 to 32 bits can be accomplished much more simply and economically with the present invention.

The present invention builds upon those prior art memory management units, as well as the more recent 32-bit memory management units. The memory mapping unit of the present invention is simplistic in design and function and is economical from a cost stand point. The memory mapping unit of the present invention is capable of providing a 32-bit address range to physically access the memory by using the more recent 32-bit addressing scheme, or alternatively, converting the prior art address scheme having less than 32 bits to a 32-bit address.

SUMMARY OF THE INVENTION

The present invention describes an apparatus for permitting a 32 address bit CPU and main memory to run 24 address bit programs. A memory mapping unit (MMU) of the present invention is placed between the CPU and the main memory. Whenever the CPU runs programs designed to generate 32-bit addresses for accessing main memory, the MMU permits the address signal to pass without remapping. However, whenever the CPU runs programs designed to generate 24-bit addresses, the MMU converts the 24 bits and provides a 32-bit physical address signal. The remapping is essential because equivalent tasks of each program need to access equivalent areas of memory.

The MMU as implemented in the preferred embodiment processes the four most significant bits of the 24-bit address signal and generates the twelve most significant bits of the 32-bit physical address signal. Further, the processing is achieved using combinatorial logic implemented in gate-array technology.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block digram showing the various disposition of address signals processed by the memory mapping unit of the present invention.

FIG. *2a* is a schematic diagram showing the upper left quadrant of a mapping circuit of the preferred embodiment.

FIG. *2b* shows the upper right quadrant of the mapping circuit.

FIG. *2c* shows the lower left quadrant of the mapping circuit.

FIG. *2d* shows the lower right quadrant of the mapping circuit.

DETAILED DESCRIPTION OF THE INVENTION

A memory mapping unit is described for use in a digital computer which includes a central processing unit (CPU) and a main memory. In the following description, numerous specific details are set forth such as specific memory sizes, part numbers, circuits, etc., in order to provide a thorough understanding of the present invention. However, it will be obvious to one skilled in the art that the present invention may be practiced without these specific details. In other instances, well-known structures and circuits are not described in detail in order not to unnecessarily obscure the present invention.

Referring to FIG. 1, the memory mapping scheme of the present invention is shown. A CPU 10 is coupled to access memory 11. A data bus 12 couples the data between CPU 10 and memory 11. The memory 11 is arranged such that it is mapped by a 32-bit address signal from CPU 10. The memory mapping unit (MMU) 15 of the present invention is located to accept logical address signals from CPU 10 and to provide physical address signals to memory 11. The CPU 10 of the preferred embodiment is a Motorola 68020 CPU chip which provides a 32-bit address signals. The eight lower significant bits (LSBs) are passed directly to memory 11 on address bus 16 to provide the 8 LSBs for the physical address signal. The other 24 bits are provided from the CPU 10 as logical address on logical address bus 17.

Bus 17 is coupled to MMU 15, wherein the 24 address bits, LA8–LA31, are split into two paths. Bits LA8–LA19 are directly coupled to provide physical address bits A8–A19 on physical address bus 18. Bits LA14–LA31 are coupled to mapping circuit 20, wherein circuit 20 provides the 12 most significant bits (MSBs) A20–A31 of the physical address signal on bus 18. Although bus 16 and 18 are shown as two separate buses in FIG. 1, in actuality buses 16 and 18 comprise a single 32-bit physical address bus for accessing memory 11. Various other lines are coupled between CPU 10, MMU 15 and memory 11, and are shown by a single line 22 in FIG. 1 for the purpose of simplicity. Clocking signals, function code signals, bus acknowledge signals, strobing signals and other control signals are included as part of line 22.

In operation the lower order 8 bits, A0–A7, are coupled to memory 11 without transitioning through MMU 15. The other 24 address bits from CPU 10 are coupled to MMU 15 on bus 17. Physical address bus 18 provides address bits A8–A31, which when combined with address signals A0–A7 provide the 32 bits needed to access memory 11. When CPU 10 is processing software which was written to provide 32 bits of addressing, circuit 20 will accept LA20–LA31 and pass the 12 bits through to provide address bits A20–A31. Therefore, in the 32-bit mode all 32 bits from the CPU 10, A0–A7 and LA8–LA31, are coupled straight through to memory 11 as physical address signals A0–A31 permitting the CPU 10 to provide the physical address of memory 11.

When executing earlier Macintosh TM computer software, only the 24 LSBs of the 32-bit address field emanating from CPU 10 contain useful information. The 8 MSBs, LA24–LA31, are not relevant to the address field, and hence, are ignored by the MMU 15. The 8 LSBs are generated directly onto bus 16 as address signals A0–A7. The other meaningful address bits, LA8–LA23 are inputted into MMU 15. MMU 15 passes LA8–LA19 directly as before to bus 18 as A8–A19. The MSBs of the 24-bit address, LA20–LA23, are remapped by circuit 20 to provide the 12 physical address signals A20–A31. Therefore, in the 24-bit mode, the lower 20 address bits are passed directly through to memory 11 as address signals A0–A19 and the upper 12 bits from CPU 10 are converted and remapped by circuit 20 to provide address signals A20–A31.

The mapping scheme of mapping a 24-bit address range to a 32-bit address range to access memory 11 as used in the preferred embodiment is shown below:

TABLE 1

24 bit address range		32 bit address range	
$xx00 0000	$xx7F FFFF	$0000 0000	$007F FFFF
$xx80 0000	$xx8F FFFF	$4000 0000	$400F FFFF
$xx90 0000	$xx9F FFFF	$F900 0000	$F90F FFFF
$xxA0 0000	$xxAF FFFF	$FA00 0000	$FA0F FFFF
$xxB0 0000	$xxBF FFFF	$FB00 0000	$FB0F FFFF
$xxC0 0000	$xxCF FFFF	$FC00 0000	$FC0F FFFF
$xxD0 0000	$xxDF FFFF	$FD00 0000	$FD0F FFFF
$xxE0 0000	$xxEF FFFF	$FE00 0000	$FE0F FFFF
$xxF0 0000	$xxFF FFFF	$5000 0000	$500F FFFF

The addresses are shown depicted in hexidecimal code, such that each digit is represented by 4 address bits. For example, a 24-bit address of $90 0000 is converted by the MMU 15 to a 32-bit address $F900 0000. It should be noted that in the remapping scheme the lower five digits, which determine address bits A0–A19

are never changed when remapped to the 32-bit address range. Only the most significant digit (the 4 MSBs) of the 24-bit address is converted to provide bits A20–A31, when remapped to the 32-bit physical address. For example, a 24-bit address of $B0 0000 is converted to a 32-bit address by remapping the digit B to provide the three most significant digits FB0 of the 32-bit address. For an address of $BF FFFF, the digit B is still remapped to digits FBO to provide a physical address of $FB0F FFFF. An address between $B0 0000 and $BF FFFF is remapped linearly between $FB00 0000 and $FBOF FFFF. The other address ranges are remapped equivalently. In functional terms, circuit 20 when in the 24-bit mode will take the most significant digit of the 24 bit address, LA20–LA23, and generate a new three digit, 12 MSBs, of the 32-bit physical address.

Referring again to Table 1, the remapping scheme of remapping the 24-bit physical address space of the earlier 24-bit software into a 32-bit physical address space is arbitrary and is left to the designer. However, the remapping scheme of the preferred embodiment as shown in Table 1 is designed specifically, such that portions of the memory allotted to a particular task in the earlier version of the Macintosh TM computer is mapped to an equivalent memory space in the memory of the newer computer system using the 32-bit memory address. Obviously, because of the size of memory 11, there will be excess memory space when the 24-bit physical address space is mapped into memory 11. Although a particular addressing scheme using a 24-bit to a 32-bit conversion is described, it is appreciated that oher remapping address schemes, including conversion of other than 24 bits, can be practiced without departing from the spirit and scope of the invention.

Referring to FIGS. 2*a–d*, a circuit schematic of circuit 20 of FIG. 1 is shown. Various logical address signals LA14–LA31, as well as various control signals, are shown as inputs to input buffers 31. The upper signals LA14–LA19, R/$\overline{W}$, FC0–FC2, $\overline{LAS}$, and clocking signal C16M are utilized to develop $\overline{PAS}$ and $\overline{BERR}$ signals, and are not pertinent to the address conversion provided by the MMU 15. The actual address translation is provided by a portion of the circuit associated with input signals LA20–LA31 and function code signal FC3.

The $\overline{BGACK}$ is a bus acknowledge signal, which is not actually used for ramapping, but is necessary for activating output buffers 60 and 101. A 68020 users manual can be consulted for a precise purpose of the various CPU signals described above. The 24 or the 32-bit mode of the circuit 20 is controlled by the state of signal FC3. Whenever FC3 is low, the MMU 15 operates to transfer the 32 bits from the CPU straight through as physical address to memory 11. Whenever FC3 is high, MMU 15 is in its 24 bit mode and signals LA20–LA23 representing the MSBs of a 24-bit address signal is used to provide a remapped 32-bit address signal.

When in the 32-bit mode, signals LA24–LA31 are coupled to NAND gates 41–48, which outputs are coupled to NAND gates 51–58. The outputs of NAND gates 51–58 are each coupled to its respective tristate output buffers 60. When in the 32-bit mode, FC3 is high, placing a high on second input of each of NAND gates 41–48 such that the outputs of gates 41–48 will be determined by the state of the signals LA24–LA31. FC3 signal is inverted by inverter 71, which output is coupled to an input of various NAND gates 81–91. The output of NAND gates 81-91 are coupled to various input of NAND gates 51-58 and 65-67 as shown in the schematic. Signals LA23-LA20 are coupled through buffer 73 to AND gate 64 and NAND gates 61-63, respectively. In the 32-bit mode, signals LA20-LA31 are coupled through dual NAND gate configuration, such as NAND gates 41 and 51, or through a single AND gate, such as for LA23, wherein the state of the signals LA20-LA31 are unchanged as they are outputted as A20-A31.

The outputs of gates 51-58 and 64-67 are each passed through its respective tristate output buffer 60 to provide address signals A20-A31. Buffers 60 have their tristate enable line coupled to $\overline{BGACK}$ signal through inverter 75. As long as signal $\overline{BGACK}$ remains high, tristate buffers 60 are enabled to provide an output. However, when $\overline{BGACK}$ goes low the tristate buffers 60 are placed in its tristate position and circuit 20 is decoupled from providing address signals on lines A20-A31.

When operating in the 24-bit mode, function code FC3 goes low and a low state is placed on the input of gates 41-48 and 61-64, such that the output of these gates 41-48 and 61-63 remains high and the output of gate 64 remains low causing LA20-LA31 from transitioning to the output A20-A31. Therefore, signals on LA24-LA31 are basically decoupled from passing through circuit 20, due to the operation of gates 41-48 and 61-64. In the 24-bit mode, where bits 24-31 represented by LA24-LA31 are non-functional bits, they are excluded from the operation of the circuit 20.

Signals LA20-LA23, which represent the most significant digit of a 24-bit address and which provide the remapping in the preferred embodiment, are used for the necessary decoding to generate remapped signals A20-A31. Buffer 73s and inverters 74 couple signals LA20-LA23 to corresponding NAND gates 81-91 as shown on the schematic to provide the necessary decoding for the address translation. FC3 is low in this instance such that the output of inverter 71 is at a high state, permitting gates 81-91 to respond to various inputs coupling signals LA20-LA23. Outputs of gates 81-91 are coupled to NAND gates 51-58 and 65-67, wherein further address translation is provided in gates 51-58 such that LA20-LA23 of a 24-bit address signal is converted to provide A20-A31 of a 32-bit physical address signal to address memory 11 of FIG. 1.

It should be pointed out that a single AND gate 64 is used for the generation of address signal A23 in the preferred embodiment, because address signal A23 will be 0 whenever converting 24 bits to 32 bits. This is done in the preferred embodiment because, as can be seen in Table 1, the largest value encountered by the sixth digit during the remapping is a value of 7. However, address line A23 could be implemented using two NAND gates and appropriate decoding as is the case with the other address signals. Therefore, in the 24-bit mode, input lines LA24-LA31 are disregarded and signals LA20-LA23 are used to provide the decoding for generating the most significant 12 bits A20-A31 by gates 81-91, 51-58 and 64-67.

The remaining portion of circuit 20 is not required for the remapping described above, but does take advantage of the 32-bit line from the CPU 10 to provide certain user functions. The twelve input address signals LA20-LA31, as well as address signals LA14-LA19, are coupled through various NOR gates 92 and the output of NOR gates 92 are coupled to a five input

NAND gate 93, such that only when all inputs LA14-LA31 are low the output of NAND gate 93 will be low. The output of NAND gate 93 is coupled to an input of NOR gate 94 along with FC2, $\overline{LAS}$ and R/$\overline{W}$ signals, such that whenever signals LA14-LA31 are all low, FC3 is in user mode (low state) and a write is attempted, a bus error signal $\overline{BERR}$ will result at the output of buffer 95.

Flip-flop 96 is a D-type flip-flop coupling the output of NOR gate 94 to buffer 95. Output of NOR gate 94 is coupled through inverter 96 to an input of NAND gate 97, which also has its input an inverted $\overline{LAS}$ signal (LAS), as well as a combination of FC0-FC2 coupled through NAND gate 98. Gate 97 provides an output to a J input of a J-K flip-flop 99. The $\overline{Q}$ output of J-K flip-flop 99 is coupled through inverter 100 and then through buffer 101 to provide a $\overline{PAS}$ signal. Buffer 101 is a tristate buffer, wherein the enable line is coupled to the same enable line as buffers 60. Output of NAND gate 97 sets the J-K flip-flop 99 and provides a low on the $\overline{Q}$ output of the flop-flop 99. The output of NAND gate 97 is also coupled to the K input of flip-flop 99 through inverter 102 to reset flip-flop 99. Clock signal C16M is coupled to clock inputs of flip-flop 99 and 96 for sychronizing these two flip-flops. The purpose of flip-flop 99 is to simply convert the $\overline{LAS}$ signal to a PAS signal at the output of buffer 101.

As stated earlier the upper portion of circuit 20 is nonmaterial to the operation of the address translation of converting 24-bit address space into a 32-bit address space. Further, circuit 20 of the preferred embodiment is structured using known gate array techniques, such that circuit 20 is embodied in a single semiconductor chip. However, other configurations and techniques, not necessarily gate arrays, can be used to provide the address translation of the present invention without departing from the spirit and scope of the present invention. Also, circuit 20 shows other components, such as pull-up resistors associated with buffers 31, unused inverters (shown within the dotted lines associated with gates, such as gate 51), and pads for coupling various signals to and from the chip, but these features are well-known in the art and do not add to the teaching of the present invention.

Thus, a memory mapping unit for a computer is described.

We claim:

1. In a computer system which includes a central processing unit (CPU) for operating on programs of varying bit length addressing fields, a computer main memory, and a memory mapping unit (MMU), said MMU coupled to said CPU and said main memory, said MMU comprising:

input means coupled to accept a CPU address signal from said CPU;

output means coupled to provide a physical address signal to address said main memory;

decoding means coupled to said input means and said output means for translating said CPU address signal;

switching means coupled to said input means and said decoding means for switching in said decoding means, wherein during a first mode said CPU address signal has a first bit length address field and is passed through to provide said physical address signal and during a second mode said decoding means is switched in to convert said CPU address signal which has a second bit length address field to said physical address signal;

wherein two equivalent program instructions, but each having different bit length address fields, access identical areas of said memory.

2. The MMU defined in claim 1, wherein a portion of said CPU address signal is coupled to said decoding means and said portion of said CPU address signal is processed by said decoding means to provide said physical address signal.

3. The MMU defined in claim 2, wherein said portion of said CPU address signal is processed to provide a portion of said physical address signal.

4. The MMU defined in claim 3, wherein said decoding means further including a plurality of gates to provide combinatorial logic for processing said CPU address signal.

5. The MMU defined in claim 4 being implemented in a gate-array semiconductor chip.

6. In a computer system which includes a CPU for operating on a program which provides a CPU address signal having a first bit length field and also operating on another program which provides said CPU address signal having a second bit length field which length is shorter than said first bit length field, a computer main memory being accessed by a physical address signal having said first bit length field, and a memory mapping unit (MMU) coupled to said CPU and said main memory for accepting said CPU address signal and providing said physical address signal, sand MMU comprising:

input means coupled to accept said CPU address signal from said CPU;

output means coupled to provide said physical address signal to address said main memory;

decoding means coupled to said input means and said output means for translating said second bit length field to said first bit length field;

switching means coupled to said input means and said decoding means and under control of a control signal from said CPU; wherein during a first mode said CPU address signal, having a first bit lenght field, is passed through to provide said physical address; and during a second mode when said CPU is providing a second bit length field, said decoding means is switched in to convert said second bit length field to said first bit length field, which is then used as said physical address signal to access said memory;

wherein remapping of said CPU address signal is achieved during said second mode.

7. The MMU defined in claim 6, wherein during said second mode, a portion of said second bit length field is processed by said decoding means to provide said physical address signal.

8. The MMU defined in claim 7, wherein said decoding means including a set of gate-arrayed combinatorial logic.

9. The MMU defined in claim 8, wherein said switching means including a second set of gate-arrayed combinatorial logic.

10. The MMU defined in claim 9 being implemented in a semiconductor chip.

11. In a computer system which includes a CPU for generating a 32-bit CPU address signal, a computer main memory being accessed by a 32-bit physical address signal, and a Memory Mapping Unit (MMU) coupled to said CPU and said main memory for accepting said CPU address signal and providing said physical address signal, said CPU for operating programs generating either a 32-bit address or a 24-bit address, said MMU comprising;

input means coupled to accept said CPU address signal;

output means coupled to provide said physical address signal to address said main memory;

decoding means coupled to said input means and said output means for translating a 24-bit CPU address signal to a 32-bit physical address signal;

switching means coupled to said input means and said decoding means for switching in said decoding means under control of a control signal from said CPU; wherein during a first mode a 32-bit CPU address signal is passed through to provide said physical address and during a second mode said decoding means is switched in to convert said 24-bit CPU address to said 32-bit physical address;

wherein a 24-bit physical memory space is mapped into a 32-bit physical memory space during said second mode.

12. The MMU defined in claim 11, wherein twelve most significant bits (MSBs) of said 32-bit CPU signal are processed by said MMU and twenty least significant bits (LSBs) of said 32-bit CPU signal are coupled directly to provide twenty LSBs of said physical address signal.

13. The MMU defined in claim 12, wherein said 12 MSBs are passed through to provide 12 MSBs of said physical address signal during said first mode; but during said second mode, only four bits of said 12 MSBs contain address information and said four bits are processed by said decoding means to generate a remapped 12 MSBs of said physical address signal.

14. The MMU defined in claim 13, wherein said decoding means is comprised of combinatorial logic formed using gate-array technology.

15. The MMU defined in claim 14, wherein said switching means is comprised of combinatorial logic formed using gate-array technology.

16. The MMU defined in claim 15 being implemented in a semiconductor chip.

* * * * *

United States Patent [19]

Ashkin et al.

[11] **Patent Number: 4,875,158**

[45] **Date of Patent: Oct. 17, 1989**

[54] **METHOD FOR REQUESTING SERVICE BY A DEVICE WHICH GENERATES A SERVICE REQUEST SIGNAL SUCCESSIVELY UNTIL IT IS SERVICED**

[75] Inventors: **Peter B. Ashkin**, Los Gatos; **Michael Clark**, Glendale, both of Calif.

[73] Assignee: **Apple Computer, Inc.**, Cupertino, Calif.

[21] Appl. No.: **132,121**

[22] Filed: **Dec. 14, 1987**

Related U.S. Application Data

[62] Division of Ser. No. 765,396, Aug. 14, 1985.

[51] **Int. Cl.**^{4} **G06F 13/14; G06F 13/38**

[52] **U.S. Cl.** **364/200; 364/222;** 364/240.8; 364/241; 364/245; 364/262.3; 364/284.3; 340/825.08; 340/825.52

[58] **Field of Search** ... 364/200 MS File, 900 MS File; 340/825.06, 825.07, 825.08, 825.5, 825.52

[56] **References Cited**

U.S. PATENT DOCUMENTS

3,221,307	11/1965	Manning	364/200
3,646,534	2/1972	Miller	360/40
3,715,725	2/1973	Kievit et al.	364/900
3,836,888	9/1974	Boenke et al.	364/200
3,863,025	1/1975	Gonsewski et al.	375/55
3,979,723	9/1976	Hughes et al.	370/31
4,063,220	12/1977	Metcalfe et al.	340/825.5 X
4,071,908	1/1978	Brophy et al.	364/900

(List continued on next page.)

FOREIGN PATENT DOCUMENTS

0051425	5/1982	European Pat. Off. .
0104542	4/1984	European Pat. Off. .
59-52331	3/1984	Japan .
1508854	4/1978	United Kingdom .
1518565	7/1978	United Kingdom .
2035636	6/1980	United Kingdom .
2070826	5/1984	United Kingdom .
0143160	6/1985	United Kingdom .
2167274	5/1986	United Kingdom .
0207313	1/1987	United Kingdom .

OTHER PUBLICATIONS

Hill et al., "Dynamic Device Address Assignment Mechanism", IBM Technical Disclosure Bulletin, vol. 23, No. 8, Jan. 1981, pp. 3564-3565.

Search Report, Dated May 21, 1986, for British Patent Application No. 8607632.

Primary Examiner—Raulfe B. Zache

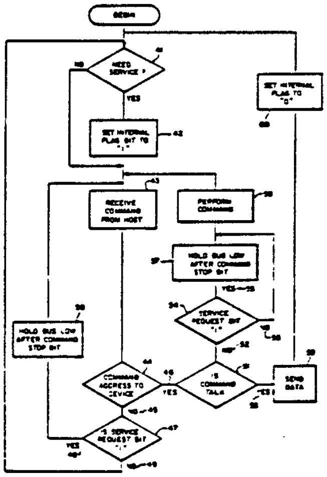
Assistant Examiner—Thomas C. Lee

Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman

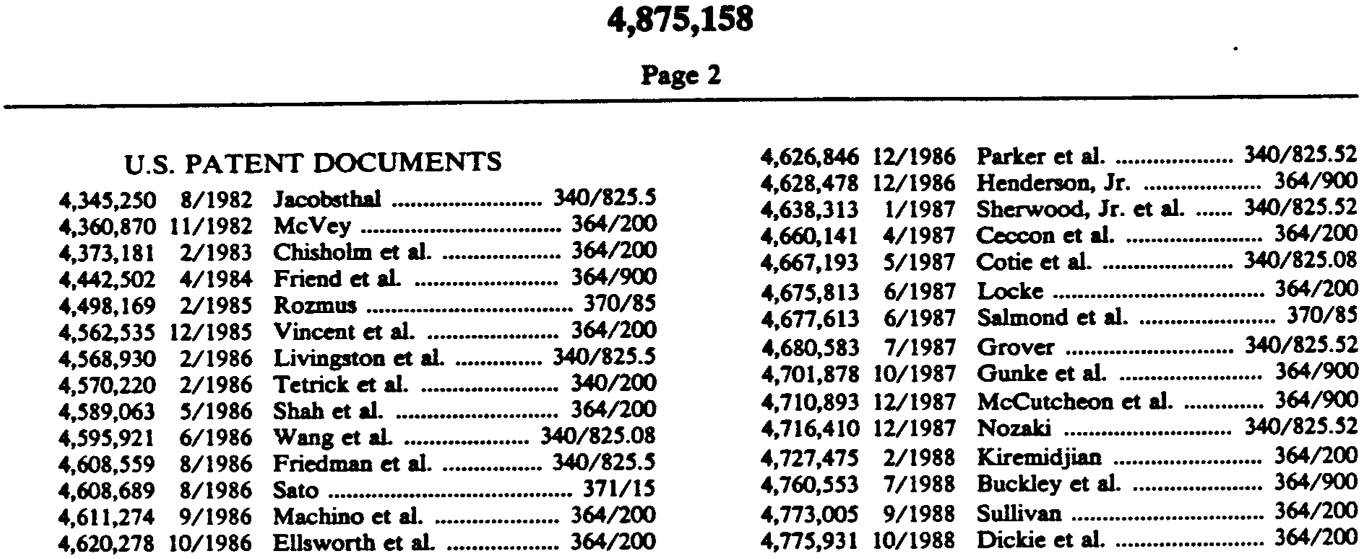
[57] **ABSTRACT**

A method for requesting service by a device coupled to a host computer through a communication medium. The host computer sets a service request bit of the device to a first logical value to allow the device to produce a service request signal if the device requires servicing. The device determines that it requires servicing and the device sets an internal flag bit to a first logical value to indicate that the device requires servicing. The device monitors a command from the host computer to see if the command is addressed to the device. If the command is not addressed to the device and if the service request bit is set to a first logical value, then the device generates a service request signal on the medium after the command by holding the communication medium low for a first period of time. If the command is addressed to the device, if the device determines that the command is not a command that services the device, and if the service request bit is set to the first logical value, then the device generates a service request signal on the medium after the command by holding the medium low for the first period of time and performs the command. If the command is addressed to the device, if the device determines that the command is not a command that services the device, and if the service request bit is not set to the first logical value, the device performs the command without generating the service request signal. The steps including and following the step of the device monitoring a command from the host computer are repeated until the device receives a command addressed to the device that services the device.

4 Claims, 4 Drawing Sheets


BEGIN
NEED SERVICE ?
41
NO
YES
SET INTERNAL FLAG BIT TO "1"
42
SET INTERNAL FLAG TO "0"
60
43
RECEIVE COMMAND FROM HOST
PERFORM COMMAND
58
HOLD BUS LOW AFTER COMMAND STOP BIT
57
YES
SERVICE REQUEST BIT "1"
54
NO
56
NO
52
IS COMMAND TALK
51
YES
SEND DATA
59
HOLD BUS LOW AFTER COMMAND STOP BIT
50
COMMAND ADDRESS TO DEVICE
44
46
YES
NO
45
IS SERVICE REQUEST BIT "1"
47
YES
48
NO
49

Macintosh ADB (?) (Apple Desktop Bus)



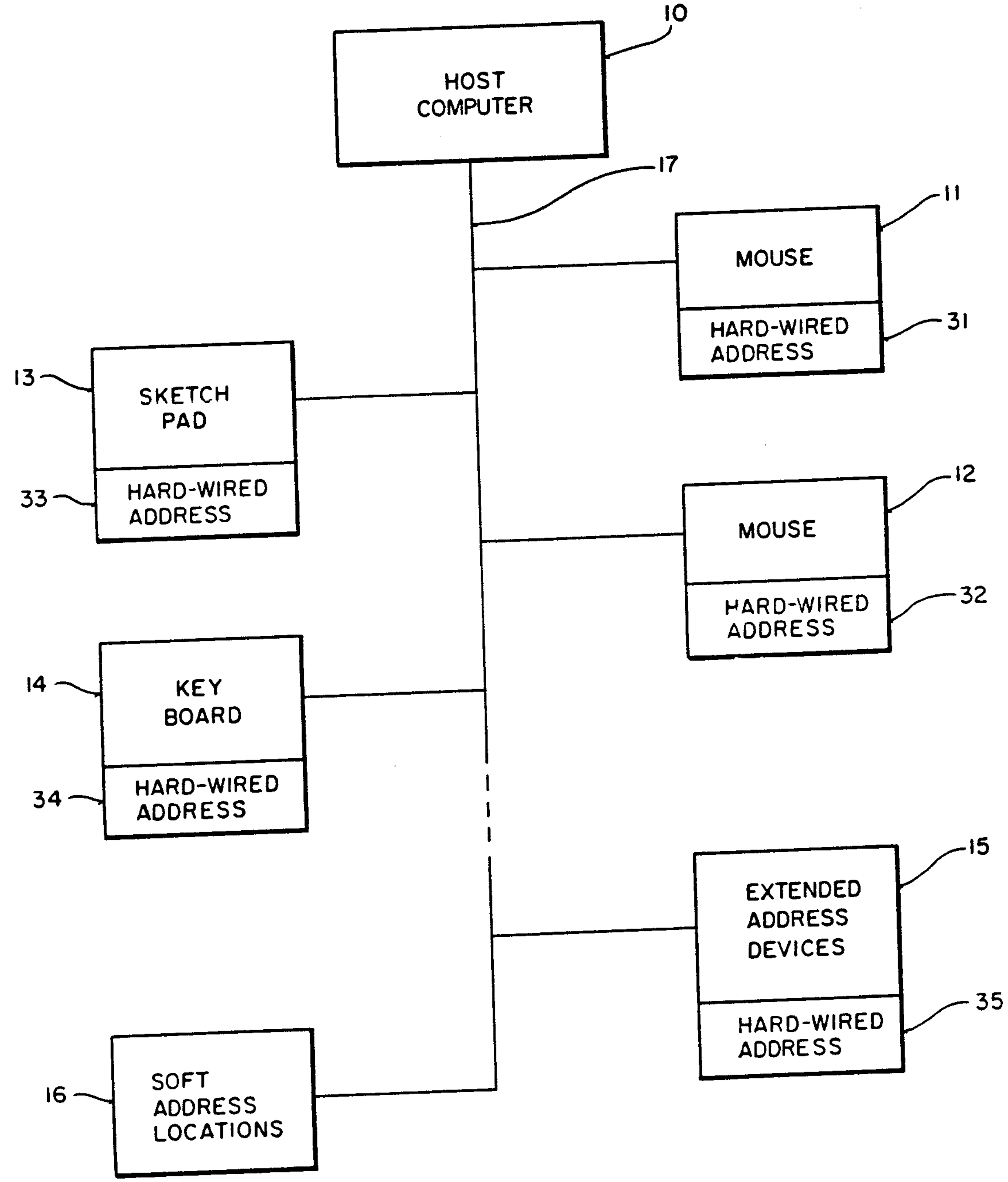
4,875,158

Page 2

U.S. PATENT DOCUMENTS

4,345,250	8/1982	Jacobsthal	340/825.5
4,360,870	11/1982	McVey	364/200
4,373,181	2/1983	Chisholm et al.	364/200
4,442,502	4/1984	Friend et al.	364/900
4,498,169	2/1985	Rozmus	370/85
4,562,535	12/1985	Vincent et al.	364/200
4,568,930	2/1986	Livingston et al.	340/825.5
4,570,220	2/1986	Tetrick et al.	340/200
4,589,063	5/1986	Shah et al.	364/200
4,595,921	6/1986	Wang et al.	340/825.08
4,608,559	8/1986	Friedman et al.	340/825.5
4,608,689	8/1986	Sato	371/15
4,611,274	9/1986	Machino et al.	364/200
4,620,278	10/1986	Ellsworth et al.	364/200
4,626,846	12/1986	Parker et al.	340/825.52
4,628,478	12/1986	Henderson, Jr.	364/900
4,638,313	1/1987	Sherwood, Jr. et al.	340/825.52
4,660,141	4/1987	Ceccon et al.	364/200
4,667,193	5/1987	Cotie et al.	340/825.08
4,675,813	6/1987	Locke	364/200
4,677,613	6/1987	Salmond et al.	370/85
4,680,583	7/1987	Grover	340/825.52
4,701,878	10/1987	Gunke et al.	364/900
4,710,893	12/1987	McCutcheon et al.	364/900
4,716,410	12/1987	Nozaki	340/825.52
4,727,475	2/1988	Kiremidjian	364/200
4,760,553	7/1988	Buckley et al.	364/900
4,773,005	9/1988	Sullivan	364/200
4,775,931	10/1988	Dickie et al.	364/200

FIG_1



“APPLE_PAT_4_875_158_03” 71 KB 2000-02-22 dpi: 300h x 300v pix: 1858h x 2672v

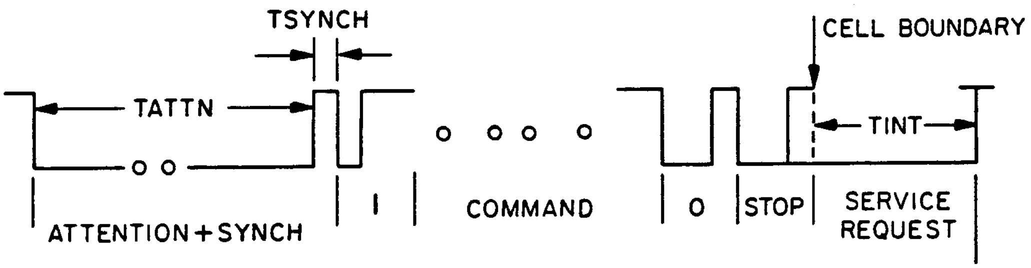


FIG_6

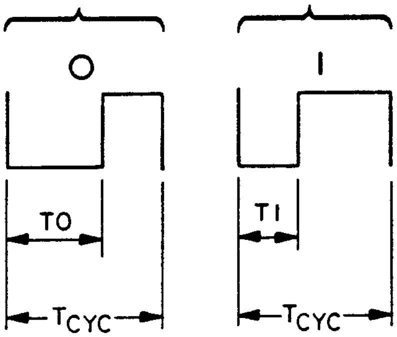


FIG 2

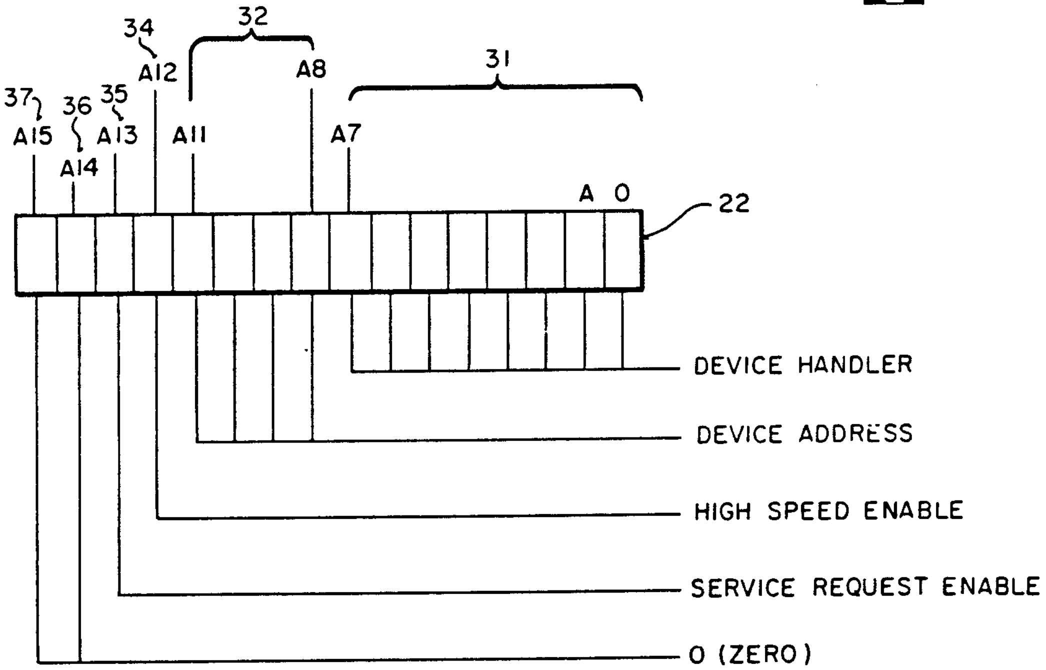


FIG_3

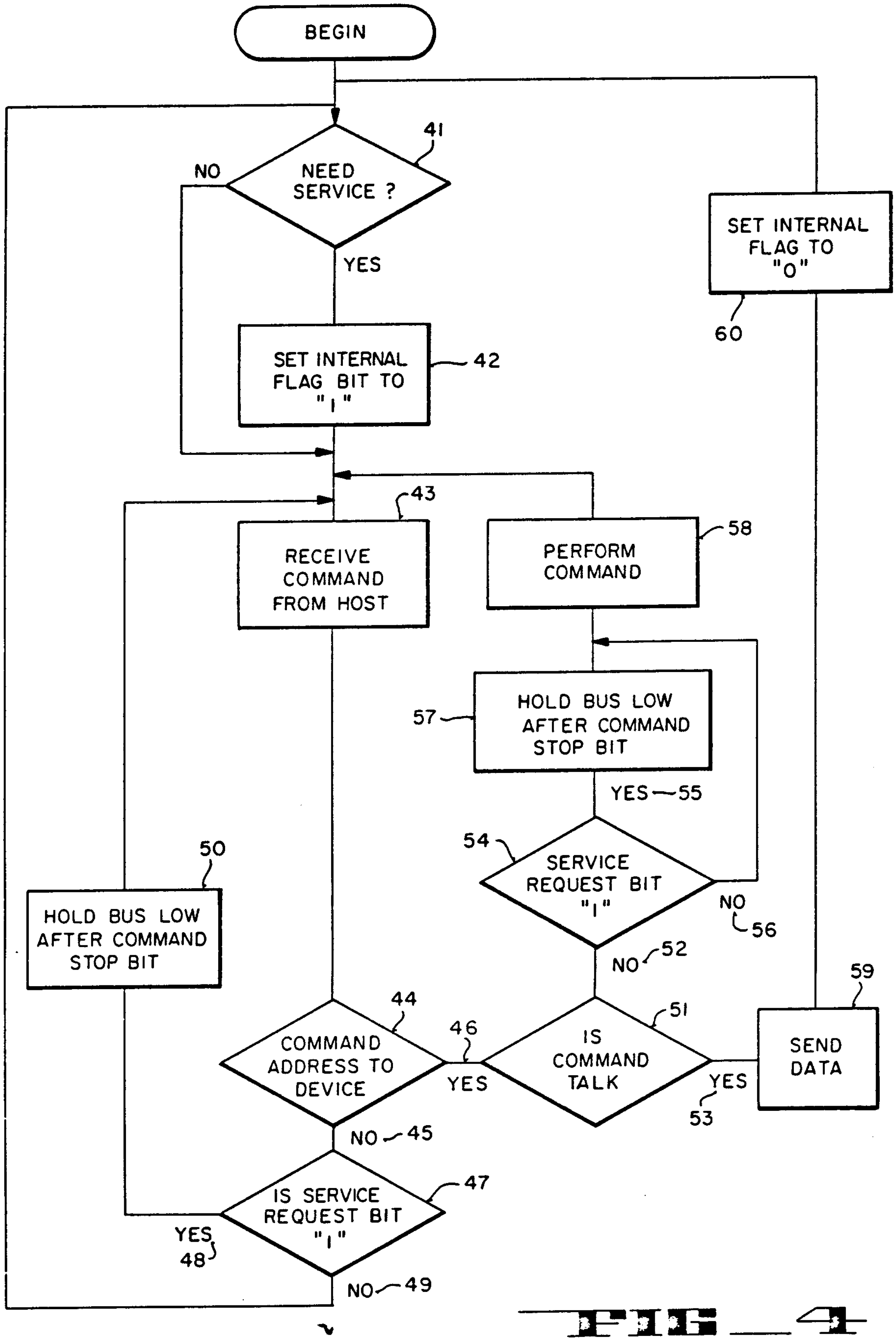

BEGIN
NEED SERVICE ?
41
NO
YES
SET INTERNAL FLAG BIT TO "1"
42
SET INTERNAL FLAG TO "0"
60
RECEIVE COMMAND FROM HOST
43
PERFORM COMMAND
58
HOLD BUS LOW AFTER COMMAND STOP BIT
57
YES—55
SERVICE REQUEST BIT "1"
54
NO
56
NO
52
HOLD BUS LOW AFTER COMMAND STOP BIT
50
IS COMMAND TALK
51
YES
53
SEND DATA
59
COMMAND ADDRESS TO DEVICE
44
46
YES
NO
45
IS SERVICE REQUEST BIT "1"
47
YES
48
NO
49

FIG. 4

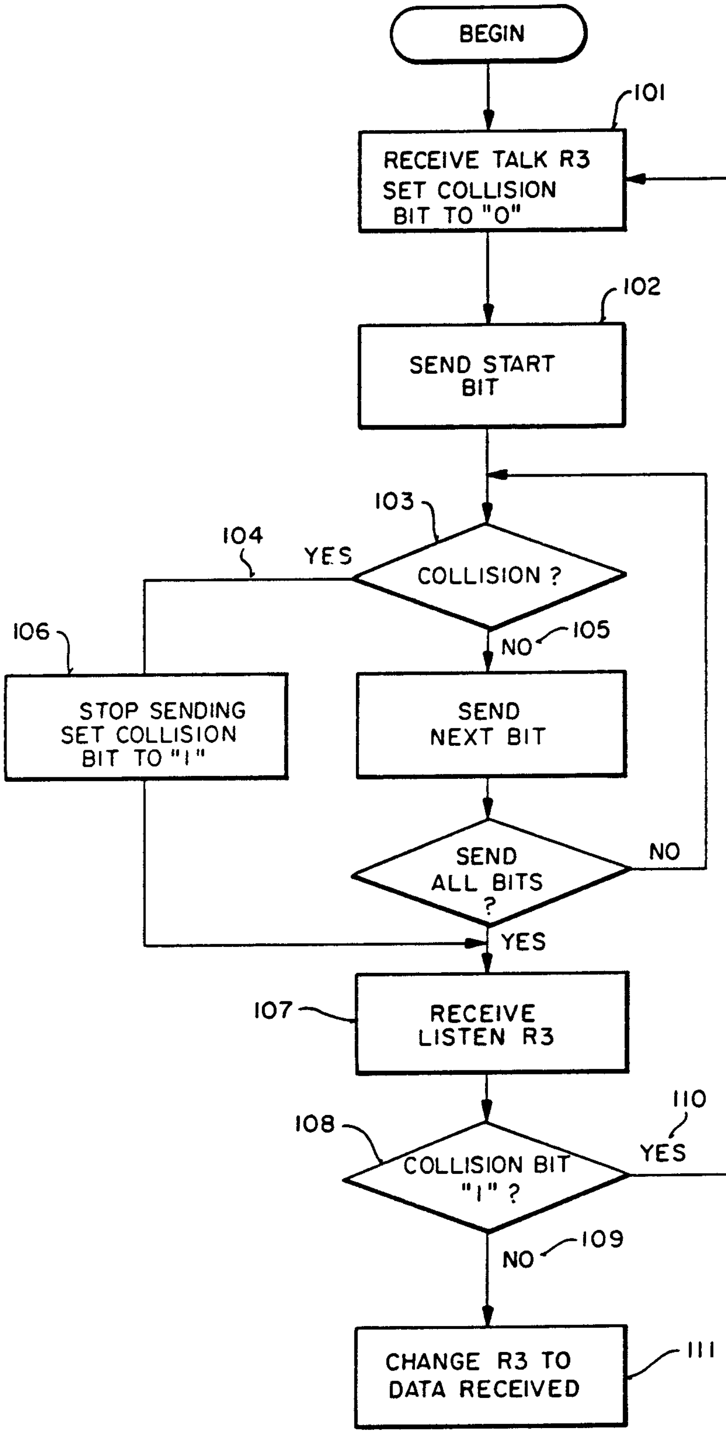

BEGIN
101
RECEIVE TALK R3 SET COLLISION BIT TO "0"
102
SEND START BIT
103
COLLISION ?
104
YES
106
STOP SENDING SET COLLISION BIT TO "1"
NO
105
SEND NEXT BIT
SEND ALL BITS ?
NO
YES
107
RECEIVE LISTEN R3
108
COLLISION BIT "1" ?
110
YES
NO
109
CHANGE R3 TO DATA RECEIVED
111

FIG _ 5

METHOD FOR REQUESTING SERVICE BY A DEVICE WHICH GENERATES A SERVICE REQUEST SIGNAL SUCCESSIVELY UNTIL IT IS SERVICED

This is a division of application Ser. No. 765,396 filed Aug. 14, 1985.

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention relates the field of communications media for transferring data between a source and a plurality of peripheral devices coupled to the source. More particularly, the present invention relates to data transfer along a peripheral device bus between a plurality of peripheral devices and a host computer.

2. Art Background

In the computing industry, it is quite common to transfer data and commands between a plurality of data processing devices, such as for example, computers, printers, memories and the like. The interconnection of computers and other peripheral devices principally developed in the early 1970's with the advent of computer networking systems, which permitted the distribution of access to computing resources beyond the immediate proximity of a main frame computer.

Networks, such as the ARPA network, were developed to provide access by various users to large time-sharing systems and the transfer of data between such systems. In the case of geographically local networks, so-caled "local area networks" (LANs) were developed to connect together a collection of computers, terminals and peripherals located, typically in the same building or adjacent buildings, and permitted each of these devices to communicate among themselves or with devices attached to other networks. Local area networks permit the implementation of distributed computing. In other words, some of the devices coupled to the local area network may be dedicated to perform specific functions, such as file storage, data base management, terminal handling, and so on. By having different machines perform different tasks, distributed computing can make the implementation of the system simplier and more efficient.

Presently, networking has only been applied to provide communications between data processing devices, which are machine input devices. However, it would also be useful to provide a networking means to provide communication between a single computer and a plurality of peripheral devices such as human input devices, listen only devices, appliances, etc. Human input devices include keyboards, cursor control devices (such as a "mouse"), and sketch pads, etc. Listen only devices include transaction logs, etc. In the prior art, such devices are attached to a host computer through a port dedicated to each device. Often, additional "cards" are required to allow a peripheral input device to be added. Further, the addition of cards requires that the host computer be powered down, with no mechanism for adding peripheral devices to a live system. Such prior art systems are inefficient since peripheral devices are not generally operated simultaneously. (for example, someone using a mouse is generally not using the keyboard or sketchpad at the same time). Thus, the devices should share a common line to the host computer without creating data traffic problems, eliminating the need for cards.

Prior art networking schemes also include elaborate methods for establishing control of the network to allow a device to transmit. Such systems are not needed for networking of peripheral devices, since only one is generally used at a time. In addition, prior art networking schemes provide for means for attached devices to identify themselves to each other through elaborate "handshaking" schemes. Again, such complexity is not required to connect peripheral devices since there is no need for these devices to identify themselves to other devices, only to the host computer.

Therefore, it is an object of the present invention to provide a communications medium for a plurality of peripheral devices, which provides a simple and efficient means for coupling those devices to a host computer.

It is a further object of the present invention to provide a communications medium by which all such peripheral devices can be coupled to a host computer at a single input.

It is still another object of the present invention of provide a communications medium which provides a means for peripheral devices to indicate a need for servicing to the host computer.

It is yet another object of the present invention to provide a communications medium which provides a means for determining if the communications medium is in use.

It is another object of the present invention to provide a communications medium which allows peripheral devices to be added during operation of the system.

SUMMARY OF THE INVENTION

A communications medium is disclosed including apparatus and methods for transferring data between a plurality of peripheral devices and a host computer. In the preferred embodiment, a plurality of peripheral devices such as human input devices (including mice, keyboards, sketchpads, etc.), appliances, listen only devices, etc., are coupled to a common cable for data transmission and reception of commands. A peripheral device coupled to the cable may signal the host computer when it requires servicing. This peripheral device will continue to request service until the host computer commands it to transmit its data. All peripheral devices of the same generic type (e.g., all keyboards), may have an identical hard wired address used as an identification number. In this manner, the host computer can identify the generic type of device communicating on the cable. If more than one of the same type of device is coupled to the cable (e.g., 2 mice), the host computer will assign new addresses in the status registers of the mice so they can be differentiated.

In the preferred embodiment, a return to zero modulation scheme is used to transmit data and commands over the cable. As a result, a peripheral device will assume a collision if it attempts to transmit a high signal on the cable and the cable is pulled low by another device. In order to simplify the protocol of the system, only the computer can initiate communication.

The present invention permits the addition of peripheral devices to a computer while the computer is in use, without the need to power down the computer system. The present invention can be embodied in a narrow band medium, as well as broad band, fiber optic, infrared and other media.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is block diagram illustrating the networking system of the present invention.

FIG. 2 is a timing diagram illustrating the present invention's use of return to zero encoding.

FIG. 3 illustrates a register of a peripheral device of the present invention.

FIG. 4 is a flow chart illustrating the sequence of operations utilized by a peripheral device to request service by the host computer.

FIG. 5 is a flow chart illustrating the sequence the operations utilized to provide new addresses to devices sharing the same hard-wired address.

FIG. 6 is a timing diagram illustrating a command transaction of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

A peripheral device bus including apparatus and methods for transferring data between a plurality of peripheral devices coupled to a host computer is disclosed. In the following description numerous specific details are set forth, such as specific numbers, registers, addresses, times, signals, and formats, etc., in order to provide a thorough understanding of the present invention. However, it will be apparent to one skilled in the art that the present invention may be practiced without these specific details. In other instances, well-known circuits and devices are shown in block diagram form in order not to unnecessarily obscure the present invention.

Referring to FIG. 1, the preferred embodiment of the present invention may be seen. A plurality of peripheral devices, generally identified by numbers 11 through 16 are coupled through a single cable 17 to a host computer 10. In the preferred embodiment, all devices communicate with the host computer by a mini-phono jack with the following connecter assignments; tip-power, ring-data, sleeve-power return. A "high" signal (1) is 2.4 volts minimum. A "low" signal (0) is 0.8 volts maximum. Although a single cable is contemplated in the preferred embodiment of the present invention, other communications media, such as broad band methods, fiber optic systems, and infrared signals, are contemplated.

The bus of the present invention supports coded devices (for which a keystroke represents a symbol or a function, such as a keyboard 14), relative devices (in which movement of a display cursor in response to a control device, such as a mouse 11 or 12, may be from any starting point), and absolute devices (for which there is a constant and direct relationship between display position and device position, such as sketch pad 13).

The system also permits the networking of extended address devices. Extended address devices share a common hard wired address 35, but further include an address unique to the individual device which the host computer must recognize before the device can be accessed. For example, it is contemplated that appliances may be coupled to the host computer and controlled by the host computer. In such a situation, all appliances would have an identical hardwired fixed address. The host computer, on a first level, would simply address the hard wired address for appliances. At this time, all appliances coupled to that address are inactive. An individual appliance may be activated by the host computer if the host computer sends a signal to that appliance which matches the extended address of the appliance. An extended address is an individual identification number, which, in the preferred embodiment, may be up to 64 bytes long. Once the host computer has provided the extended address, the device having that address is active. Subsequent commands to the appliance address location will be executed by that device without the need for providing the extended address each time. An activated appliance will respond to all commands to the appliance address, while unactivated devices remain passive. To deactivate an active extended address device, the host computer provides the extended address of another extended address device, activating it and deactivating the previously active device. It is contemplated that any device which would be controlled by the host computer is suitable for the present networking scheme, such as lights, ovens, sprinkler systems, phone answering machines, etc. It is contemplated that at least one other hardwired address for extended address devices be provided in the present system. Such an address would be used for system protection schemes or user identification schemes. For example, a device at this location could contain an extended address which must be provided by the system user before the system could be enabled. In other instances, individual operations could require that the extended address of other security devices be provided by the host computer prior to performance. Such security devices could function as "keys" to lock the entire system or certain operations performed on the system.

Also reserved for use on the network of the present invention are soft address locations 16. Soft address locations are reserved for duplicates of peripheral devices coupled to the bus. When more than one mouse is coupled to the bus, for example, the host computer assigns new addresses to each mouse, those addresses being at the soft address locations.

Although specific examples have been given for each type of device coupled to the bus, there may be more than one kind of each type of device with that address. For example, a sketch pad has been given as an absolute device but a touch screen would also be considered an absolute device and be assigned the same fixed command address as the sketch pad. In those situations, the host computer will assign new addresses from the soft address locations to each device.

In the preferred embodiment of the present invention, the various peripheral devices have been assigned addresses as shown below:

Address	Device Types	Example
0000 (zero)	extended address device	security systems, user ID
0001 (one)	extended address device	appliances
0010 (two)	coded devices	keyboard
0011 (three)	relative devices	mouse, track ball
0100 (four)	absolute devices	sketchpad, touch screen
0101 (five)	reserved	none
0110 (six)	reserved	none
0111 (seven)	reserved	none
1000 (eight)	soft addressed	duplicate peripheral devices
—	—	—
—	—	—
1111 (15)	soft addressed	duplicate peripheral devices

It will appreciated by one skilled in the art that other addresses may be assigned to these devices containing more or less bits than in the preferred embodiment. Fixed hard-wired addresses **31**, **32**, **33**, and **34** are shown in FIG. **1** for mouse **11**, mouse **12**, sketch pad **13**, and keyboard **14**, respectively.

All peripheral devices have four registers in the preferred embodiment to receive data and send data. For each device, register **3** talk and register **3** listen have status information such as device address and handler information. The remaining registers are data registers which are device specific except register **2** listen which contains the extended addresses for extended address devices or device specific contents for soft addressed devices.

In the preferred embodiment of the present invention, there are three types of communication on the peripheral bus; commands, data and global signals. Commands are sent from the host computer to the peripheral devices, data is sent from the host computer to the devices or from the devices to the host computer, and global signals are special messages sent to the entire system.

In the preferred embodiment data is encoded as the ratio of low time to high time of each bit cell. A bit cell boundary is defined by a falling edge on the bus. A "zero" is encoded as a bit cell in which the low time is greater than the high time. This is shown in FIG. **2** by bit cell **20**. Therefore, a "1" is defined as a bit cell in which the low time is less than the high time as shown by cell **21** of FIG. **2**. In the present preferred embodiment, a start bit is defined as a "1". A stop bit is a "0" which does not have an additional falling edge to define the bit cell time. The stop bit is used to synchronize the stopping of transactions on the bus.

The period for each bit cell of command signals and low speed data transmission is approximately 100 microseconds plus or minus 30%. For high speed data transmission, the bit cell is 50 microseconds plus or minus 1%. The format of a data transaction is a start bit (1), followed by up to 256 bits of data and ending with a stop bit. It will be appreciated that when other communications media are utilized, other signaling methods may be utilized.

Commands are sent only by the host. In the preferred embodiment of the present invention, there are three commands; talk, listen, and flush. As shown in FIG. **6**, to signal the start of a command, an attention pulse is sent out. An attention pulse is generated by the host computer by transmitting a bus low for a period of "T-attn". In the preferred embodiment, T-attn is approximately 560–1040 microseconds. The attention pulse is followed by a synch pulse to give the initial bus timing. The following edge of the synch pulse is used as a timing reference for the first bit of the command. The command is followed by a stop bit, (in the preferred embodiment a "0"). After the stop bit, the bus returns to its normally high state unless a device requests service.

The command is an 8 bit value in the preferred embodiment. The command includes a 4 bit device address field which specifies the fixed hardwired address of the desired peripheral device (e.g., 0011 for a mouse). The next 2 bits form the command and the final 2 bits form a register address field which allows a specific register, R0–R3 within an addressed peripheral device to be specified. In the preferred embodiment, the commands have the following bit code:

Command	Code
Flush	01
Listen	10
Talk	11

The talk command orders the addressed device to provide its data to the host computer. The listen command orders the addressed device to accept data from the host computer and place it in one of its registers. The flush command has an effect on each device which is defined by the individual device. It can be used for such functions as clearing a register or resetting all keys on a keyboard so that they will be sent again.

When a peripheral devices is addressed to talk, it must respond within a certain period, called the "time out" period. The time out, "T1t", is approximately 140 to 260 microseconds (2 bit cells). The selected device, if it does not time out, becomes active on the bus and performs its data transaction, and then "untalks" itself and goes inactive on the bus.

Global signals are used for transactions which are neither commands nor data transactions. Global signals include: attention and synch, which is used to signal the start of a command and to give initial bus timing; service request, a transaction that devices use to signal the host that they require service; and reset, used to issue a break on the bus by holding the bus low for a minimum of "Tres", which is approximately 2.8 to 5.2 milliseconds, (40 bit cells). Global signals will be described in more detail in conjunction with other transactions.

Since a peripheral device can only send data when it has been commanded to talk by the host computer, the present system provides a means for a device to notify the host computer that it needs servicing. This is accomplished by having the device send a service request signal to the host computer. In the present invention, a service request is sent by holding the bus low after the stop bit of any command transaction. Each of the peripheral devices coupled to the bus include a number of registers (in the preferred embodiment four registers). FIG. **3** shows one of the registers for a peripheral device. Bit A13 has been identified as the service request enable bit. When this bit is set high by the host computer, the device is enabled to hold the bus low after the stop bit of a command transaction, as shown in FIG. **6**, if the device needs service. A device will keep requesting service until it receives a talk command from the host. The flow chart in FIG. **4** shows the steps followed by a device requiring service.

Initially the device determines if it requires servicing, Block **41**, that is, if it has data to send to the host. If it does, it sets an internal flag bit, Block **42**. When the next command is sent out from the host, Block **43**, the device checks to see if the command is addressed to the device, Block **44**. If the command was not adddressed to the device Branch **45**, the device checks to see if its service request enable bit, (bit A13 of register **3**), is set high, Block **47**. If so, Branch **48**, it holds the bus low after the command stop bit, Block **50**. (See FIG. **6**) The device then waits until the next command is received from the host to see if it will be addressed to talk, Block **43**. If the command is addressed to the device, Branch **46**, the device determines if it is a command to talk, Block **51**. If it is not a command to talk, Branch **52** the device sends a service request, Block **57**, performs whatever command is instructed, Block **58**, and awaits the next command, Block 43. If the command is to talk, Branch 53, the device sends its data, Block 59, and considers its service request to be satisfied, Block 60. The device continues to monitor itself to determine when it needs service, Block 41. By allowing the host computer to control the service request enable bit, more efficient operation of the bus is realized. When a service request is received, the host computer need only ask those devices whose service request bit was enabled whether they need servicing. Additionally, the host computer can disable certain devices that are not required for particular applications.

When sending data, the device is able to detect collisions. If a peripheral device tries to output a 1 and the data line is or goes to a 0, the device assumes it has lost a collision to another device. This means that another device is also sending on the bus. When this happens the losing device untalks itself from the bus and preserves the data which was being sent for retransmission. The device sets an internal flag bit if it loses a collision. Prior art peripheral devices were unable to detect collisons. This novel feature of the present invention permits more efficient operation of the communications medium. By having the device sense a collision, it can preserve the data that is transmitted and indicate to the host computer that it requires serving. Additionally, the collision detection scheme of the present invention does not require a waiting period before a collision is assumed. A device will end its transmission if the line is modulated by another device or simply not begin its transmission if the line is already in use. Further, this collision detection scheme is useful in locating multiple devices at a single hardwired address location, such as mouse 11 and mouse 12 of FIG. 1.

In such a situation, the host will change the address of the devices by forcing a collision of devices sharing the same address. The host achieves this by issuing a talk R3 command addressed to those devices. As shown in FIG. 3, Register 3 22 (one of the registers of the device) contains the following information. Bits A0 through A7 31 contain a device handler which tells the host computer the function of a device and the use of data provided by the device. Bits A8 through A11 32 are an address field which can be changed when more than one device, having the same command address, is coupled to the bus. In that situation, one of the soft address locations are assigned to bits A8 through A11 32 which then serve as the command address for that device. Until that time, those bit locations contain a random number which aids in the detection of collisions. For example, if two mice received a talk R3 command and both began talking at the same time, neither would detect a collision. However, by having random numbers in the address field 32 of register 3 22, the output of the two devices will eventually differ. When that occurs, one of the devices will detect a collision and stop talking. Bit A12 34 is a high speed enable bit which if set, provides for data transmission at the higher modulation rate (50 microseconds per bit frame). The high speed enable bit is set by the host computer. If the host computer is unable to receive data at the higher modulation rate, it sets the high speed enable bit low in each of the devices. If the host computer is able to accept data at the higher modulation rate, and the device is able to transmit at the higher rate, (that information being contained in the handler bits 31 of register 3), the host computer sets the high speed enable bit 31 high for the device. As previously mentioned, bit A13 35 is service request enable which is set by the host to enable the device to perform a service request transaction. Bits A14 36 and A15 37 are reserved for future use and are set to 0.

When a device receives a talk R3 command the device provides its status (handler and address) to the host computer. If there are two devices of the same type coupled to the bus, only one can respond since the other will detect a collision. FIG. 5 shows the method of assigning new addresses on the bus.

After receiving a talk R3 signal, Block 101, the device sends its status from Register 3. If the line goes low, the device determines that there has been a collision, Branch 104, it stops sending (untalks itself) and sets and internal flag bit to indicate a collision, Block 106. The host sends a listen R3 to the mouse address, Block 107. Each commend resets the internal collision flag of the device. The device checks to see if its collision bit is set, Block 108. If the collision bit is not set, Branch 109, the device changes A8 through A11 to the soft address provided by the listen R3 command, Block 111. In this manner the address of the winning device is changed with the host computer keeping track of the new address of the device. If a collision bit is detected by the device after a listen R3 command, Branch 110, the device does not change the soft address bits, but may change other fields in R3. The host computer sends out another talk R3 command, Branch 101 to see if any devices remain at the mouse address. In this situation the remaining mouse will send its start bit, Block 102, not detect a collision, Branch 105, and send its status from register 3, Block 112. The host computer will send back a listen R3 command to the mouse address, Block 107. The remaining mouse will not detect a collision bit being set in this instance, Branch 109 so it will change bits A8 through A11 of register 3 to the soft address received from the host computer, Block 111. The host computer then sends out another talk R3 command to the mouse address, Block 101. This time, since no mouse remains at that address, the bus is timed out and the host computer knows that it has assigned new addresses to each of the mice sharing the mouse address.

In one embodiment of the present invention, peripheral devices have a device on them to indicate activity called the activator. The activator can be a special key on a keyboard or a button on a mouse. When more than one of a device is coupled to the bus, the host computer can display a message requesting one of the devices to use the activator. The host can then issue a listen R3 command which will change the address of the device which is activated. In this manner individual devices can be located and assigned new addresses in multiuser applications.

Thus, a peripheral device bus has been described which allows a plurality of peripheral devices to be coupled to a host computer through a single port.

We claim:

1. A method for requesting service by a device coupled to a host computer through a communication medium, comprising the steps of:

(1) the host computer setting a service request bit of the device to a first logical value to allow the device to produce a service request signal if the device requires servicing;

(2) the device determining that it requires servicing and setting an internal flag bit to a first logical value to indicate that the device requires servicing;

(3) the device monitoring a command from the host computer to see if the command is addressed to the device;

(4) if the command is not addressed to the device and if the service request bit is set to the first logical value, then the device generating a service request signal on the medium after the command by holding the communication medium low for a first period of time;

(5) if the command is addressed to the device, if the device determines that the command is not a command that services the device, and if the service request bit is set to the first logical value, then the device:

(a) generating the service request signal on the medium after the command by holding the medium low for the first period of time and

(b) performing the command;

(6) if the command is addressed to the device, if the device determines that the command is not a command that services the device, and if the service request bit is not set to the first logical value, then the device performing the command without generating the service request signal;

(7) repeating steps 3, 4, 5, and 6 until the device receives a command addressed to the device that services the device.

2. The method of claim 1 for requesting service, wherein the first logical value is a logical one.

3. The method of claim 1 for requesting service, wherein the command that services the device is a talk command addressed to the device.

4. The method of claim 1 for requesting service, comprising the additional step of the device sending its data to the host computer after receiving the talk command addressed to the device.

* * * * *

United States Patent [19]

Farand

[11] **Patent Number: 4,884,069**

[45] **Date of Patent: Nov. 28, 1989**

[54] **VIDEO APPARATUS EMPLOYING VRAMS**

[75] Inventor: **Tobin E. Farand, Mountain View, Calif.**

[73] Assignee: **Apple Computer, Inc., Cupertino, Calif.**

[21] Appl. No.: **27,847**

[22] Filed: **Mar. 19, 1987**

[51] Int. Cl.^{4} **G06F 13/00**

[52] U.S. Cl. **340/799; 340/798**

[58] **Field of Search** 340/726, 747, 799, 750,
340/749, 724, 792, 725, 798, 800; 364/518, 521

[56] **References Cited**

U.S. PATENT DOCUMENTS

3,836,902	9/1974	Okuda et al.	340/799
3,974,493	8/1976	Cavaignac et al.	340/798
4,259,668	3/1981	Nishimura et al.	340/799
4,283,765	8/1981	Rieger	364/521
4,404,554	9/1983	Tweedy et al.	340/726
4,491,834	1/1985	Oguchi	340/726
4,685,070	8/1987	Flinchbaugh	340/747

FOREIGN PATENT DOCUMENTS

0082746	12/1982	European Pat. Off. .
0150453	12/1984	European Pat. Off. .
0182454	7/1985	European Pat. Off. .
0189576	12/1985	European Pat. Off. .
0197413	3/1986	European Pat. Off. .
2155670	3/1984	United Kingdom .
2146811	9/1984	United Kingdom .

OTHER PUBLICATIONS

IBM Enhanced Graphics Adapter, Aug. 2, 1984, pp. 1-75.

IBM Technical Disclosure Bulletin, vol. 19, No. 2, Jul. 1976, pp. 548-550, New York, U.S.; W. J. Auen et al.: "Dynamic Image Alignment".

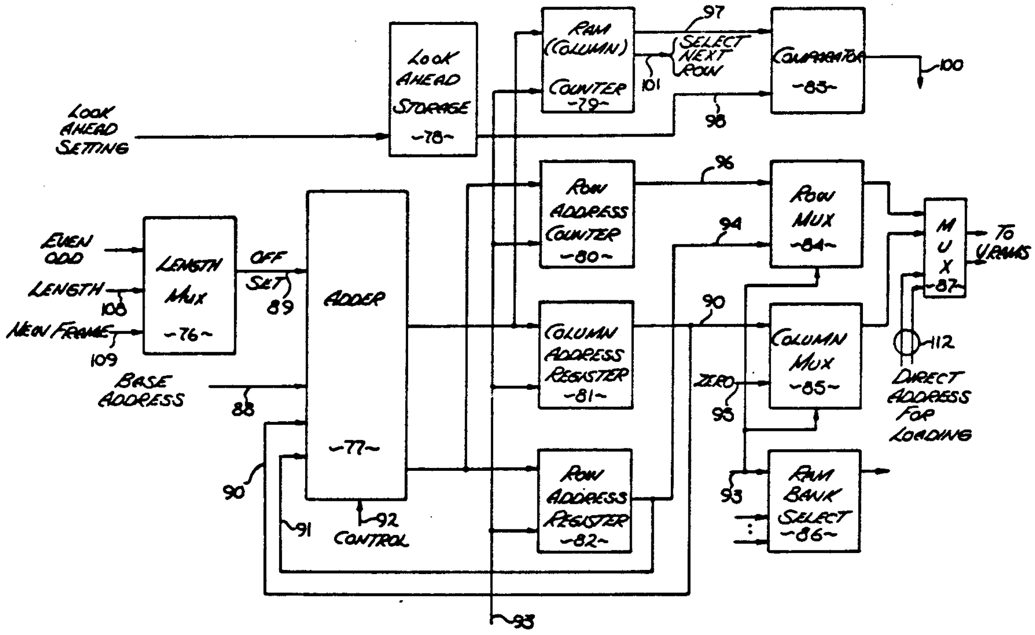
Primary Examiner—Alvin Oberley

Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor and Zafman

[57] **ABSTRACT**

A video card using VRAMs for a computer which includes a CPU and main memory. The VRAM addresses are generated in a manner making it unnecessary to have an integer number of scan lines per memory row. A counter keeps track of the shift register position in the VRAMs and a new row address is generated in hardware independent of the scan line. A look-ahead feature detects the approaching end of the shift register data and initiates a timing sequence to reload the shift register.

12 Claims, 5 Drawing Sheets



Macintosh 2 video·board

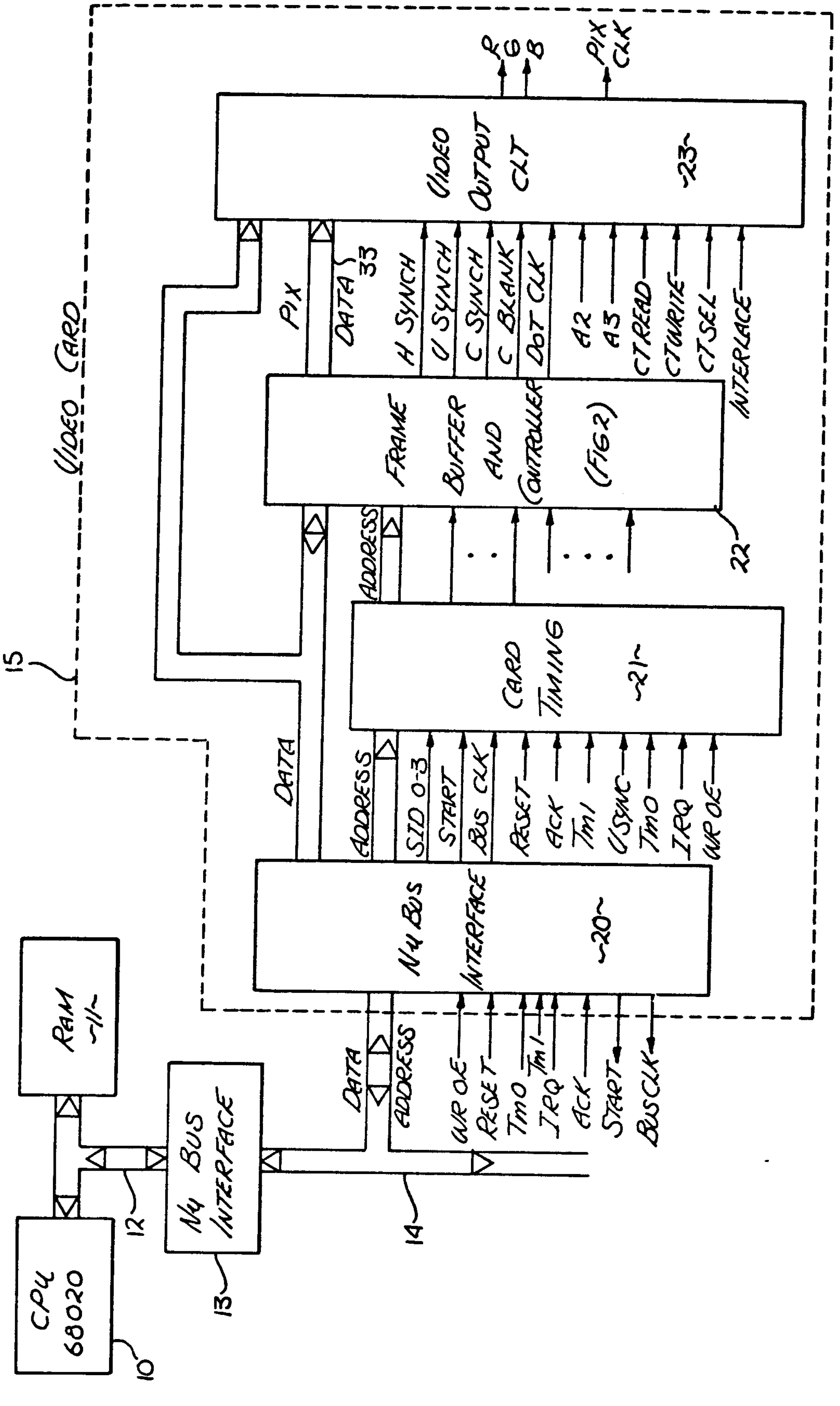


Fig. 1

“APPLE_PAT_4_884_069_02” 158 KB 2000-02-22 dpi: 300h x 300v pix: 1861h x 2897v

U.S. Patent **Nov. 28, 1989** **Sheet 2 of 5** **4,884,069**

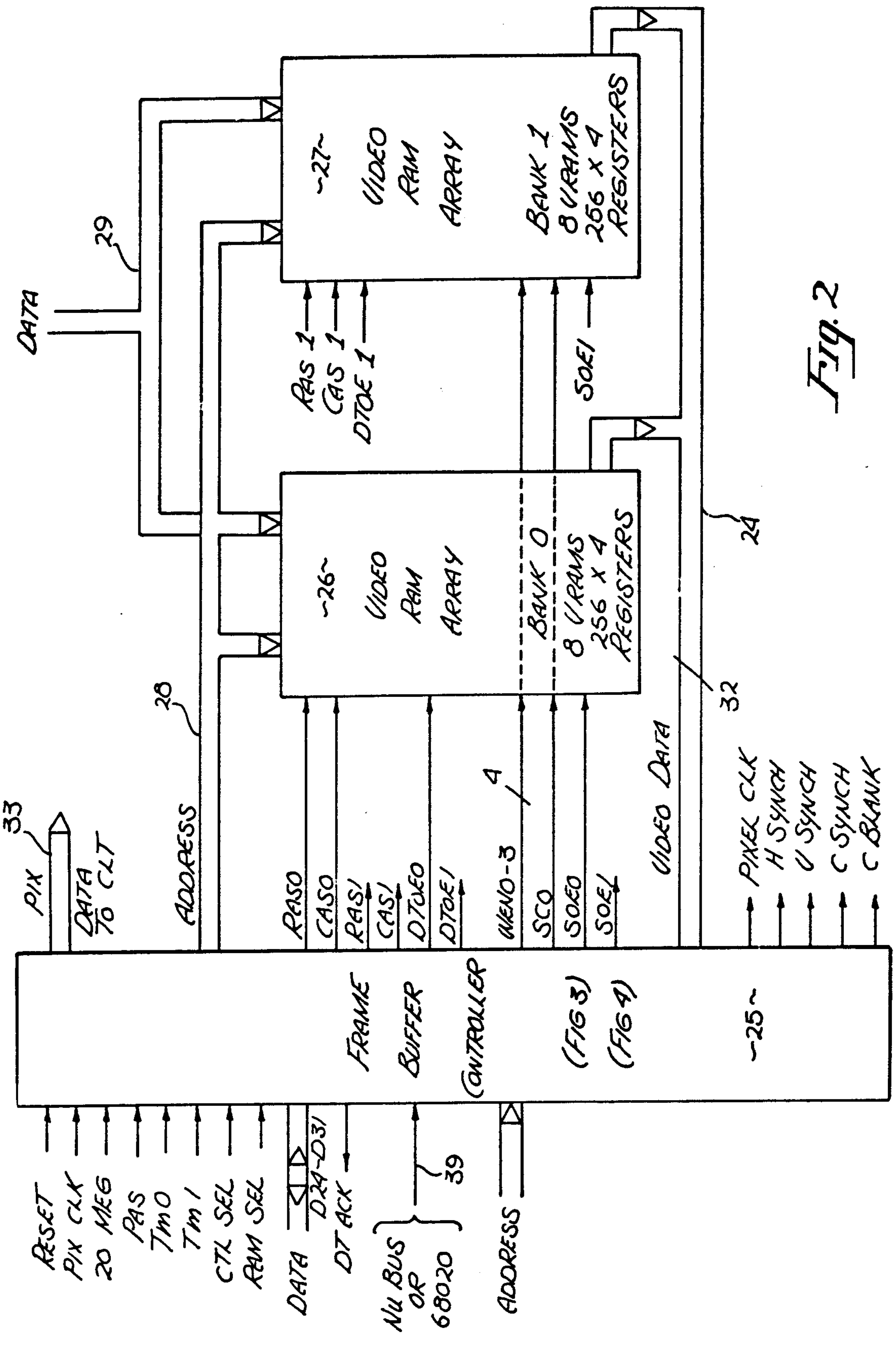


Fig. 2

"APPLE_PAT_4_884_069_03" 168 KB 2000-02-22 dpi: 300h x 300v pix: 1898h x 2847v

U.S. Patent Nov. 28, 1989 Sheet 3 of 5 4,884,069

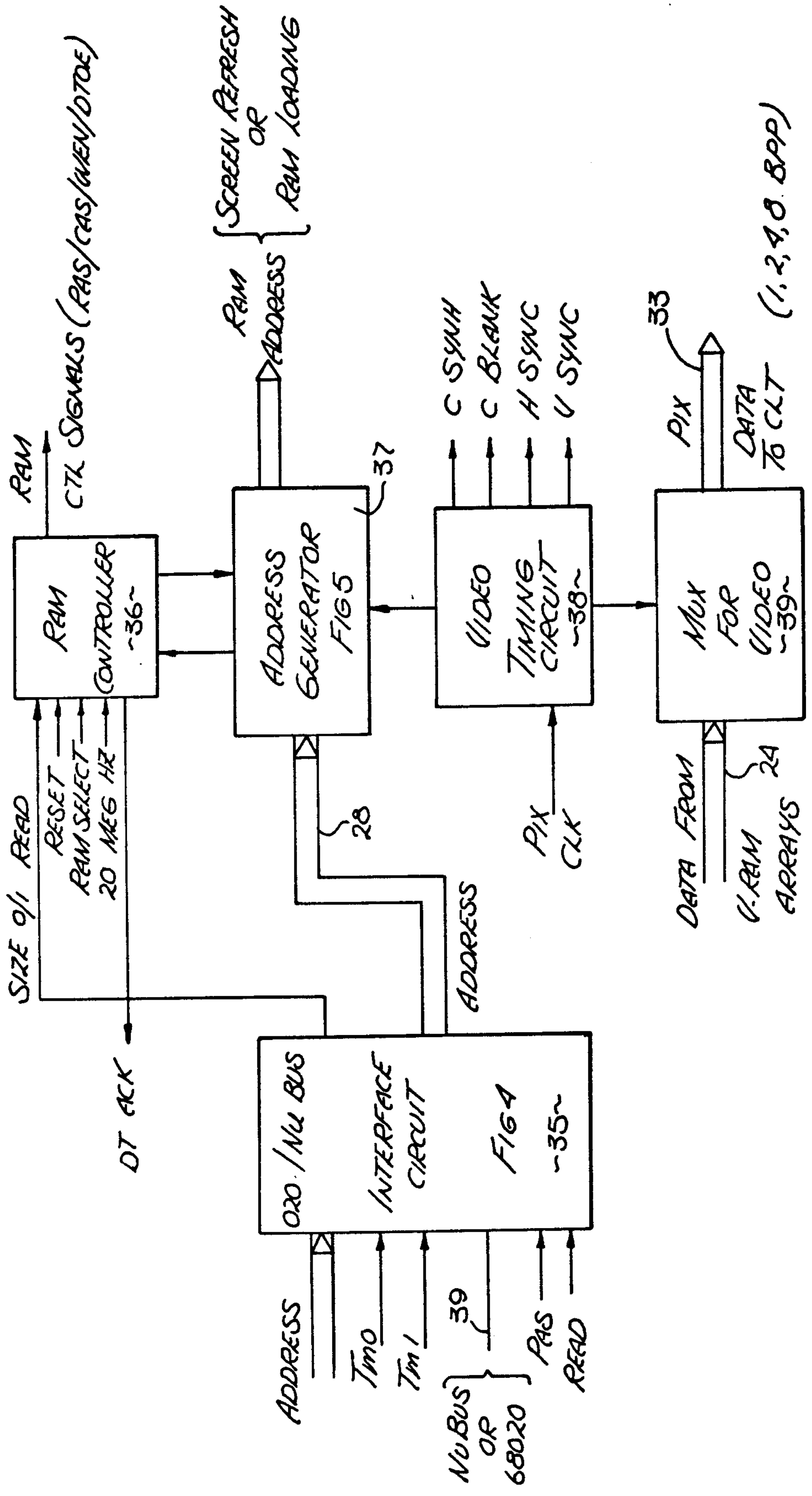


Fig.3

“APPLE_PAT_4_884_069_04” 132 KB 2000-02-22 dpi: 300h x 300v pix: 1880h x 2897v

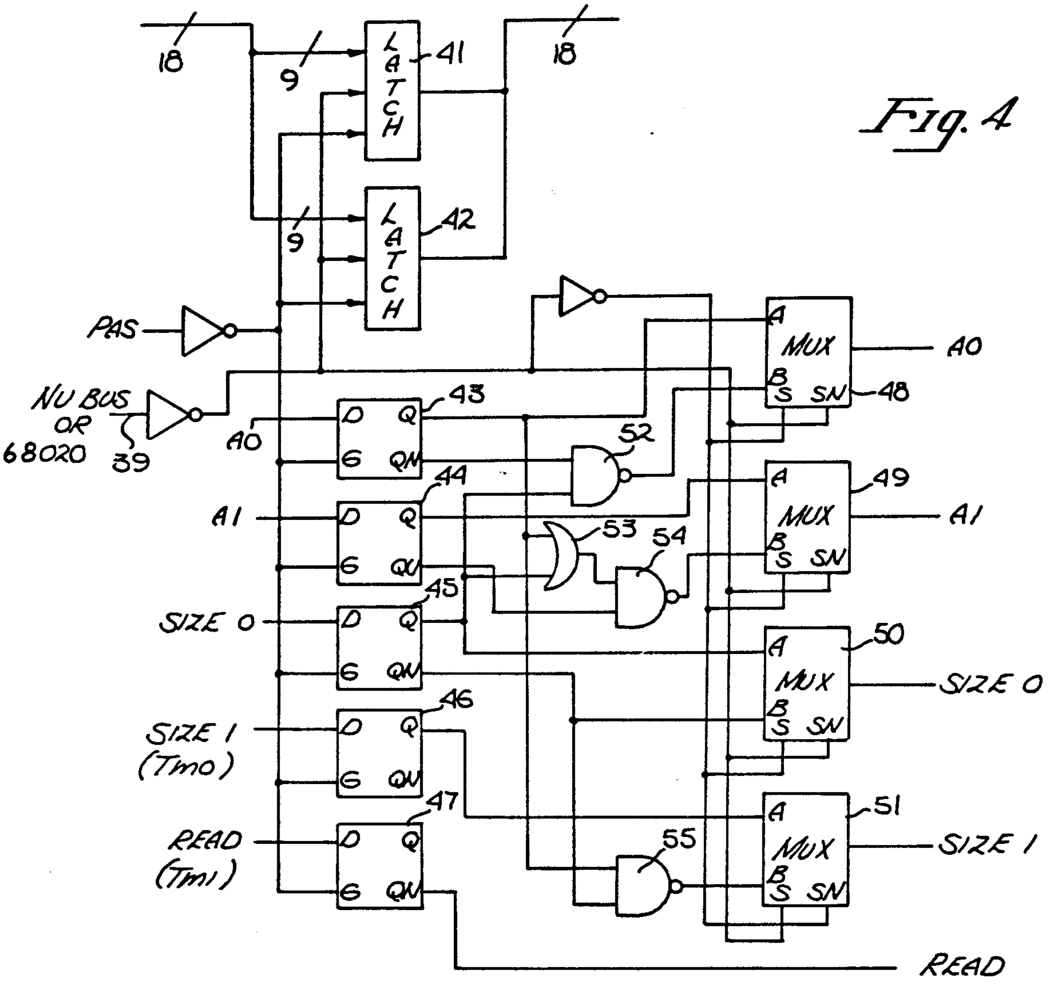

18
9
LATCH
41
18
9
LATCH
42
PAS
NU BUS OR 68020
39
A0
D Q
G QN
43
52
44
A1
D Q
G QN
53
54
45
SIZE 0
D Q
G QN
46
SIZE 1 (Tmo)
D Q
G QN
47
READ (Tmi)
D Q
G QN
55
A MUX B S SN
48
A0
A MUX B S SN
49
A1
A MUX B S SN
50
SIZE 0
A MUX B S SN
51
SIZE 1
READ

Fig. 4

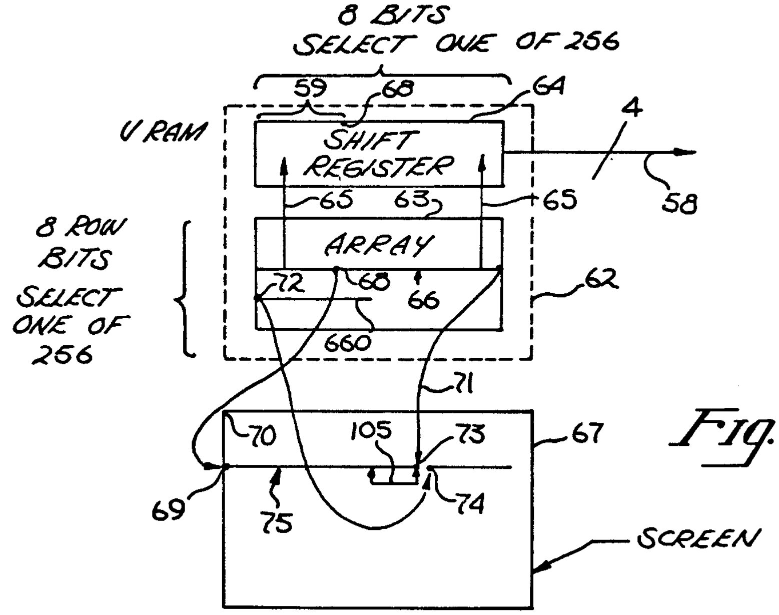

8 BITS
SELECT ONE OF 256
59
68
64
4
V RAM
SHIFT REGISTER
58
65
63
65
8 ROW BITS
SELECT ONE OF 256
ARRAY
72
68
66
62
660
71
70
105
73
67
69
75
74
SCREEN

Fig. 6

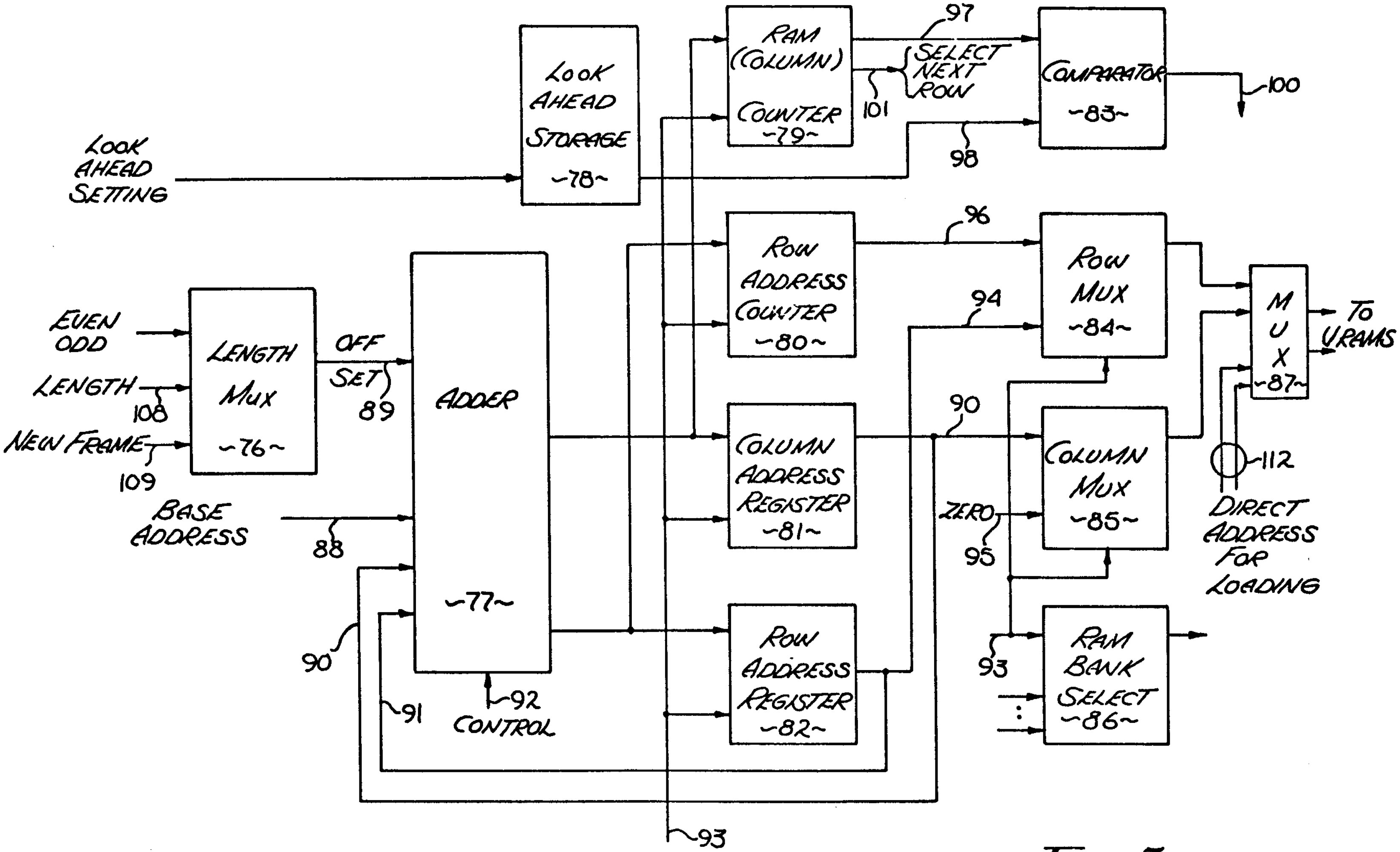

LOOK AHEAD SETTING
LOOK AHEAD STORAGE ~78~
RAM (COLUMN) COUNTER ~79~
97
SELECT NEXT ROW
101
98
COMPARATOR ~83~
100
ROW ADDRESS COUNTER ~80~
96
ROW MUX ~84~
94
M U X ~87~
TO VRAMS
EVEN ODD
LENGTH
108
NEW FRAME
109
LENGTH MUX ~76~
OFF SET
89
ADDER ~77~
BASE ADDRESS
88
90
91
92
CONTROL
COLUMN ADDRESS REGISTER ~81~
90
COLUMN MUX ~85~
ZERO
95
112
DIRECT ADDRESS FOR LOADING
ROW ADDRESS REGISTER ~82~
93
RAM BANK SELECT ~86~
93

Fig. 5

VIDEO APPARATUS EMPLOYING VRAMS

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention relates to the field of frame buffers for video displays and more particular, to addressing mechanisms for frame buffers.

2. Prior Art

Video random-access memories (VRAMs) have become commercially available in recent years for use with video displays. These devices include a memory array for storing pixel data and a shift register both formed on the same substrate. A row address is used to transfer data to the shift register. A column address is then used to identify a starting location in the shift register from which data is read out. Shift register operations can occur asynchronously with array accesses. Typically, the data is shifted out of the shift register at a much faster rate than that associated with dynamic RAM accessing.

In many applications, there is an integer number of scan lines displayed per row line in the memory. That is, a shift integer is not emptied midway in a scan line. There are timing and other problems if this correlation is not maintained.

The present invention provides circuitry for addressing the VRAMs while allowing a non-integer or integer number of scan lines per row of video memory. Among the features provided by the present invention is a lookahead mechanism used to initiate a memory cycle before the shift register is emptied. This permits the shift register to become empty in the middle of a scan line and to be reloaded in time to continue the scan.

SUMMARY OF THE INVENTION

A video apparatus (sometimes hereinafter referred to as the video section or video card) for providing video data from an array of VRAMs for a display for a computer is described. An interface means is used for interfacing between the video section and the central processing unit (CPU) of the computer. The pixel data stored in the VRAMs is addressed by an address generator which is coupled between the interface means and the VRAMs. The address generator includes a row address storage means and column address storage means which store a row and column address, respectively. A column counter is coupled to receive the column address and is clocked in synchronous with the pixel clock rate (more specifically, at the rate data is shifted from the shift register of the VRAMs). A row address counter is coupled to receive the row address. The addressing means includes a control means which causes the row counter to increment when the column counter reaches a predetermined count (e.g., 256 where the shift register has 256 stages). As this occurs the column count is returned to zero, allowing the next full row in the VRAM array to be used for the display.

Additionally, in the preferred embodiment a signal is generated before the shift register is emptied. This signal is generated by keeping track of the amount of pixel data remaining in the shift register. This lookahead feature is used to initate a time sequence for data transfer from the memory locations of the VRAMs into the VRAM shift registers.

Other features of the present invention such as the video section's compatibility with two different buses

are described in more detail in the following description.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram of the video section (video card) embodying the present invention in its presently preferred embodiment and also illustrates the card's coupling to a computer through a NuBus interface circuit.

FIG. 2 is a block diagram of the frame buffer and controller of FIG. 1.

FIG. 3 is a partial block diagram of the controller of FIG. 2.

FIG. 4 is a circuit diagram of a portion of the bus interface circuit of FIG. 3.

FIG. 5 is a detailed block diagram of the address generation means used in the presently preferred embodiment of the invention.

FIG. 6 is a diagram used to explain the operation of the address generation means of FIG. 5.

DETAILED DESCRIPTION OF THE INVENTION

A video apparatus having an array of VRAMs for use in a computer which computer includes a central processing unit (CPU) and main memory is described. In the following description, numerous specific details are set forth such as specific number of bits, etc., in order to provide a thorough understanding of the present invention. It will be obvious, however, to one skilled in the art that the present invention may be practiced without these details. In other instances, well-known circuits and timing has not been described in detail in order not to unnecessarily obscure the invention.

OVERVIEW OF THE COMPUTER

The video apparatus of the present invention is realized as a video card which is inserted into the motherboard of a computer. The computer, as shown in FIG. 1, includes a CPU 10 which is a commercially available 68020 microprocessor. The CPU communicates with a main memory, RAM 11, over a bus 12. The bus 12 is a standard bus structure using the protocol associated with the 68020 microprocessor. For instance, the address and data signals are transferred over separate lines, that is, they are not multiplexed over common lines. The computer includes a plurality of slots into which cards are inserted. These slots are coupled to a NUBUS bus 14 ("NUBUS" is a trademark of Texas Instruments Incorporated). The NUBUS interface circuit 13 provides the interface between the 68020 bus 12 and NUBUS. (By way of example, the interface circuit 13 includes multiplexing/demultiplexing means since on the NUBUS the data and address signals are multiplexed.) The video card 15, as mentioned, engages one of the slots in the computer and communicates with the NUBUS 14. The outputs from the card 15 include the standard red, green, blue (RGB) signals which are coupled to a video monitor to provide a color display.

Numerous circuits associated with the computer of the Figures such as a ROM which stores systems programs are not illustrated. Other aspects of the computer are disclosed in copending applications entitled MEMORY MAPPING UNIT, Ser. No. 015,907, Filed 2/18/87, U.S. Pat. No. 4,774,652; A COMPUTER WITH EXPANSION SLOTS FOR CARDS, Ser. No. 025,499, Filed 3/13/87; CARD FOR COMPUTER WITH EXPANSION SLOTS, Ser. No. 025,500, Filed

3/13/87; METHOD AND APPARATUS FOR DETERMINING AVAILABLE MEMORY SIZE, Ser. No. 027,005, Filed 3/17/87, now abandoned, all assigned to the assignee of the present invention.

The computer of FIG. 1 with its slots provides an "open architecture" version of the Apple Macintosh computer. Moreover, the 68020 provides enhanced processing capabilities over earlier versions of this computer. The video card 15 provides a color video signal as opposed to the non-color video on the earlier versions of this computer.

VIDEO CARD

The major elements of the video card 15 shown in FIG. 1 are the NUBUS interface circuit 20, card timing circuit 21, frame buffer and controller 22 and the video output circuit 23. The present application focuses mainly on the frame buffer and controller 22 since the present invention for the most part resides there. The circuits 20, 21 and 23 are discussed only in general terms, mainly to show the environment in which the present invention is used.

The NUBUS interface circuit 20 provides interface between the computer NUBUS 14 and the video card 15. The data and address signals are buffered within the circuit 20. Well-known timing signals and control signals associated with the NUBUS are also coupled to the card through the circuit 20. These are shown as the write output enable (WROE), Reset, TM0 and TM1, interrupt request (IRQ), acknowledge, Start and Bus CLK. The output from circuit 20 includes separate data and address buses. The data bus is coupled both to the frame buffer and controller 22 and the video output circuit 23. The address bus is coupled through the card timing circuit 21 to the frame buffer and controller 22. The NUBUS interface circuit 20 is constructed using well-known components and its construction is not critical to the present invention.

The card timing circuit 21 performs card level timing. The video timing used with the present invention is generated within the frame buffer and controller 22 and is described later in the application. This card level timing is not unique to the present invention and well-known timing circuits may be used. The card timing circuit 21 receives the slot identification lines for use in a decoder to generate a select signal. Other signals received by this circuit 21 include: Start, Bus CLK, Reset, ACK, TM0, TM1, IRQ, vertical synchronization (VSYNC) and WROE. In its currently preferred embodiment, the card timing circuit is fabricated from three programmable array logic integrated circuits. Also included as part of circuit 21 is a configuration ROM which provides configuration information for the video card.

The frame buffer and controller 22 is described in detail beginning with FIG. 2. In general, it provides the video timing and RAM timing for the video RAMs, memory control, RAM address generation and digital pixel data generation. Specific inputs to the frame buffer and controller 22 are set forth in subsequent figures.

The video output circuit 23 includes a color lookup table (CLT). Such tables are well-known in the art and, for instance, receive a code (e.g., 8 bits of pixel data) and provide a digital signal representing a predetermined color, for example, 8 bits representing red, 8 bits representing green, and 8 bits representing blue. These digital signals are then converted to analog signals and used to drive a color monitor. These color lookup tables are in some case ROMs. The particular CLT used in circuit 23 is a RAM which is written into the data bus.

OVERVIEW OF THE FRAME BUFFER AND CONTROLLER

As shown in FIG. 2, the frame buffer and controller includes the controller 25 and two banks of RAM, RAM array 26 (bank 0) and RAM array 27 (bank 1). The arrays 26 and 27 store the pixel data for the display and this data is sent to the color lookup table at the pixel clock rate (up to 8 bits in parallel) over bus 33. As currently implemented, the display comprises 640×480 pixels with a pixel clock rate of 30.24 mHz. The pixel data is read from the arrays through bus 24 (32 bits from the selected array) and then clocked out on bus 33 at either 1, 2, 4 or 8 bits per pixel. The data is loaded into the arrays directly from the data bus 29. The addresses for the arrays are generated within the controller 25 and coupled to the arrays via the bus 28.

The frame buffer controller 25 is described in more detail in conjunction with FIGS. 3 and 4. The controller receives a reset signal,' the pixel clock (PIX CLK), a 20 mHz timing signal, a physical address strobe (PAS), the TM0 and TM1 signals, a control select signal and a RAM select signal. The data lines D24–D31 are coupled to the controller and are used to load control registers. A data acknowledge signal (DT ACK) is provided by the controller as part of the data transfer protocol. As will be described in detail in conjunction with FIG. 4, the controller permits interfacing with either the NUBUS or a 68020 bus. The signal on line 34 indicates which of the two buses are coupled to the controller. (As currently employed and shown, the NUBUS is used.) The controller 25 also receives a 19 bit address field (one for bank select).

In addition to the pixel data output and addresses, the controller provides the control signals for the arrays 26 and 27. Standard row address strobe (RAS) signals and column address strobe (CAS) signals are provided for both arrays. RAS0 indicates the row address strobe for bank 0 and RAS1 is used to indicate the row address strobe for bank 1. Similar "0" and "1" designations are used for other control signals. The DTOE0 and DTOE1 signals are standard video RAM signals (data transfer output enable) which cause the loading of the shift register in the video RAM). The WEN0–3 lines (4 lines) are coupled to both arrays for byte lane selection when data is read into the arrays from the bus 29. SC0 is the serial clock signal which is coupled to both arrays. SOE0 and SOE1 are the serial output enables, one for each of the banks.

Additionally, the controller provides standard timing signals, specifically, the pixel clock, horizontal synchronization (H SYNCH), vertical synchronization (V SYNCH), composite synchronization (C SYNCH), and composite blanking (C BLANK).

Each array in the currently preferred embodiment comprises 8 commercially available video RAMs, specifically NEC Part. No. 41264. Each of these "chips" includes an array organization of 256 rows (1K bits per row) and a shift register with 256 stages (4 bits per stage). Therefore, each 16-bit address (8 row address signals and 8 column address signals multiplexed on bus 28) selects one of the rows in each of the video RAMs and allows the transfer of 256×4 bits into the shift register of each RAM. The SOE0 and SOE1 signals permits the selection of either array 26 or 27, and each array is thus able to couple 32 bits of data onto bus 24 since there are eight 256×4 registers within each array.

CONTROLLER

In FIG. 3, the major elements of the controller 25 are illustrated as interface circuit 35, RAM controller 36, address generator 37, video timing circuit 38 and multiplexer for video 39. Certain of the signals coupled to the controller 25 of FIG. 2 are coupled to the interface circuit 35 of FIG. 3. The interface circuit 35 of FIG. 3 is different and not part of the interface circuit 20 of FIG. 1. The latter provides interface between the video card and the NUBUS. The interface circuit 35 on the other hand accepts signals either from the NUBUS or directly from a 68020 bus and provides control signals which are used by the controller and buffer. The circuit 35 will be described in detail in conjunction with FIG. 4.

The RAM controller 36 receives the size 0, size 1 and read signals from the circuit 35 in addition to other inputs to the controller, specifically reset, RAM select and the 20 mHz clock signal. The controller provides the ordinary control signals for the RAM, mainly the RAS, CAS, WEN, DTOE, etc., signals. It also provides a data acknowledge signal for the NUBUS or 68020 handshake. The size 0 and size 1 signals determine which byte lane(s) of the 32-bit data bus is(are) being used. Controller 36 also controls the refreshing of the VRAMs. The RAM controller 36 employs ordinary circuits, not critical to the present invention.

The address generator 37 is described in conjunction with FIGS. 5 and 6.

The video timing circuit 38 receives the pixel clock and generates composite synchronization and blanking signals, and the horizontal and vertical synchronization signals. The timing circuit also provides timing signals to the generator 37 and to the multiplexer 39. The timing circuit 38 is fabricated employing well-known circuits.

The multiplexer 39 receives the 32-bit of data from the RAM arrays on bus 24 and couples the video data onto the pixel data bus 33. The data is coupled either at 1, 2, 4 or 8 bits per pixel depending on the mode selected.

NUBUS/68020 INTERFACE CIRCUIT

Referring now to FIG. 4, the interface circuit includes latches 41 and 42. These latches receive 18 lines of the address bus. The latching is controlled by the physical address strobe (PAS). The NUBUS or 68020 select signal on line 39 controls polarity of outputs from the circuit of FIG. 4 (NUBUS and 68020 have opposite polarity standards). Thus, the signal on line 39 is coupled to latches 41 and 42 to control output polarity on line 18 and similarly, the signal on line 39 is coupled to the multiplexers 48–51 for the same purpose. (The polarity of the read signal is not changed.)

The latch 43 receives the A0 signal, latch 44 the A1 signal, latch 45 the size 0 signal, latch 46 the size 1 signal and latch 47 the read signal. The output of latch 43 is coupled to multiplexer 48 and as is apparent when the A terminal of multiplexer 48 is selected, the A0 signal appears at the output of latch 48. The QN output from latch 43 and the Q output from latch 45 are coupled to the NAND gate 52 and provide the B input to multiplexer 48. The Q output of latch 44 is coupled to the A input of multiplexer 49 and hence, when the A input of multiplexer 49 is selected, the A1 signal appears at the output of this multiplexer. The Q output of latch 45 and the Q output of latch 43 are coupled through the OR gate 53 and provide one input to the NAND gate 54. The QN output from latch 44 provides the other input to NAND gate 54. The output of NAND gate 54 is coupled to the B input of the multiplexer 49. Multiplexer 50 receives the Q output of latch 45 at its A input terminal and hence, the size 0 signal is coupled to the output of multiplexer 50 when input A is selected. The B terminal of this multiplexer receives the QN output of latch 45. The multiplexer 51 receives the Q output of latch 46 (size 1 signal) which again is coupled to the output of multiplexer 51 when the A terminal is selected. The B input terminal of multiplexer 51 is coupled to the output of the NAND gate 55. The inputs to this NAND gate are the QN output of latch 45 and the Q output of latch 43. The read signal is coupled directly through latch 47.

To understand the operation of the circuit of FIG. 4, it should first be understood that the major control signals from the 68020 bus are: read, size 0, size 1, A0, A1 and PAS. The data and address signals are not multiplexed. For the NUBUS, the major control signals are: TM0, TM1, A0, A1, Start, with the address and data being multiplexed and inverted. The size 0 and size 1 signals indicate the size of the data transferred, that is, 8, 16, 32, 24, or 32 bit wide transfer on the 32-bit bus. The A0 and A1 signals indicate where on the bus the transfer is to occur, that is, for example, an 8-bit transfer may occur on lines D7–D15. However, the NUBUS does not support a 3 byte transfer, therefore, size 0 (input to latch 45) is high at all times when the signals applied to the circuit of FIG. 1 are NUBUS signals.

A0, A1, size 0, size 1 and Read as shown in FIG. 4 are the designations for 68020 compatible signals which when used are directly coupled through the circuit and appear at the output of the multiplexers (except for Read). When the input to the circuit of FIG. 4 is coupled from a NUBUS, the equations which follow are implemented by the circuit of FIG. 4 (the TM1 signal is interpreted as a Read signal). The "x" in the following equations indicate an output from the multiplexers.

$$XA0=\overline{A0}\cdot Size\ 0$$

$$XA1=A0+Size\ 0\cdot\overline{A1}$$

$$X\ Size\ 0=Size\ 0$$

$$X\ Size\ 1=\overline{A0}\cdot\overline{Size\ 0}$$

The implementation of the above equations translates the NUBUS control signals into the same signals that would be sensed at the output of the interface circuit if that circuit were directly coupled to the 68020 bus.

VRAM ADDRESS GENERATOR

Before describing the address generator, it will be helpful to examine a VRAM and its addressing mechanism. In FIG. 6, a VRAM 62 is illustrated having a memory array 63 and a shift register 64. This VRAM is one of the plurality of VRAMs which form the RAM arrays 26 and 27 of FIG. 2. As mentioned, an 8-bit row address coupled to the VRAMs selects a row of data such as row 66 of array 63. This data is shifted into the shift register 64 as indicated by lines 65. The column address applied to the RAM 64 selects the starting location at which data from the shift register 64 is shifted from the shift register onto the output line 58 (4 bits at a time). For example, the column address may select a location corresponding to column 68 along the row 66; then the first data appearing on line 58 is data stored at location 68. As the shift register shifts, the data represented by the brackets 59 is shifted from the register 64.

Referring now to FIG. 5, the address generator includes a multiplexer 76. This multiplexer receives a signal which indicates whether a particular frame comprises odd or even lines of an interlaced display. A second signal coupled to the MUX 76 on lines 108 provides a digital number representing the length of the digital data required for each scan line pair (even and odd line) of the display As mentioned, the currently preferred embodiment can use 1, 2, 4, or 8 bits per pixel, therefore, this length is not fixed. (Different programs may, through software, select different lengths.) Obviously, if one bit per pixel is used, substantially less data and hence, substantially less memory space is used to store the pixel data for each scan line. The signal on line 109 indicates when a new frame begins and is used, as will be described, to control the selection at the offset at the output of the multiplexer 76 (lines 89). The length multiplexer 76 includes circuits which allows the output on lines 89 to be zero, the number on lines 108 or one-half the number on lines 108 (the purposes of which will be described).

The adder 77 is an ordinary digital adder which adds the offset on lines 89 to either the base address on lines 88 or to the address on lines 90 and 91. Control signals on lines 92 for each new frame cause the signals on lines 88 to be added to zero or ½ the number on lines 108, depending on whether an odd or even frame is being displayed. Thereafter, (for the remainder of the frame) the digital number on lines 89 are added to the digital numbers on lines 90 and 91. The output of the adder which is a VRAM memory address includes a row field and a column field (8 bits each) which are coupled to registers 81 and 82. The row address is also coupled to the row address counter 80 and similarly the column address is also coupled to the RAM (column) counter 79.

The row multiplexer 84 selects between the output of the row address counter 80 (lines 96) and the row address register 82 (lines 94). At the beginning of each frame, multiplexer 84 selects the output of register 82. When the shift register associated with the VRAMs reaches its end, the address on lines 96 is selected. The counter 80 increments (by 1) the address that is stored in registers 2 each time the shift register reaches its end.

The column multiplexer 85 selects between the contents of register 81 and a zero address on lines 95. At the beginning of each scan line, the address from register 81 is selected. This address which is also coupled into the counter 79 is incremented at the rate data is shifted in the shift registers of the VRAMs. (This is slower than the pixel clock rate since there are 32 bits from the VRAM for each count in counter 79.) When the counter 79 reaches a predetermined count (e.g., 256) an output signal occurs on line 101. This signal causes the multiplexer 84 to select lines 96 and the multiplexer 85 to select the zero address.

Lines 93 provide the timing signals and control signals to implement the counting and the address transfers described below.

The comparator 83 compares the count within the counter 79 with a digital number stored within the lookahead storage means 78. The contents of the counter are coupled to the comparator 83 via lines 97 and the contents of the storage means 78 are coupled to the comparator 83 via lines 98. When the count in counter 79 matches the number stored in the storage means 78, a signal occurs at the output of the comparator 83 on line 100. In the currently preferred embodiment, the lookahead storage means 78 stores a digital number which can be changed (typically by software).

The multiplexer 87 selects between the output of multiplexers 84 and 85, and lines 112. The address on lines 112 are received via the NUBUS from CPU. They are used to load the video RAMs in an ordinary manner. The addresses from the multiplexers 84 and 85 are the addresses used during scanning (screen refresh).

RAM bank select 86 receives additional information and decodes it in an ordinary manner to select between bank zero and bank one of the memory arrays. For purposes of the following discussion, the particular bank selected is not critical.

OPERATION OF THE ADDRESS GENERATOR

Assume now that the VRAM arrays contain pixel data for the display. (This data, as mentioned, is received on the data bus 29 with addresses from lines 112 which are then coupled to the VRAMs through the bus 28 of FIG. 2. The CPU provides a base address which corresponds, by way of example, to the location for data for the upper lefthand corner 70 of the display 67 of FIG. 6. This address need not correspond to the beginning of a row line in memory; that is, there can be a column address so that data for pixel 70 begins midway in the shift register. This base address is coupled on lines 88 to the adder 77. Since this is a new frame (assume odd lines) zero is coupled on lines 89 to the adder. The output from the adder 77 comprises the base address which is coupled to registers 81 and 82 and also loaded into counters 79 and 80. The multiplexers 84 and 85 select this address and it is coupled to the VRAMs. As the data is clocked from the shift register (e.g., shift register 64 of FIG. 6) the counter 79 is incremented. Data words of 32 bits are coupled from the VRAM with each shifting of the shift registers. If 8 bits per pixel are used, then counter 79 is incremented at one-fourth the pixel clock rate. Similarly, if one bit per pixel is used, the counter 79 is incremented at 1/32 the rate of the pixel clock. (In fact, the shift register can operate synchronously from the pixel clock so long as data is accessed at a rate fast enough to meet the demands of the display mode. Temporary storage or buffers may then be necessary.)

When the counter 79 reaches the predetermined count (e.g., 256), the last stages of the shift register is being accessed. The signal on line 101 causes the row multiplexer 84 to select the address on lines 96. For the example, this is the base row address incremented by one; that is, the next row in memory. Also, the signal on line 101 causes the multiplexer 85 to select lines 95 and the first stages of the shift register is selected. Additionally, counter 79 is reset (zero count).

For each scan line thereafter, the row address from row address register 82 and the column address from register 81 are added to the offset on lines 89. The new address is then coupled to registers 81 and 82 and selected by multiplexers 84 and 85.

When odd scan lines are displayed, the offset 89 is added to the base address after the first line as described above (except for scan line 1 where base address is used). That is, for scan line 3, the address on lines 90 and

91 (which is the base address) is added to the offset to obtain the next line. For line 5, the offset is added to the address on lines 90 and 91 which corresponds to scan line 3, thereby providing the starting address for scan line 5, etc.

For even scan lines, the location in the VRAM for scan line 2 must be addressed at the start of the frame. Here one-half the length on lines 108 is added to the base address on lines 88 to obtain the address for scan line 2. This address from lines 90 and 91 is added to the full length (offset on line 89) to provide the address for scan line 4 and the remaining scan lines in the frame.

Thus, to summarize for odd lines the offset is initially zero, whereas for even scan lines, the offset is initially one-half the length. It will be apparent that for non-interlaced displays the even-odd signal is not required and the length on lines 108 corresponds to the length of data between consecutive scan lines on the display.

Referring now to FIG. 6, the importance of the address generation of FIG. 5 can be more readily appreciated. Assume that scan line 75 of display 67 is being scanned. Further assume that the address coupled to registers 81 and 82 correspond to row 66 of the array 63 and column location 68. This entire row is transferred into the shift register and the first data from the shift register corresponding to the column location 68. This provides the pixel data for pixel 69 of scan line 75. As the data is shifted from the shift register 64, it is used through, of course, the color lookup table to provide the video signal as needed to paint line 75. The counter 79 is incremented; for this case the number of counts needed to reach 256 corresponding to bracket 59. When the end of the shift register is reached, data is loaded from the next row in the array shown as row 660 (this address is from counter 80). Now the column address is zero, selected by multiplexer 85 from lines 95. The data at location 72 provides the pixel data for pixel 74 of line 75.

Thus, the data for pixel 73 came from the end of row 66 as indicated by line 71. The data for the next pixel 74 came from the next row (row 660) but from the beginning of the shift register (column 72). The significance of this is that the storage of the data within the array 63 is not necessarily mapped with a fixed number of rows corresponding to a fixed number of scan lines. This allows the data to be more efficiently stored within array 63.

Memory cycle time is required to address a row and transfer data from the row into the shift register. This is a relatively long time when compared to the pixel rate. The present invention provides a lookahead feature to alert the system to the fact that the end of the data in the shift register is approaching. The line 105 of display 67 is used to illustrate that before the data for pixel 73 is reached a lookahead mechanism is activated.

The lookahead mechanism employs the lookahead storage 78 of FIG. 5. This number is stored, as mentioned, and compared with the contents of counter 79. Before the end of the shift register is reached, a signal occurs on line 100. This signal is used as a RAM control signal for the start of a time sequence to transfer data into the shift register. (The DTOE signal can be asserted while data is being shifted out of the shift register to permit rapid transfer of data from the next row into the shift register.) This provides a smooth transition of data from row-to-row of the memory array. The lookahead 105 of FIG. 6 is programmable, that is, a longer lookahead is used where more pixel data is needed (e.g., 8 bits per pixel) and a shorter time is used where less pixel data is needed (e.g., 1 bit per pixel).

In the currently preferred embodiment, comparator 83 examines the six most significant bits of the counter 79 and storage means 80 is programmable from 3 to 6 bits.

Thus, an address generator has been described which makes very efficient use of video RAMs and permits the pixel data to be stored in the video RAMs without having an integer number of scan lines per row of memory.

I claim:

1. A video apparatus for providing video data for a display when coupled to a computer which includes a central processing unit comprising:

interface means for interfacing with said central processing unit;

a pixel data memory having a plurality of video random-access memories each of which includes a storage array and a shift register;

addressing means for addressing said video random-access memories coupled between said interface means and said pixel data memory comprising:

(a) row address storage means for storing a row address;

(b) column address storage means for storing a column address;

(c) row counter means for incrementing said row addresses;

(d) column counter means for receiving said column addresses;

(e) control means for selecting said row counter means when said column counter reaches a predetermined count, said control means resetting said column counter means when said row counter means is incremented;

(f) comparator means for comparing the count in said column counter means with a certain count which is less than said predetermined count so as to provide a signal before said shift registers are emptied, said signal causing new address signals to be generated for said video random-access memories;

said pixel data memory being accessed by said addressing means to provide video data for said display.

2. The video apparatus defined by claim 1 including an adder coupled to receive a control input, a base address, an offset and the outputs of said storage means, the output of said adder being coupled to said storage means.

3. The video apparatus defined by claim 2 wherein the outputs of said row and column address storage means are added to said offset for new scan lines of said display.

4. The apparatus defined by claim 3 wherein said certain count, is programmable.

5. A video apparatus for providing video data for a display when coupled to a computer which includes a 68020 central processing unit and a main memory, said computer including a NUBUS which communicates with said central processing unit and said main memory, comprising:

interface means for selectively interfacing with each of said NUBUS or said 68020 central processing unit;

a pixel data memory having a plurality of video random-access memories each of which includes a storage array and a shift register;

addressing means for addressing said video random-access memories coupled between said interface means and said pixel data memory comprising:

(a) row address storage means for storing a row address;

(b) column address storage means for storing a column address;

(c) a row counter coupled to receive said row address;

(d) a column counter coupled to receive said column address;

(e) control means for causing said row counter to increment when said column counter reaches a predetermined count;

(f) comparator means for comparing the count in said column counter with a certain count which is less than said predetermined count so as to provide a signal before the shift registers in said video random-access memories are emptied, said signal causing new address signals to be generated for said video random-access memories:

said pixel data memory being accessed by said addressing means to provide video data for said display.

6. The apparatus defined by claim 5 wherein said certain count is programmable.

7. A video apparatus for providing video data for a display when coupled to a computer which includes a central processing unit and a main memory comprising:

interface means for interfacing with said central processing unit;

a pixel data memory having a plurality of video random-access memories each of which includes a storage array and a shift register;

addressing means for addressing said video random-access memories coupled between said interface means and said pixel data memory comprising:

(a) an adder for receiving a base address and an offset;

(b) row address storage means for storing a row address received from said adder;

(c) column address storage means for storing a column address received from said adder, wherein the outputs of said row and column storage means provide an additional input to said adder;

(d) a row counter coupled to receive said row address;

(e) a column counter coupled to receive said column address, and clocked at the rate data is shifted from the shift register of said video random-access memories;

(f) control means for causing said row counter to increment when said column counter reaches a predetermined count, said control means resetting said column counter when said row counter is incremented;

(g) comparator means for comparing the count in said column counter with a certain count which is less than said predetermined count so as to provide a signal before the shift registers in said video random-access memories are emptied, said signal causing said new address signals to be generated for said video random-access memories;

said pixel data memory being accessed by said addressing means to provide video data for said display.

8. The apparatus defined by claim 7 wherein said row and column address storage means receive said base address from said adder for a certain new frame of said display and for subsequent scan lines of said display said adder providing the sum of the addresses stored in said storage means and said offset.

9. The apparatus defined by claim 8 wherein said offset is a function of the number of bits of pixel data used for each scan line.

10. The apparatus defined by claim 9 wherein said certain count is programmable.

11. In a video apparatus for providing video data for a display when said apparatus is coupled to a computer which includes a central processing unit and a main memory, an improvement comprising:

a plurality of video random-access memories each of which includes a memory array which is addressed by a row address and a shift register which is addressed by a column address;

addressing means for providing said row address and column address for transfer of said video data directly from said memory array to said shift register;

detection means for providing a first signal prior to the end of the shifting of the data from said shift registers, said first signal initiating a time sequence for transfer of said video data from said memory arrays directly to said shift registers in said video random-access memories, thereby permitting either a non-integer or integer number of scan lines per row of said memory array, said detection means provides said first signal when said shift register has a certain number of bits of data remaining and said certain number of bit is programmable; and

control means for providing second signals to said video random-access memories when said first signal is received from said detection means;

said video random-access memories being addressed by said addressing means.

12. The improvement defined by claim 11 wherein said addressing means comprises:

(a) row address storage means for storing said row address;

(b) column address storage means for storing said column address;

(c) a row counter coupled to receive said row address;

(d) a column counter coupled to receive said column address;

(d) said control means including means for causing said row counter to increment when said column counter reaches a predetermined count;

(f) said detection means being coupled to said column counter.

* * * * *

United States Patent [19]

Ashkin et al.

[11] **Patent Number: 4,910,655**

[45] **Date of Patent: Mar. 20, 1990**

[54] **APPARATUS FOR TRANSFERRING SIGNALS AND DATA UNDER THE CONTROL OF A HOST COMPUTER**

[75] Inventors: **Peter B. Ashkin**, Los Gatos; **Michael Clark**, Glendale, both of Calif.

[73] Assignee: **Apple Computer, Inc.**, Cupertino, Calif.

[21] Appl. No.: **765,396**

[22] Filed: **Aug. 14, 1985**

[51] **Int. Cl.^{4}** **G06F 13/42; G06F 13/14**

[52] **U.S. Cl.** **364/200; 364/222.2; 364/242.92; 364/260.1; 364/261.2; 364/284.3; 371/57.2; 340/825.07; 340/825.52**

[58] **Field of Search** ... 364/200 MS File, 900 MS File, 364/514; 371/22, 57; 340/825.50, 825.52, 825.51, 825.53, 825.07, 825.06; 370/85, 94; 375/55

[56] **References Cited**

U.S. PATENT DOCUMENTS

3,221,307	11/1965	Manning	364/200
3,646,534	2/1972	Miller	360/40
3,715,725	2/1973	Kievit et al.	364/200
3,787,627	1/1974	Abramson et al.	370/67
3,836,888	9/1974	Boenke et al.	364/200
3,863,025	1/1975	Gonsewski et al.	375/55
3,979,723	9/1976	Hughes et al.	370/31
4,063,220	12/1977	Metcalfe et al.	340/825.5
4,071,908	1/1978	Brophy et al.	364/900
4,345,250	8/1982	Jacobsthal	371/57
4,360,870	11/1982	McVey	364/200
4,373,181	2/1983	Chisholm et al.	364/200
4,442,502	4/1984	Friend et al.	364/900
4,498,169	2/1985	Rozmus	340/825.5
4,562,535	12/1985	Vincent et al.	364/200
4,568,930	2/1986	Livingston et al.	340/825.52
4,570,220	2/1986	Tetrick et al.	364/200
4,589,063	5/1986	Shah et al.	364/200
4,595,921	6/1986	Wang et al.	340/825.08
4,608,559	8/1986	Friedman et al.	340/825.5
4,608,689	8/1986	Sato	364/200 X
4,611,274	9/1986	Machino et al.	364/200
4,620,278	10/1986	Ellsworth et al.	364/200
4,626,846	12/1986	Parker et al.	340/825.52
4,628,478	12/1986	Henderson, Jr.	364/900
4,638,313	1/1987	Sherwood, Jr. et al.	340/825.52
4,660,141	4/1987	Ceccon et al.	364/200
4,667,193	5/1987	Cotie et al.	340/825.08
4,675,813	6/1987	Locke	364/200
4,677,613	6/1987	Salmond et al.	370/85
4,701,878	10/1987	Gunket et al.	364/900
4,710,893	12/1987	McCutchean et al.	364/900
4,716,410	12/1987	Nozaki	340/825.52
4,760,553	7/1988	Buckley et al.	364/900
4,773,005	9/1988	Sullivan	364/200
4,775,931	10/1988	Dickie et al.	364/200

FOREIGN PATENT DOCUMENTS

0051425	8/1982	European Pat. Off. .
0104545	4/1984	European Pat. Off. .
59-52331	3/1984	Japan .
1508854	4/1978	United Kingdom .
1518565	7/1978	United Kingdom .
2035636	6/1980	United Kingdom .
2070826	5/1984	United Kingdom .

OTHER PUBLICATIONS

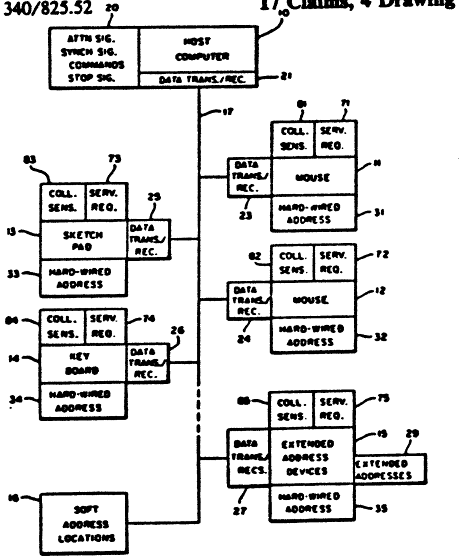
Hill et al., "Dynamic Device Address Assignment Mechanism", IBM Technical Disclosure Bulletin, vol. 23, No. 8, Jan. 1981, pp. 3564–3565.

Primary Examiner—Thomas C. Lee
Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman

[57] **ABSTRACT**

A communications medium for transmitting data between a plurality of peripheral devices and a host computer. Only one device may talk on the bus at a time and only in response to a command from the host. When a peripheral device needs to be serviced, it sends out a service request signal by holding the bus low after any command signal. The device continues to request service until it receives a talk command from the host computer. When more than one device are the same type (for example, two mice) are coupled to the bus, the host computer assigns new addresses to the devices. Devices detect collisions by sensing a low signal on the bus when they attempt to send a "1".

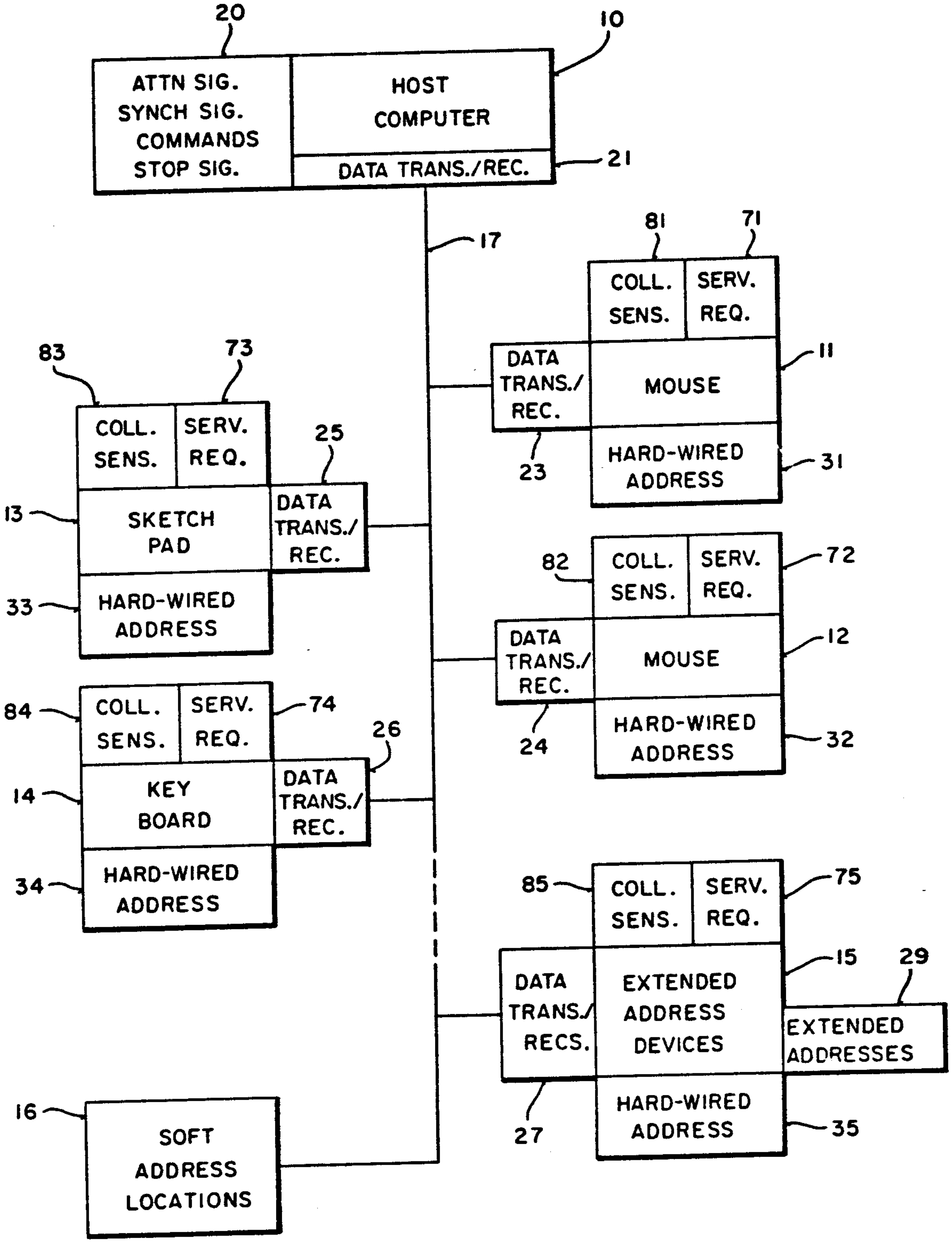
17 Claims, 4 Drawing Sheets



ADB
(Apple
Desktop
Bus)

"APPLE_PAT_4_910_655_01" 788 KB 2000-02-22 dpi: 600h x 600v pix: 4049h x 6101v

FIG_1



"APPLE_PAT_4_910_655_02" 120 KB 2000-02-22 dpi: 300h x 300v pix: 1973h x 2749v

FIG_6

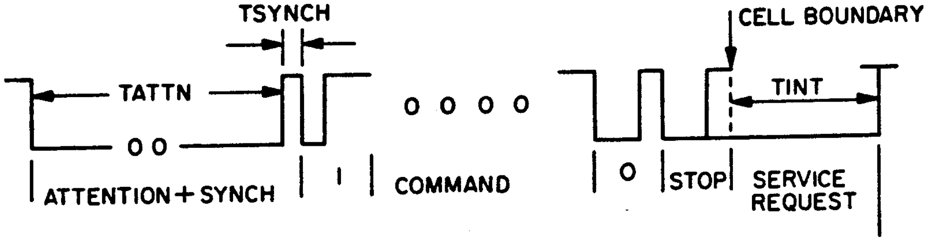


FIG_2

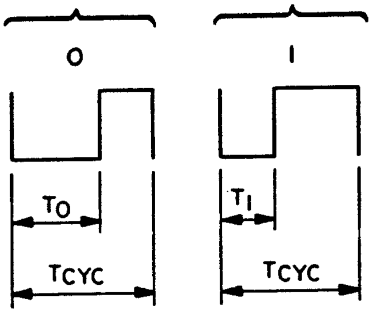
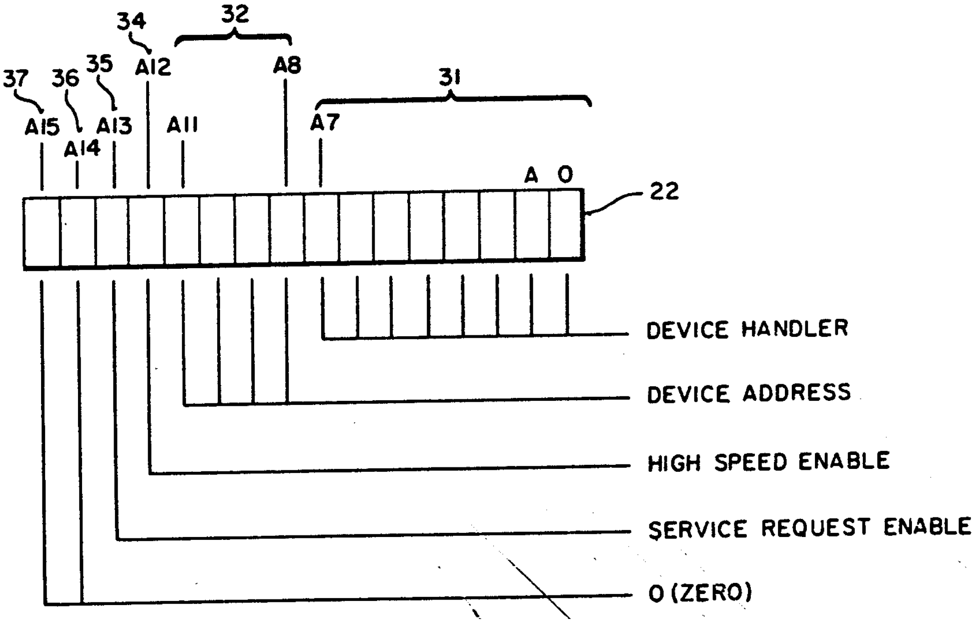
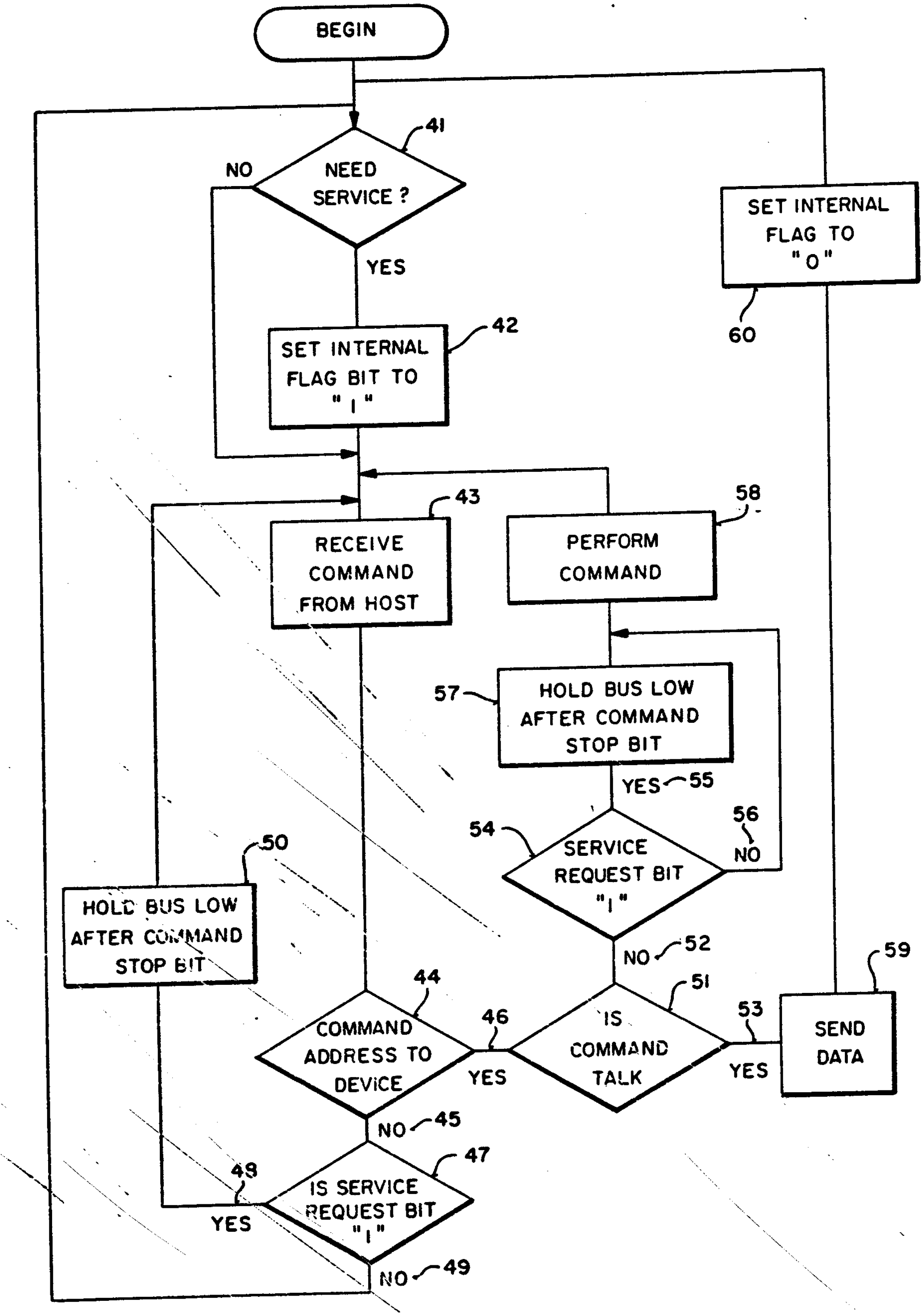


FIG_3



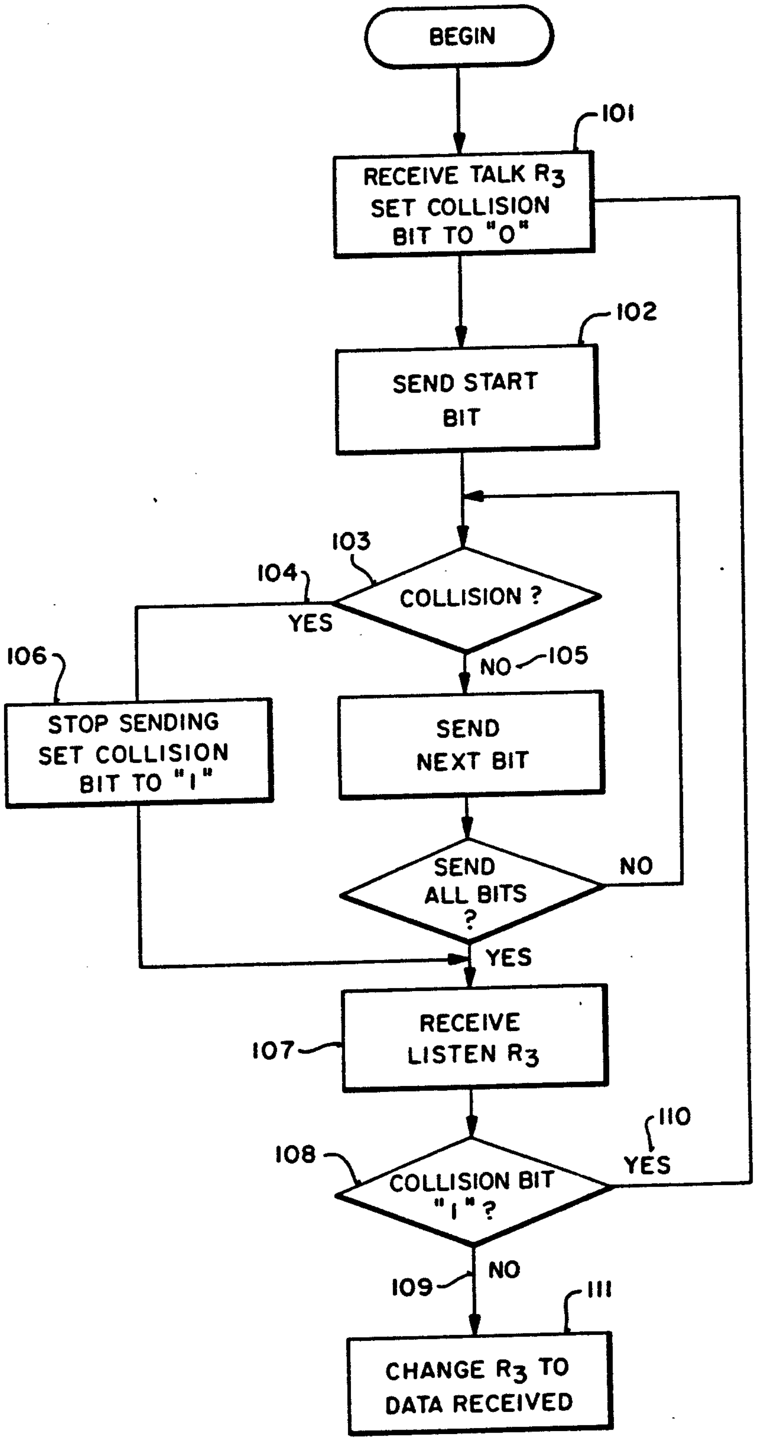
"APPLE_PAT_4_910_655_03" 102 KB 2000-02-22 dpi: 300h x 300v pix: 1923h x 3038v

FIG _ 4


BEGIN
41
NEED SERVICE ?
NO
YES
42
SET INTERNAL FLAG BIT TO "1"
SET INTERNAL FLAG TO "0"
60
43
RECEIVE COMMAND FROM HOST
58
PERFORM COMMAND
57
HOLD BUS LOW AFTER COMMAND STOP BIT
YES
55
54
SERVICE REQUEST BIT "1"
56
NO
NO
52
51
IS COMMAND TALK
53
YES
SEND DATA
59
50
HOLD BUS LOW AFTER COMMAND STOP BIT
44
COMMAND ADDRESS TO DEVICE
46
YES
NO
45
47
IS SERVICE REQUEST BIT "1"
48
YES
NO
49

"APPLE_PAT_4_910_655_04" 134 KB 2000-02-22 dpi: 300h x 300v pix: 1905h x 3119v

FIG_5


BEGIN
101
RECEIVE TALK R_3 SET COLLISION BIT TO "0"
102
SEND START BIT
103
COLLISION ?
104
YES
NO
105
106
STOP SENDING SET COLLISION BIT TO "1"
SEND NEXT BIT
SEND ALL BITS ?
NO
YES
RECEIVE LISTEN R_3
107
108
COLLISION BIT "1" ?
110
YES
109
NO
111
CHANGE R_3 TO DATA RECEIVED

APPARATUS FOR TRANSFERRING SIGNALS AND DATA UNDER THE CONTROL OF A HOST COMPUTER

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention relates the field of communications media for transferring data between a source and a plurality of peripheral devices coupled to the source. More particularly, the present invention relates to data transfer along a peripheral device bus between a plurality of peripheral devices and a host computer.

2. Art Background

In the computing industry, it is quite common to transfer data and commands between a plurality of data processing devices, such as for example, computers, printers, memories and the like. The interconnection of computers and other peripheral devices principally developed in the early 1970's with the advent of computer networking systems, which permitted the distribution of access to computing resources beyond the immediate proximity of a main frame computer.

Networks, such as the ARPA network, were developed to provide access by various users to large time-sharing systems and the transfer of data between such systems. In the case of geographically local networks, so-called "local area networks" (LANs) were developed to connect together a collection of computers, terminals and peripherals located, typically in the same building or adjacent buildings, and permitted each of these devices to communicate among themselves or with devices attached to other networks. Local area networks permit the implementation of distributed computing. In other words, some of the devices coupled to the local area network may be dedicated to perform specific functions, such as file storage, data base management, terminal handling, and so on. By having different machines perform different tasks, distributed computing can make the implementation of the system simplier and more efficient.

Presently, networking has only been applied to provide communications between data processing devices, which are machine input devices. However, it would also be useful to provide a networking means to provide communication between a single computer and a plurality of peripheral devices such as human input devices, listen only devices, appliances, etc. Human input devics include keyboards, cursor control devices (such as a "mouse"), and sketch pads, etc. Listen only devices include transaction logs, etc. In the prior art, such devices are attached to a host computer through a port dedicated to each device. Often, additional "cards" are required to allow a peripheral input device to be added. Further, the addition of cards requires that the host computer be powered down, with no mechanism for adding peripheral devices to a live system. Such prior art systems are inefficient since peripheral devices are not generally operated simultaneously. (for example, someone using a mouse is generally not using the keyboard or sketchpad at the same time). Thus, the devices could share a common line to the host computer without creating data traffic problems, eliminating the need for cards.

Prior art networking schemes also include elaborate methods for establishing control of the network to allow a device to transmit. Such systems are not needed for networking of peripheral devices, since only one is generally used at a time. In addition, prior art networking schemes provide for means for attached devices to identify themselves to each other through elaborate "handshaking" schemes. Again, such complexity is not required to connect peripheral devices since there is no need for these devices to identify themselves to other devices, only to the host computer.

Therefore, it is an object of the present invention to provide a communications medium for a plurality of peripheral devices, which provides a simple and efficient means for coupling those devices to a host computer.

It is a further object of the present invention to provide a communications medium by which all such peripheral devices can be coupled to a host computer at a single input.

It is still another object of the present invention to provide a communications medium which provides a means for peripheral devices to indicate a need for servicing to the host computer.

It is yet another object of the present invention to provide a communications medium which provides a means for determining if the communications medium is in use.

It is another object of the present invention to provide a communications medium which allows peripheral devices to be added during operation of the system.

SUMMARY OF THE INVENTION

A communications medium is disclosed including apparatus and methods for transferring data between a plurality of peripheral devices and a host computer. In the preferred embodiment, a plurality of peripheral devices such as human input devices (including mice, keyboards, sketchpads, etc.), appliances, listen only devices, etc., are coupled to a common cable for data transmission and reception of commands. A peripheral device coupled to the cable may signal the host computer when it requires servicing. This peripheral device will continue to request serivce until the host computer commands it to transmit its data. All peripheral devices of the same generic type (e.g., all keyboards), may have an identical hard wired address used as an indentifcation number. In this manner, the host computer can identify the generic type of device communicating on the cable. If more than one of the same type of device is coupled to the cable (e.g., 2 mice), the host computer will assign new addresses in the status registers of the mice so they can be differentiated.

In the preferred embodiment, a return to zero modulation scheme is used to transmit data and commands over the cable. As a result, a peripheral device will assume a collision if it attempts to transmit a high signal on the cable and the cable is pulled low by another device. In order to simplify the protocol of the system, only the computer can initiate communication.

The present invention permits the addition of peripheral devices to a computer while the computer is in use, without the need to power down the computer system. The present invention can be embodied in a narrow band medium, as well as broad band, fiber optic, infrared and other media.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is block diagram illustrating the networking system of the present invention.

FIG. 2 is a timing diagram illustrating the present invention's use of return to zero encoding.

FIG. 3 illustrates a register of a peripheral device of the present invention.

FIG. 4 is a flow chart illustrating the sequence of operations utilized by a peripheral device to request service by the host computer.

FIG. 5 is a flow chart illustrating the sequence the operations utilized to provide new addresses to devices sharing the same hard-wired address.

FIG. 6 is a timing diagram illustrating a command transaction of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

A peripheral device bus including apparatus and methods for transferring data between a plurality of peripheral devices coupled to a host computer is disclosed. In the following description numerous specific details are set forth, such as specific numbers, registers, addresses, times, signals, and formats, etc., in order to provide a thorough understanding of the present invention. However, it will be apparent to one skilled in the art that the present invention may be practiced without these specific details. In other instances, well-known circuits and devices are shown in block diagram form in order not to unnecessarily obscure the present invention.

Referring to FIG. 1, the preferred embodiment of the present invention may be seen. A plurality of peripheral devices, generally identified by numbers 11 through 16 are coupled through a single cable 7 to a host computer 10. In the preferred embodiment, all devices communicate with the host computer by a mini-phono jack with the following connector assignments; tip-power, ring-data, sleeve-power return. A "high" signal (1) is 2.4 volts minimum. A "low" signal (0) is 0.8 volts maximum. Although a single cable is contemplated in the preferred embodiment of the present invention, other communications media, such as broad band methods, fiber optic systems, and infrared signals, are contemplated.

The bus of the present invention supports coded devices (for which a keystroke represents a symbol or a function, such as a keyboard 14), relative devices (in which movement of a display cursor in response to a control device, such as a mouse 11 or 12, may be from any starting point), and absolute devices (for which there is a constant and direct relationship between display position and device position, such as sketch pad 13).

The system also permits the networking of extended address devices. Extended address devices share a common hard wired address 35, but further include an address unique to the individual device which the host computer must recognize before the device can be accessed. Extended addresses 29 for respective extended address devices 15 are shown collectively in FIG. 1 by a single block diagram 29 coupled to extended address devices 15. For example, it is contemplated that appliances may be coupled to the host computer and controlled by the host computer. In such a situation, all appliances would have an identical hardwired fixed address. The host computer, on a first level, would simple address the hard wired address for appliances. At this time, all appliances coupled to that address are inactive. An individual appliance may be activated by the host computer if the host computer sends a signal to

that appliance which matches the extended address of the appliance. An extended address is an individual identification number, which, in the preferred embodiment, may be up to 64 bytes long. Once the host computer has provided the extended address, the device having that address is active. Subsequent commands to the appliance address location will be executed by that device without the need for providing the extended address each time. An activated appliance will respond to all commands to the appliance address, while unactivated devices remain passive. To deactivate an active extended address device, the host computer provides the extended address of another extended address device, activating it and deactivating the previously active device. It is contemplated that any device which could be controlled by the host computer is suitable for the present networking scheme, such as lights, ovens, sprinkler systems, phone answering machines, etc. It is contemplated that at least one other hardwired address for extended address devices be provided in the present system. Such an address would be used for system protection schemes or user identification schemes. For example, a device at this location could contain an extended address which must be provided by the system user before the system could be enabled. In other instances, individual operations could require that the extended address of other security devices be provided by the host computer prior to performance. Such security devices could function as "keys" to lock the entire system or certain operations performed on the system.

Also reserved for use on the network of the present invention are soft address locations 16. Soft address locations are reserved for duplicates of peripheral devices coupled to the bus. When more than one mouse is coupled to the bus, for example, the host computer assigns new addresses to each mouse, those addresses being at the soft address locations.

Although specific examples have been given for each type of device coupled to the bus, there may be more than one kind of each type of device with that address. For example, a sketch pad has been given as an absolute device but a touch screen would also be considered an absolute device and be assigned the same fixed command address as the sketch pad. In those situations, the host computer will assign new addresses from the soft address locations to each device.

In the preferred embodiment of the present invention, the various peripheral devices have been assigned addresses as shown below:

Address	Device Types	Example
0000 (zero)	extended address device	security systems, user ID
0001 (one)	extended address device	appliances
0010 (two)	coded devices	keyboard
0011 (three)	relative devices	mouse, track ball
0100 (four)	absolute devices	sketchpad, touch screen
0101 (five)	reserved	none
0110 (six)	reserved	none
0111 (seven)	reserved	none
1000 (eight)	soft addressed	duplicate peripheral devices
. . .	. . .	. . .
. . .	. . .	. . .
1111 (15)	soft addressed	duplicate peripheral devices

It will appreciated by one skilled in the art that other addresses may be assigned to these devices containing more or less bits than in the preferred embodiment. Fixed hart-wired addresses 31, 32, 33 and 34 are shown in FIG. 1 for mouse 11, mouse 12, sketch pad 13, and keyboard 14, respectively.

In FIG. 1, data transmitter/receiver circuit 23 for mouse 11 is shown in block diagram form coupled to mouse 11. Data transmitter/receiver circuit 24 for mouse 12 is shown in block diagram form coupled to mouse 12. Data transmitter/receiver circuit 25 for sketch pad 13 is shown in block diagram form coupled to sketch pad 13. Data transmitter/receiver circuit 26 for keyboard 14 is shown in block diagram form coupled to keyboard 14. Data transmitter/receiver circuits 27 for respective extended address devices 15 are shown collectively in FIG. 1 by a single block diagram 27 coupled to extended address devices 15.

All peripheral devices have four registers in the preferred embodiment to receive data and send data. For each device, register 3 talk and register 3 listen have status information such as device address and handler information. The remaining registers are data registers which are device specific except register 2 listen which contains the extended addresses for extended address devices or device specific contents for soft addressed devices.

In the preferred embodiment of the present invention, there are three types of communication on the peripheral bus: commands, data and global signals. Commands are sent from the host computer to the peripheral devices, data is sent from the host computer to the devices or from the devices to the host computer, and global signals are special messages sent to the entire system.

In the preferred embodiment data is encoded as the ratio of low time to high time of each bit cell. A bit cell boundary is defined by a falling edge on the bus. A "zero" is encoded as a bit cell in which the low time is greater than the high time. This is shown in FIG. 2 by bit cell 20. Therefore, a "1" is defined as a bit cell in which the low time is less that the high time as shown by cell 21 of FIG. 2. In the present preferred embodiment, a start bit is defined as a "1". A stop bit is a "0" which does not have an additional falling edge to define the bit cell time. The stop bit is used to synchronize the stopping of transactions on the bus.

The period for each bit cell of command signals and low speed data transmission is approximately 100 microseconds plus or minus 30%. For high speed data transmission, the bit cell is 50 microseconds plus or minus 1%. The format of a data transaction is a start bit (1), followed by up to 256 bits of data and ending with a stop bit. It will be appreciated that when other communications media are utilized, other signaling methods may be utilized.

Commands are sent only by the host. In the preferred embodiment of the present invention, there are three commands; talk, listen, and flush. As shown in FIG. 6, to signal the start of a command, an attention pulse is sent out. An attention pulse is generated by the host computer by transmitting a bus low for a period of "T-attn". In the preferred embodiment, T-attn is approximately 560-1040 microseconds. The attention pulse is followed by a synch pulse to give the initial bus timing. The following edge of the synch pulse is used as a timing reference for the first bit of the command. The command is followed by a stop bit, (in the preferred embodiment a "0"). After the stop bit, the bus returns to its normally high state unless a device requests service.

The command is an 8 bit value in the preferred embodiment. The command includes a 4 bit device address field which specifies the fixed hardwired address of the desired peripheral device (e.g., 0011 for a mouse). The next 2 bits form the command and the final 2 bits form a register address field which allows a specific register, R0-R3 within an addressed peripheral device to be specified. In the preferred embodiment, the commands have the following bit code:

Command	Code
Flush	01
Listen	10
Talk	11

The talk command orders the addressed device to provide its data to the host computer. The listen command orders the addressed device to accept data from the host computer and place it in one of its registers. The flush command has an effect on each device which is defined by the individual device. It can be used for such functions as clearing a register or resetting all keys on a keyboard so that they will be sent again. Generator 20 for generating the attention signal, the synchronization signal, the commands, and the stop signal is shown in FIG. 1 coupled to host computer 10. In FIG. 1, circuitry 21 for transmitting data from host computer 10 and receiving data from the peripheral devices is shown in block diagram form coupled to host computer 10.

When a peripheral devices is addressed to talk, it must respond within a certain period, called the "time out" period. The time out, "Tlt", is approximately 140 to 260 microseconds (2 bit cells). The selected device, if it does not time out, becomes active on the bus and performs its data transaction, and then "untalks" itself and goes inactive on the bus.

Global signals are used for transactions which are neither commands nor data transactions. Global signals include: attention and synch, which is used to signal the start of a command and to give initial bus timing; service request, a transaction that devices use to signal the host that they require service; and reset, used to issue a break on the bus by holding the bus low for a minimum of "Tres", which is approximately 2.8 to 5.2 milliseconds, (40 bit cells). Global signals will be described in more detail in conjunction with other transactions.

Since a peripheral device can only send data when it has been commanded to talk by the host computer, the present system provides a means for a device to notify the host computer that it needs servicing. This is accomplished by having the device send a service request signal to the host computer. In the present invention, a service request is sent by holding the bus low after the stop bit of any command transaction. Each of the peripheral devices coupled to the bus include a number of registers (in the preferred embodiment four registers). FIG. 3 shows one of the registers for a peripheral device. Bit A13 has been identified as the service request enable bit. When this bit is set high by the host computer, the device is enabled to hold the bus low after the stop bit of a command transaction, as shown in FIG. 6, if the device needs service. A device will keep requesting service until it receives a talk command from the host. The flow chart in FIG. 4 shows the steps followed by a device requiring service.

Initially the device determines if it requires servicing, Block 41, that is, if it has data to send to the host. If it does, it sets an internal flag bit, Block 42: When the next command is sent out from the host, Block 43, the device checks to see if the command is addressed to the device, Block 44. If the command was not addressed to the device, Branch 45, the device checks to see if its service request enable bit, (bit A13 of register 3), is set high, Branch 47. If so, Branch 48, it holds the bus low after the command stop bit, Block 50. (See FIG. 6) The device then waits until the next command is received from the host to see if it will be addressed to talk, Block 43. If the command is addressed to the device, Branch 46, the device determines if it is a command to talk, Block 51. If it is not a command to talk, Branch 52, the device sends a service request, Block 57, performs whatever command is instructed, Block 58, and awaits the next command, Block 43. If the command is to talk, Branch 53, the device sends its data, Block 59, and considers its service request to be satisfied, Block 60. The device continues to monitor itself to determine when it needs service, Block 41. By allowing the host computer to control the service request enable bit, more efficient operation of the bus is realized. When a service request is received, the host computer need only ask those devices whose service request bit was enabled whether they need servicing. Additionally, the host computer can disable certain devices that are not required for particular applications.

In FIG. 1, service request signal generator 71 for mouse 11 is shown in block diagram form coupled to mouse 11. Service request signal generator 72 for mouse 12 is shown in block diagram form coupled to mouse 12. Service request signal generator 73 for sketch pad 13 is shown in block diagram form coupled to sketch pad 13. Service request signal generator 74 for keyboard 14 is shown in block diagram form coupled to keyboard 14. Service request signal generators 75 for respective extended address devices 15 are shown collectively in FIG. 1 by a single block diagram 27 coupled to extended address devices 15.

When sending data, the device is able to detect collisions.

In FIG. 1, collision sensing circuit 81 for mouse 11 is shown in block diagram form coupled to mouse 11. Collision sensing circuit 82 for mouse 12 is shown in block diagram form coupled to mouse 12. Collision sensing circuit 83 for sketch pad 13 is shown in block diagram form coupled to sketch pad 13. Collision sensing circuit 84 for keyboard 14 is shown in block diagram form coupled to keyboard 14. Collision sensing circuits 85 for resepective extended address devices 15 are shown collectively in FIG. 1 by a single block diagram 85 coupled to extended address devices 15. If a peripheral device tries to output a 1 and the data line is or goes to a 0, the device assumes it has lost a collision to another device. This means that another device is also sending on the bus. When this happens the losing device untalks itself from the bus and preserves the data which was being sent for retransmission. The device sets an internal flag bit if it loses a collision. Prior art peripheral devices were unable to detect collisons. This novel feature of the present invention permits more efficient operation of the communications medium. By having the device sense a collision, it can preserve the data that is transmitted and indicate to the host computer that it requires serving. Additionally, the collision detection scheme of the present invention does not require a waiting period before a collision is assumed. A device will end its transmission if the line is modulated by another device or simply not begin its transmission if the line is already in use. Further, this collision detection scheme is useful in locating multiple devices at a single hard-wired address location, such as mouse 11 and mouse 12 of FIG. 1.

In such a situation, the host will change the address of the devices by forcing a collision of devices sharing the same address. The host achieves this by issuing a talk R3 command addressed to those devices. As shown in FIG. 3, Register 3 22 (one of the registers of the device) contains the following information. Bits A0 through A7 31 contain a device handler which tells the host computer the function of a device and the use of data provided by the device. Bits A8 through A11 32, are an address field which can be changed when more than one device, having the same command address, is coupled to the bus. In that situation, one of the soft address locations are assigned to bits A8 through A11 32 which then serve as the command address for that device. Until that time, those bit locations contain a random number which aids in the detection of collisions. For example, if two mice received a talk R3 command and both began talking at the same, neither would detect a collision. However, by having random numbers in the address field 32 of register 3 22, the output of the two devices will eventually differ. When that occurs, one of the devices will detect a collision and stop talking. Bit A12 34 is a high speed enable bit which if set, provides for data transmission at the higher modulation rate (50 microseconds per bit frame). The high speed enable bit is set by the host computer. If the host computer is unable to receive data at the higher modulation rate, it sets the high speed enable bit low in each of the devices. If the host computer is able to accept data at the higher modulation rate, and the device is able to transmit at the higher rate, (that information being contained in the handler bits 31 of register 3), the host computer sets the high speed enable bit 34 high for the device. As previously mentioned, bit A13 35 is service request enable which is set by the host to enable the device to perform a service request transaction. Bits A14 36 and A15 37 are reserved for future use and are set to 0.

When a device receives a talk R3 command the device provides its status (handler and address) to the host computer. If there are two devices of the same type coupled to the bus, only one can respond since the other will detect a collision. FIG. 5 shows the method of assigning new addresses on the bus.

After receiving a talk R3 signal, Block 101, the device sends its status from Register 3. If the line goes low, the device determines that there has been a collision, Branch 104, it stops sending (untalks itself) and sets an internal flag bit to indicate a collision, Block 106. The host sends a listen R3 to the mouse address, Block 107. Each talk commend resets the internal collision flag of the device. The device checks to see if its collision bit is set, Block 108. If the collision bit is not set, Branch 109, the device changes A8 through A11 to the soft address provided by the listen R3 command, Block 111. In this manner the address of the winning device is changed with the host computer keeping track of the new address of the device. If a collision bit is detected by the device after a listen R3 command, Branch 110, the device does not change the soft address bits, but may change other fields in R3. The host computer sends out another talk R3 command, Branch 101, to see if any devices remain at the mouse address. In this situation the remaining mouse will send its start bit, Block 102, not detect a collision, Branch 105, and send its status from register 3, Block 112. The host computer will send back a listen R3 command to the mouse address, Block 107. The remaining mouse will not detect a collision bit being set in this instance, Branch 109, so it will change bits A8 through A11 of register 3 to the soft address received from the host computer, Block 111. The host computer then sends out another talk R3 command to the mouse address, Block 101. This time, since no mouse remains at that address, the bus is timed out and the host computer knows that it has assigned new addresses to each of the mice sharing the mouse address.

In one embodiment of the present invention, peripheral devices have a device on them to indicate activity called the activator. The activator can be a special key on a keyboard or a button on a mouse. When more than one of a device is coupled to the bus, the host computer can display a message requesting one of the devices to use the activator. The host can then issue a listen R3 command which will change the address of the device which is activated. In this manner individual devices can be located and assigned new addresses in multiuser applications.

Thus, a peripheral device bus has been described which allows a plurality of peripheral devices to be coupled to a host computer through a single port.

We claim:

1. An apparatus for transferring signals and data, wherein the signals and data are transferred under the control of a host computer from the host computer to first and second peripheral devices and from the first and second peripheral devices to the host computer, wherein the signals and data are transferred over a bus coupling the first and second peripheral devices to the host computer, and wherein the bus is normally in a logical first state, comprising:

means, coupled to the first peripheral device, containing a first hard-wired identification number as a first address of the first peripheral device;

means, coupled to the second peripheral device, containing a second hard-wired identification number as a first address of the second peripheral device;

means, coupled to the host computer, for generating a plurality of signals for transmission over the bus to at least one of the peripheral devices, wherein the plurality of signals comprise an attention signal, a synchronization signal, one of a plurality of commands, and a stop signal, wherein the plurality of signals allow the host computer to control at least one of the peripheral devices, and wherein each of the plurality of commands includes an address of at least one of the first and second peripheral devices to which the command is directed;

means, coupled to the first peripheral device, for generating a first service request signal for transmission to the host computer by holding the bus at a logical second state for a period of time after transmission of the stop signal, wherein the first service request signal indicates to the host computer that at least one of the peripheral devices has data to send to the host computer and requests a command from the host computer that would permit the peripheral device to transmit the data to the host computer, and wherein the transmission of the first service request signal is selectively enabled and disabled by the host computer;

means, coupled to the second peripheral device, for generating a second service request signal for transmission to the host computer by holding the bus at the logical second state for a period of time after transmission of the stop signal, wherein the second service request signal indicates to the host computer that at least one of the peripheral devices has data to send to the host computer and requests a command from the host computer that would permit the peripheral device to transmit the data to the host computer, and wherein the transmission of the service request signal is selectively enabled and disabled by the host computer;

means, coupled to the host computer, for transmitting data from the host computer to at least one of the peripheral devices over the bus;

means, coupled to the first peripheral device, for transmitting data over the bus to the host computer from the first peripheral device if and only if one of the plurality of commands received by the first peripheral device from the host computer is a command to transfer data from the first peripheral device to the host computer;

means, coupled to the second peripheral device, for transmitting data over the bus to the host computer from the second peripheral device if and only if one of the plurality of commands received by the second peripheral device is a command to transfer data from the second peripheral device to the host computer;

first collision sensing means, coupled to the first peripheral device, for setting a collision detect bit to the logical first state from the logical second state when the first peripheral device attempts to transmit data in the logical first state on the bus but the bus is in or goes to the logical second state, wherein the first peripheral device stops transmitting data after the collision detect bit is set to the logical first state, and wherein the first collision sensing means provides a control over access to the bus by the first peripheral device;

second collision sensing means, coupled to the second peripheral device, for setting a collision detect bit to the logical first state from the logical second state when the second peripheral device attempts to transmit data in the logical first state on the bus but the bus is in or goes to the logical second state, wherein the second peripheral device stops transmitting data after the collision detect bit is set to the logical first state, and wherein the second collision sensing means provides a control over access to the bus by the second peripheral device; and

means, coupled to the host computer, for storing (1) a first number at a first soft address location and (2) a second number at a second soft address location, wherein if a collision is detected and is the result of the first hard-wired identification number being the same as the second hard-wired identification number, then the host computer (1) sends the first number stored at the first soft address location as data over the bus to the first peripheral device for storage by the first peripheral device as a second address of the first peripheral device and (2) sends the second number stored at the second soft address location as data over the bus to the second peripheral device for storage by the second peripheral device as a second address of the second peripheral device, wherein the first number is different from the second number.

2. The apparatus of claim 1 for transferring signals and data, wherein

the first peripheral device includes a register for storing the first number as the second address of the first peripheral device, and

the second peripheral device includes a register for storing the second number as the second address of the second peripheral device.

3. The apparatus of claim 2 for transferring signals and data, wherein

the means for generating the first service request signal includes a first service request enable bit that can be selectively set by the host computer to the logical first state and the logical second state, wherein if the host computer sets the first service request enable bit to the logical first state, the means for generating the first service request signal is enabled to send the service request signal to the host computer, wherein if the host computer sets the first service request enable bit to the logical second state, the means for generating the first service request signal is disabled from sending the first service request signal to the host computer; and

the means for generating the second service request signal includes a second service request enable bit that can be selectively set by the host computer to the logical first state and the logical second state, wherein if the host computer sets the second service request enable bit to the logical first state, the means for generating the second service request signal is enabled to send the serivce request signal to the host computer, wherein if the host computer sets the second service request enable bit to the logical second state, the means for generating the second service request signal is disabled from sending the second service request signal to the host computer.

4. The apparatus of claim 3 for transferring signals and data, wherein

the means for transmitting data from the first peripheral device includes an internal flag bit that can be selectively set by the first peripheral device data transmitting means to the logical first state and the logical second state, wherein the first peripheral device data transmitting means sets the internal flag bit to the logical first state if the first peripheral device has data to send to the host computer, and wherein the first peripheral device data transmitting means sets the internal flag bit to the logical second state after the first peripheral device data transmitting means has sent the data to the host computer; and

the means for transmitting data from the second peripheral device includes an internal flag bit that can be selectively set by the second peripheral device data transmitting means to the logical first state and the logical second state, wherein the second peripheral device data transmitting means sets the internal flag bit to the logical state if the second peripheral device has data to send to the host computer, and wherein the second peripheral device data transmitting means sets the internal flag bit to the logical second state after the second peripheral device data transmitting means has sent the data to the host computer.

5. The apparatus of claim 4 for transferring signals and data, wherein the logical first state is a logical high state and the logical second state is a logical low state.

6. The apparatus of claim 5 for transferring signals and data, wherein at least one of the first and second peripheral devices comprises a mouse.

7. The apparatus of claim 5 for transferring signals and data, wherein at least one of the first and second peripheral devices comprises a sketch pad.

8. The apparatus of claim 5 for transferring signals and data, wherein at least one of the first and second peripheral devices comprises a keyboard.

9. The apparatus of claim 5 for transferring signals and data, wherein the first and second collision sensing means each includes pulse detection circuitry for detecting that the signal on the bus is in a logical low state.

10. The apparatus of claim 9 for transferring signals and data, wherein the signals sent over the bus are in a return-to-zero encoding format.

11. The apparatus of claim 5 for transferring signals and data, wherein the plurality of commands comprise a talk command, a listen command, and a flush command.

12. The apparatus of claim 5 for transferring signals and data, wherein

the means for transmitting data from the first peripheral device includes a first high speed enable bit that the host computer can selectively set to the logical first state and the logical second state, wherein if the first high speed enable bit is set to the logical first state, then the first peripheral device data transmitting means transmits data at a first modulation rate, wherein if the first high speed enable bit is set to the logical second state, then the first peripheral device data transmitting means transmits data at a second modulation rate; and

the means for transmitting data from the second peripheral device includes a second high speed enable bit that the host computer can selectively set to the logical first state and the logical second state, wherein if the second high speed enable bit is set to the logical first state, then the second peripheral device data transmitting means transmits data at the first modulation rate, wherein if the second high speed enable bit is set to the logical second state, then the second peripheral device data transmitting means transmits data at the second modulation rate, wherein the first modulation rate is higher than the second modulation rate.

13. The apparatus of claim 12 for transferring signals and data, wherein the logical first state is a logical high state and the logical second state is a logical low state.

14. The apparatus of claim 2 for transferring signals and data, further comprising a third peripheral device coupled to the host computer by the bus, wherein the third peripheral device includes means that includes (1) a third hard-wired identification number as a first address of the third peripheral device and (2) a register for storing a third number as an extended address of the third peripheral device, wherein when the host computer sends over the bus the first address of the third peripheral device and a signal that matches the extended address of the third peripheral device, the third peripheral device is initially activated, and wherein when the host computer then sends over the bus a subsequent command to the first address of the third peripheral device, the command is executed by the third

peripheral device without the host computer sending the extended address of the third peripheral device.

15. The apparatus of claim 14 for transferring signals and data, further comprising a fourth peripheral device coupled to the host computer by the bus, wherein the fourth peripheral device includes means that includes (1) the third hard-wired identification number as a first address of the fourth peripheral device and (2) a register for storing a fourth number as an extended address of the fourth peripheral device, wherein the fourth number is different from the third number, wherein when the host computer sends over the bus the extended address of the fourth peripheral device after the third peripheral device has been activated, the fourth peripheral device is initially activated and the third peripheral device is deactivated, and wherein when the host computer then sends over the bus a subsequent command to the first address of the fourth peripheral device, the command is executed by the fourth peripheral device without the host computer sending the extended address of the fourth peripheral device.

16. The apparatus of claim 15 for transferring signals and data, wherein at least one of the first and second peripheral devices comprises an appliance.

17. The apparatus of claim 15 for transferring signals and data, wherein at least one of the first and second peripheral devices comprises a system protection device.

* * * * *

United States Patent [19]

Smith et al.

[11] **Patent Number: 4,910,670**

[45] **Date of Patent: Mar. 20, 1990**

[54] **SOUND GENERATION AND DISK SPEED CONTROL APPARATUS FOR USE WITH COMPUTER SYSTEMS**

[75] Inventors: **Burrell C. Smith; Andrew J. Hertzfeld,** both of Palo Alto, Calif.

[73] Assignee: **Apple Computer, Inc., Cupertino, Calif.**

[21] Appl. No.: **51,938**

[22] Filed: **May 19, 1987**

Related U.S. Application Data

[63] Continuation of Ser. No. 573,132, Jan. 20, 1984, abandoned.

[51] **Int. Cl.⁴** **G06F 3/153;** G06F 3/16; G06F 12/00

[52] **U.S. Cl.** **364/200;** 360/73.08; 358/903; 273/DIG. 28; 84/1.01

[58] **Field of Search** .... 360/51, 65, 72.2, 73, 360/78, 53, 97, 73.03, 73.06, 73.07, 73.08; 364/200, 700, 410; 318/318, 341; 358/32, 86, 143, 145, 147, 903; 381/32; 273/DIG. 28; 340/384 E; 84/1.01, 3; 341/152

[56] **References Cited**

U.S. PATENT DOCUMENTS

3,987,419	10/1976	Morrill et al.	340/172.5
4,148,485	4/1979	Rains	273/86 B
4,272,649	6/1981	Pfeiffer	179/1 SM
4,296,446	10/1981	Zorbalas	36C/73
4,296,930	10/1981	Frederiksen	273/313
4,357,635	11/1982	Hasegawa	360/73.03 X
4,367,876	1/1983	Kotoyori	273/121 A
4,467,319	8/1984	Uchikoshi	341/152
4,475,228	10/1984	Vickers	381/51
4,481,852	11/1984	Makuta et al.	340/384 E X
4,482,974	11/1984	Kovalick	364/718 X
4,485,337	11/1984	Sandusky	360/73.03 X
4,49C,810	12/1984	Hon	364/900
4,492,992	1/1985	Rooney et al.	360/73.03
4,514,771	4/1985	Stark et al.	360/73
4,530,018	7/1985	Hoshino et al.	360/73
4,530,499	7/1985	Breslow et al.	273/1 GC
4,536,743	8/1985	Uchikoshi	341/152
4,538,176	8/1985	Nakajima et al.	358/86
4,558,383	12/1985	Johnson	360/77
4,569,019	2/1986	Diorio et al.	364/410
4,569,026	2/1986	Best	364/521
4,573,039	2/1986	Suzuki et al.	341/152
4,577,240	3/1986	Hedberg et al.	360/22
4,582,324	4/1986	Koza et al.	273/138 A
4,603,412	7/1986	Yamazaki	360/73.03 X

OTHER PUBLICATIONS

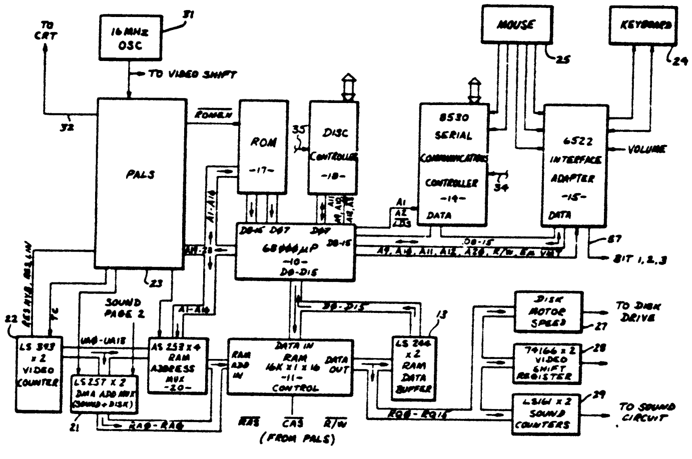
Conrad Boisvert, "Simplify CRT-system design with transparent addressing—it comes on a controller chip", *Electronic Design,* Aug. 2, 1979, pp. 90-93.

Primary Examiner—David L. Clark
Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman

[57] **ABSTRACT**

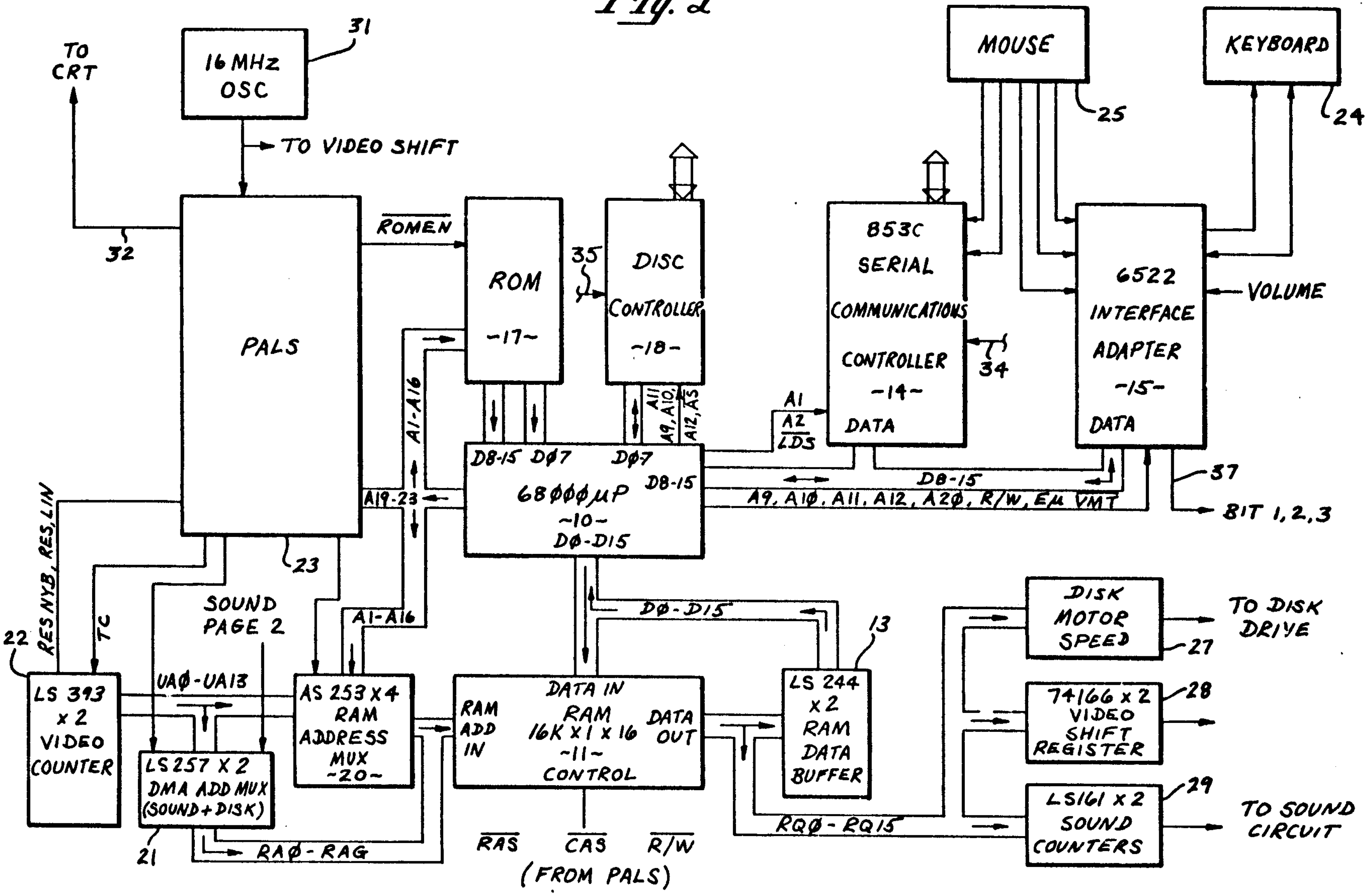
An apparatus for generating an analog audio signal and a speed control signal for a disk drive for use with a microprocessor having a RAM which provides a video signal for a raster scanned display. Direct addressing to the RAM is used during horizontal blanking periods to allow data to be read from the RAM and used to generate the audio and speed control signals. The data is updated during the blanking periods. The speed control signal to the disk controller varies as a function of track (radius).

11 Claims, 6 Drawing Sheets



MACINTOSH

Fig. 1



“APPLE_PAT_4_910_670_02” 181 KB 2000-02-21 dpi: 300h x 300v pix: 1840h x 2903v

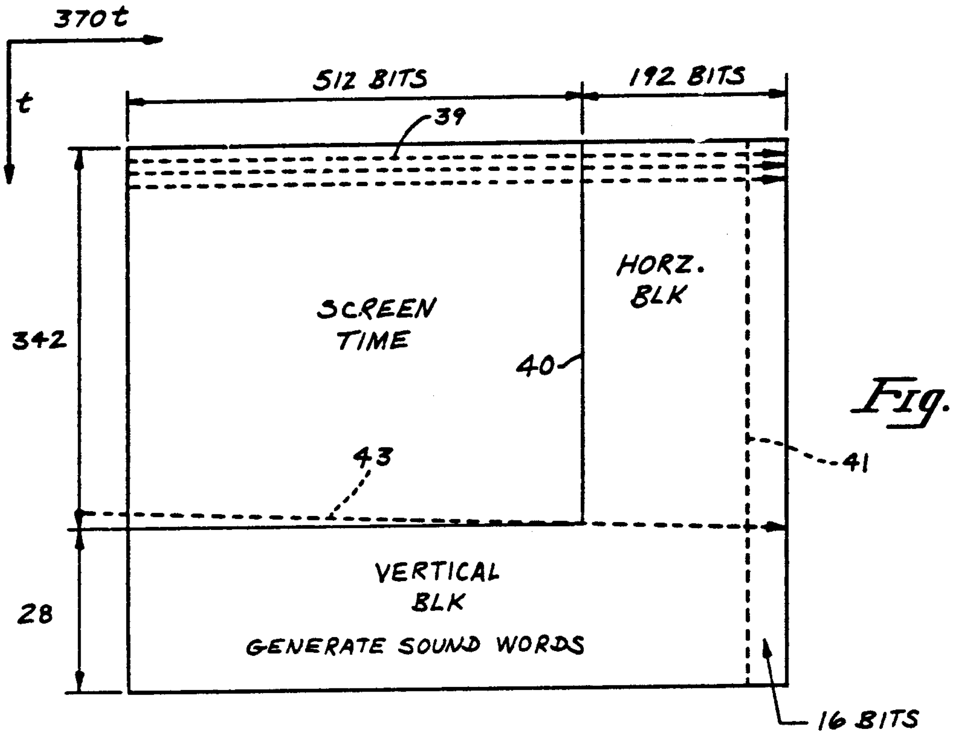


Fig. 2

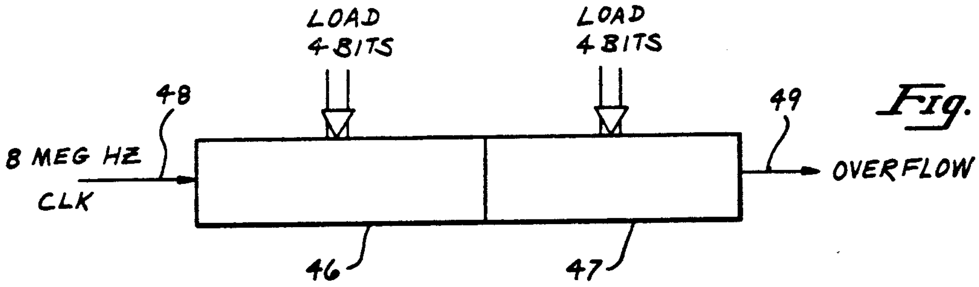


Fig. 3

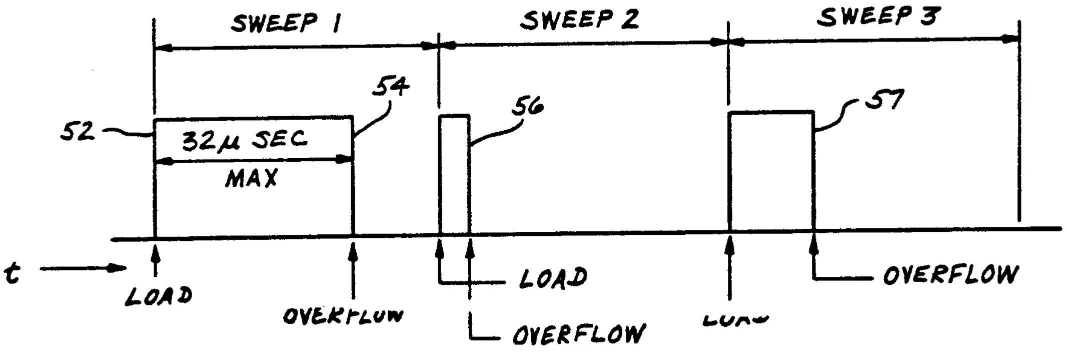


Fig. 4

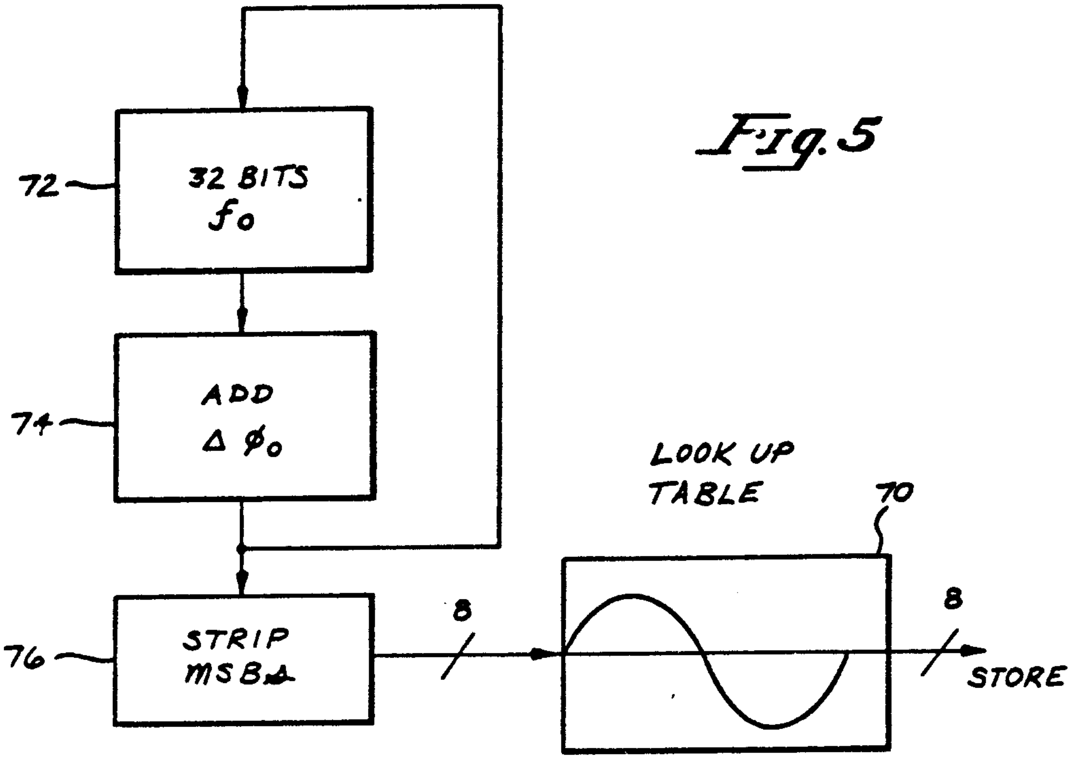


Fig. 5

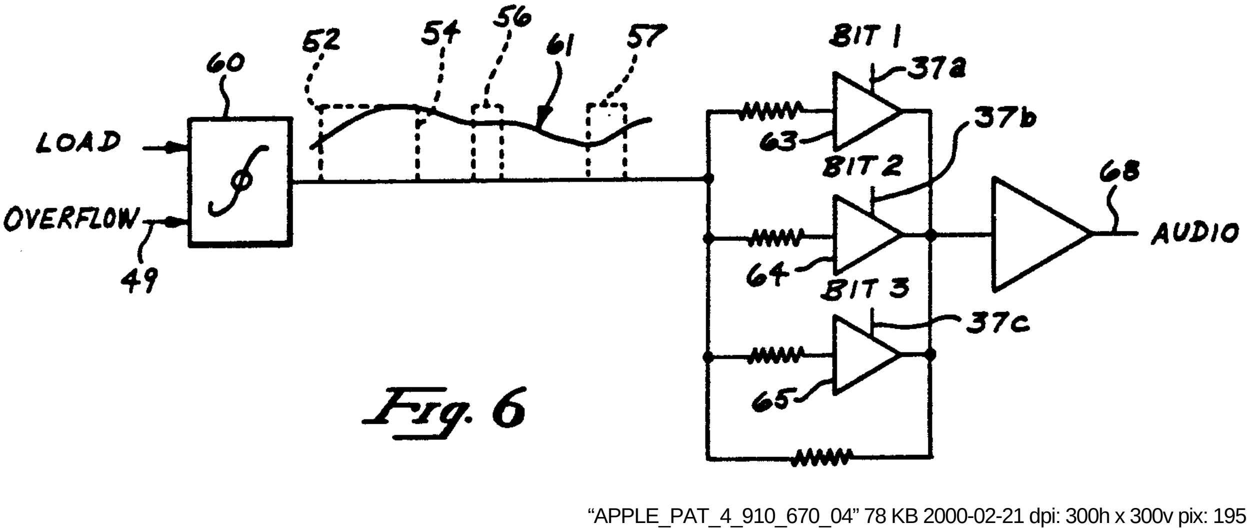
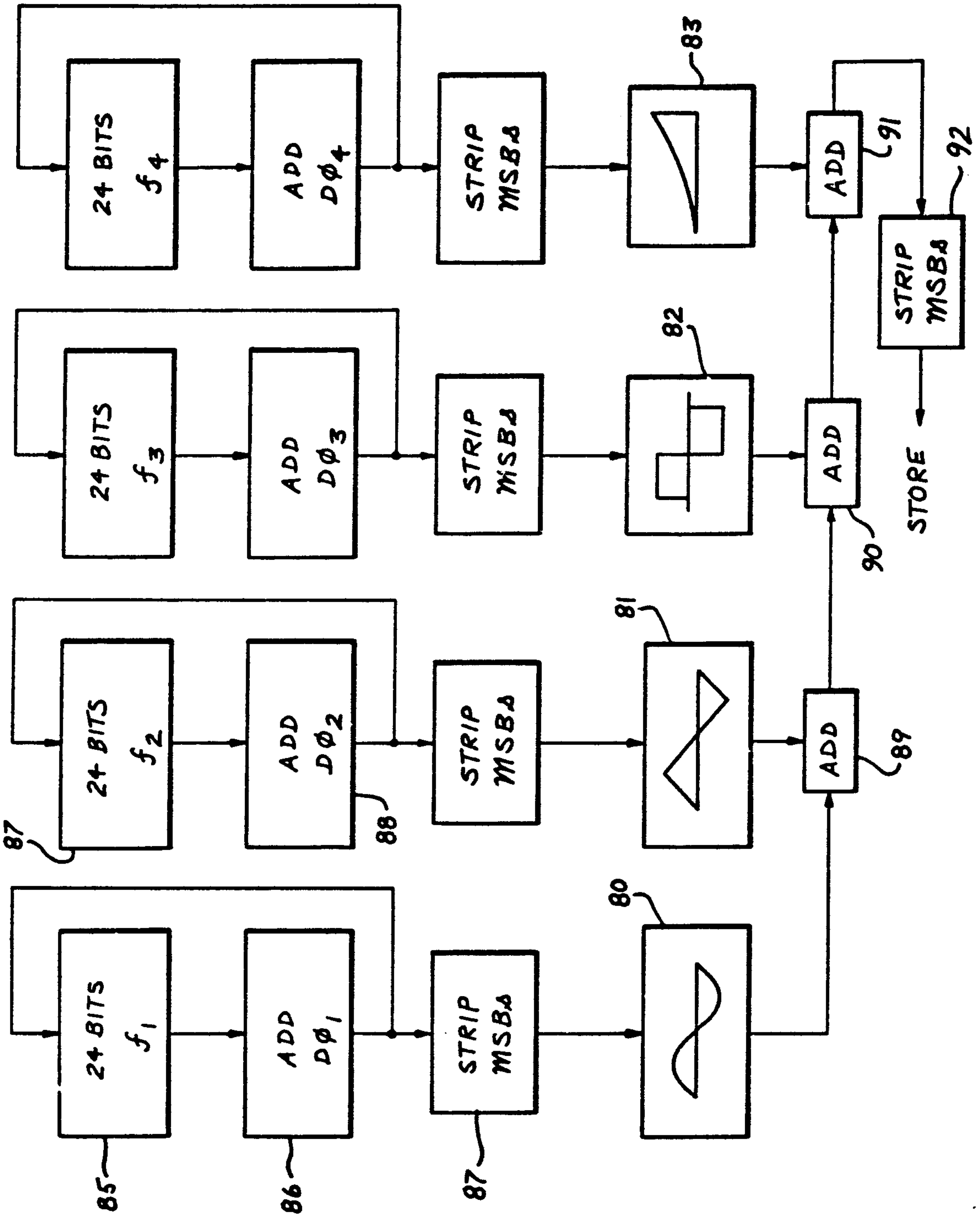


Fig. 6

“APPLE_PAT_4_910_670_04” 78 KB 2000-02-21 dpi: 300h x 300v pix: 1955h x 2585v

Fig. 7


24 BITS f_4
ADD DΦ4
STRIP MSBs
83
ADD
91
92
STRIP MSBs
STORE
24 BITS f_3
ADD DΦ3
STRIP MSBs
82
ADD
90
24 BITS f_2
ADD DΦ2
STRIP MSBs
81
ADD
89
87
88
24 BITS f_1
ADD DΦ1
STRIP MSBs
80
85
86
87

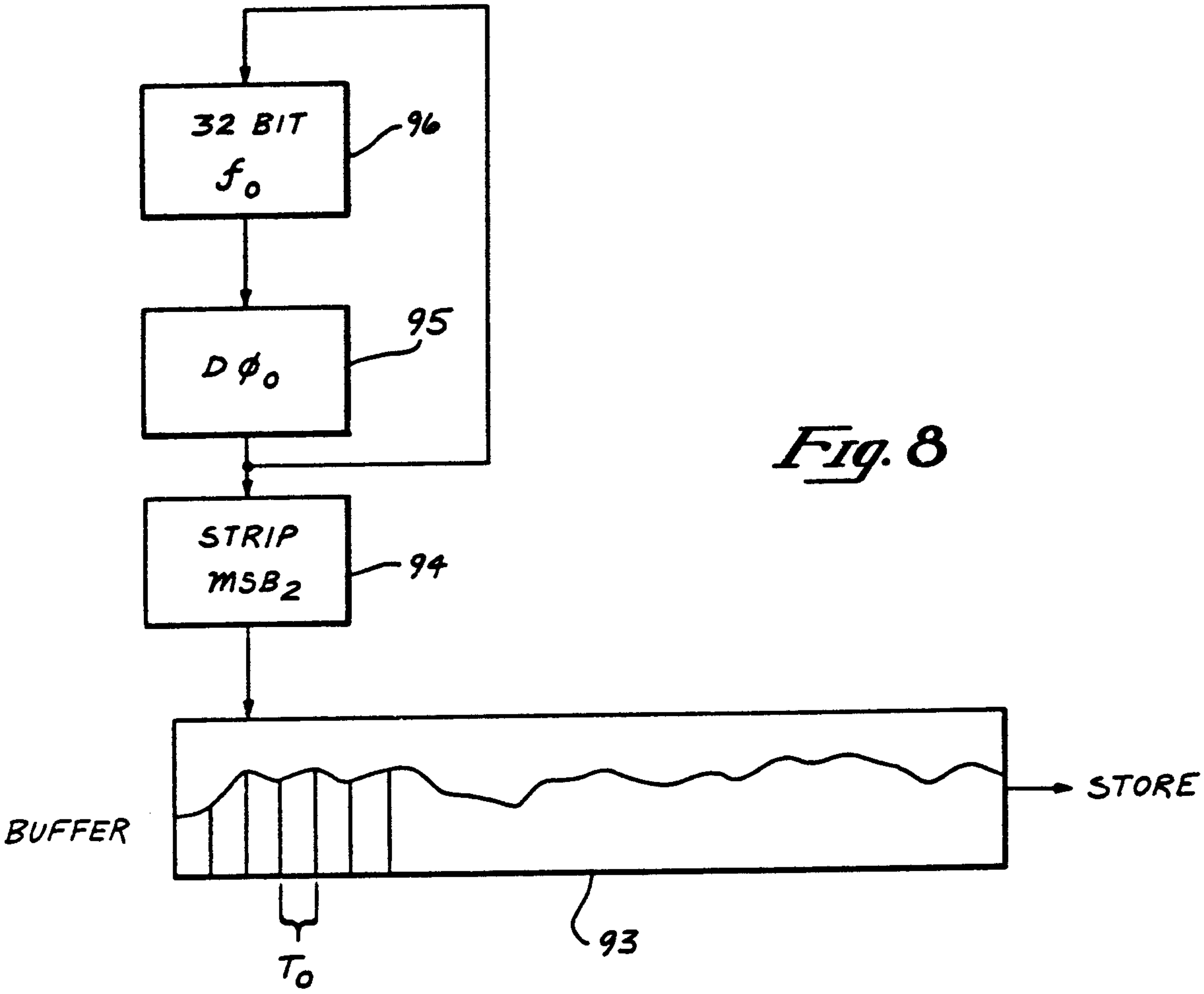

32 BIT
f_0
96
$D\phi_0$
95
STRIP
MSB_2
94
BUFFER
STORE
T_0
93

Fig. 8

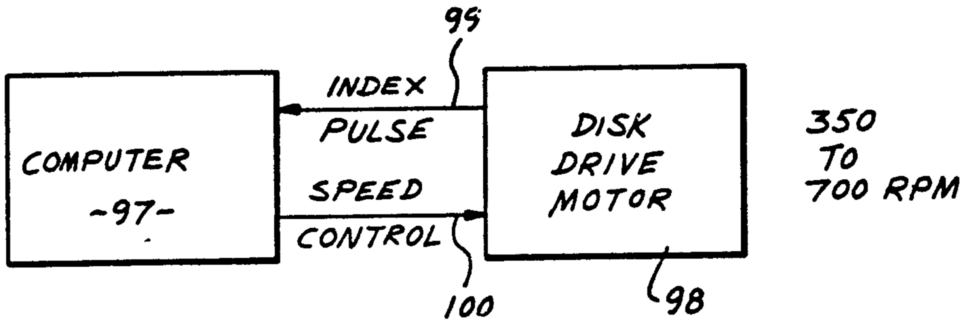

99
INDEX
PULSE
SPEED
CONTROL
COMPUTER
-97-
DISK
DRIVE
MOTOR
350
TO
700 RPM
100
98

Fig. 9

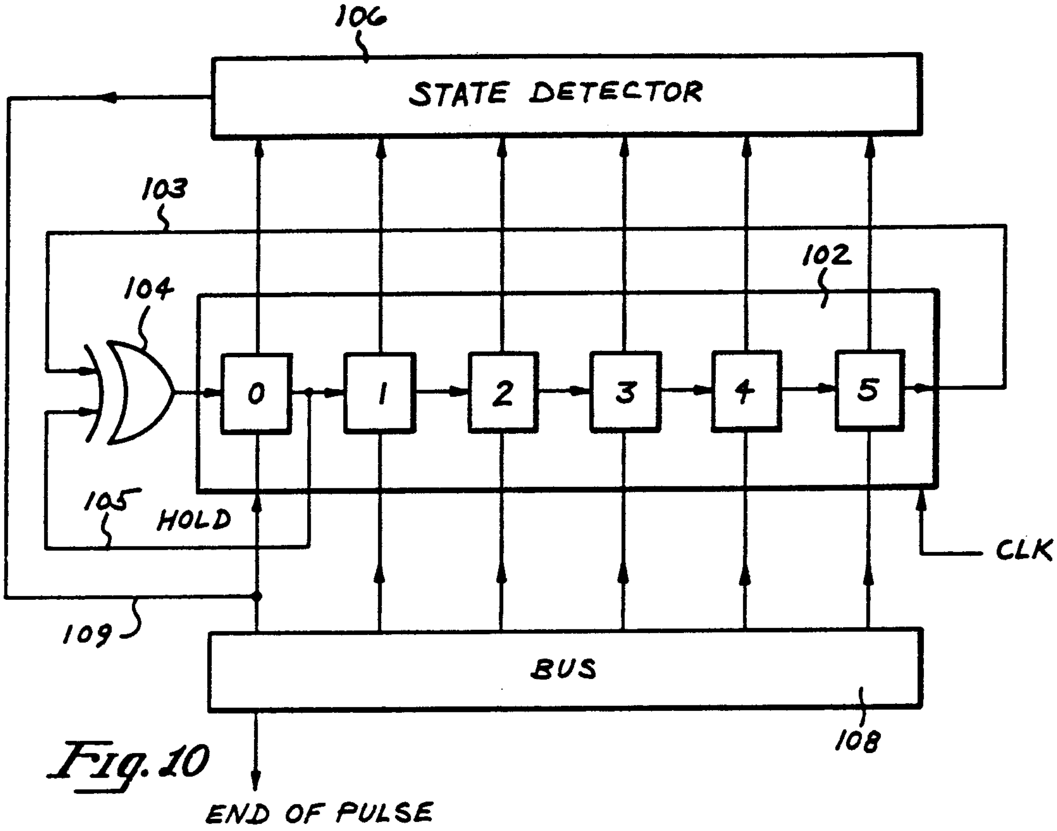


Fig. 10

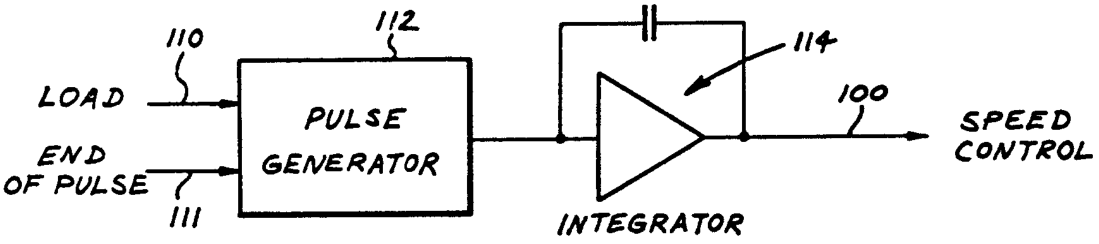


Fig. 11

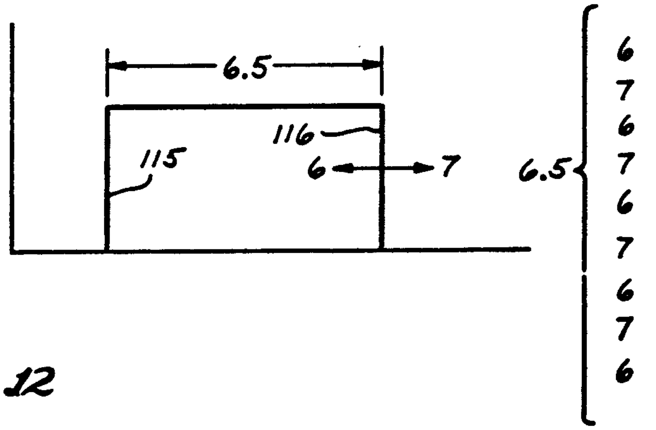


Fig. 12

SOUND GENERATION AND DISK SPEED CONTROL APPARATUS FOR USE WITH COMPUTER SYSTEMS

This is a continuation of application Ser. No. 573,132 filed Jan. 20, 1984 now abandoned.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention relates to a sound generation apparatus, particularly one employed with a computer system which includes a raster scanned display and a disk speed control apparatus.

2. Prior Art

There are countless well-known techniques for generating audio signals from digital signals. These include the more straightforward approaches where digital signals are used to provide an instantaneous amplitude of the audio signal, to the more complex vocoder techniques where transfer functions representative of voice are used. As will be seen, the present invention converts a digital signal to an analog (audio) signal, although this conversion is only one aspect of the present invention.

Most often, computer systems, particularly smaller systems (e.g., personal computers) employ raster scanned displays. The computer generates the video information and stores it in a random-access memory (RAM). Counters synchronize with the horizontal and vertical synchronization signals address the memory to provide display synchronized data signals from the memory. These signals are converted to a video signal, for instance, through a shift register. In some cases, the memory is "bit mapped" and the output from the memory is directly used to generate the video signal. In other cases, the output from the memory addresses a character generation which is scanned to provide video signals.

A considerable amount of data from RAM is required to generate a video display, particularly in a dynamic, graphics (non-text) mode. In the personal computer field, or small business computer field, where microprocessors are used along with dynamic RAMs, the generation of a video display consumes a relatively large amount of processor and memory time. It is thus difficult to provide an audio signal, particularly a complex audio signal in a display mode.

As will be seen, the present invention provides an apparatus for generating audio signals in conjunction with a microprocessor and RAM simultaneously with the generation of video signals. The audio signals are generated without disrupting the video display or computer operation, and importantly, with a minimum of hardware and processor time.

Typically, in floppy disk drives, some mechanism is employed to drive the floppy disk motor at a constant speed. When the floppy disk drive is manufactured, certain calibration steps are often used to assure that the floppy disk drive runs at a predetermined rate of rotation. This requires, in addition to the calibration steps, relatively costly speed control mechanisms. As will be seen, in the present invention, the computer is used to sense the rate of rotation of the disk drive and then provides a control signal to adjust the disk drive's rate of rotation. This eliminates the prior art calibration and also the prior art's speed control mechanism.

It has been suggested in the prior art that better utilization of floppy disks or other disks can be obtained if

uniform flux density transitions are used. This requires that the rate of rotation of the disk be made a function of the radius of the particular track being accessed. The present invention provides such a feature.

SUMMARY OF THE INVENTION

The present invention provides an apparatus for use with a computer system which includes a microprocessor and random-access memory (RAM), particularly where a raster scanned display is used with the computer system. Addressing means are used for directly accessing predetermined locations in the RAM, especially during the horizontal blanking period. The addressing means also permits data in these same locations to be updated during the blanking periods. The data stored in these locations is converted from its digital form to an analog signal. A pulse is initiated when the data from memory is loaded into a counter. The pulse is ended when the counter reaches an overflow. The resultant pulses are integrated to provide the audio signal.

The processor generates the data signals for the RAM for a single tone by adding a predetermined number to a stored number. The most significant bits of this sum identify a location in a look-up table and the resultant (digital) data signal is then stored in RAM. The predetermined number is repeatedly added to the stored number to provide each of the data signals for the RAM. For more complex tones, a number of predetermined numbers and stored numbers are used along with a plurality of look-up tables.

The present invention also provides an apparatus for controlling the rate of rotation of a disk. The addressing means used in conjunction with the sound generation apparatus are used as part of the disk control apparatus.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram of the computer system and illustrates the address multiplexing used in conjunction with the present invention.

FIG. 2 is a timing diagram used to describe times at which the digital signals representative of sound are accessed from the RAM and times at which they are updated in the RAM.

FIG. 3 is a block diagram of a counter used to generate the audio signals.

FIG. 4 illustrates waveforms generated from the counter of FIG. 3.

FIG. 5 is a flow diagram used to describe the method by which data signals are produced.

FIG. 6 is a block diagram and schematic of the circuit for providing the audio signal and volume control.

FIG. 7 is a flow diagram illustrating the method by which data signals are provided for four tones.

FIG. 8 is a flow diagram illustrating the method by which data signals are generated for a "non harmonic" audio signal.

FIG. 9 is a block diagram illustrating the general interconnection between the computer of FIG. 1 and a disk drive motor.

FIG. 10 is a block diagram illustrating part of the circuit used to generate the speed control signal for the disk drive.

FIG. 11 is a block diagram illustrating an additional portion of the circuit used for generating the speed control signal for the disk drive.

FIG. 12 is a graph illustrating development of the speed control signal.

DETAILED DESCRIPTION OF THE INVENTION

An apparatus for generating audio signals in conjunction with a computer system particularly one which generates signals for a raster scanned display and for generating a motor speed control signal is described. In the following description numerous specific details are set forth such as specific frequencies, number of lines, commercial part numbers, etc., to provide a thorough understanding of the present invention. However, it will be obvious to one skilled in the art that the present invention may be practiced without these specific details. In other instances, well-known circuits have been shown in block diagram form in order not to unnecessarily obscure the present invention.

DEFINITION

In the following description, the term "audio or sound data signal" or "sound data" is used to identify a digital signal which is converted to an analog (audio) signal. The term motor speed control refers to the control of rate of rotation of a motor or disk driven by the motor.

GENERAL ARCHITECTURE

The present invention is currently realized as part of a computer system (personal computer or small business computer) employing a Part No. 68000 microprocessor. The address lines and data lines for this microprocessor 10 are shown in FIG. 1. The other well-known lines coupled to this processor are not shown in FIG. 1. The microprocessor 10 is coupled to a random-access memory (RAM) 11 comprising sixteen 64K dynamic memory "chips". The data lines 0–15 interconnect the microprocessor 10 and RAM 11 to permit data to flow from the processor into the RAM. The data from the RAM is coupled through the RAM data buffer 13 into the processor; also data is coupled from the RAM 11 to disk motor speed controller 27, video shift register 28 and sound counters 29. The latter counters will be described in detail in conjunction with FIG. 3. Data is also received by the microprocessor 10 from the read-only memory (ROM) 17 when the ROM is enabled (ROMEN/). Similarly, data is transferred to and from the disk controller 18 when the disk controller 18 is enabled by a signal on line 35. This signal, as is the ROMEN/ signal is generated within the PALS 23. Data is likewise coupled to and from the microprocessor 10 to a serial communications controller 14 and an interface adapter 15 (Commercial Part Nos. 8530 and 6522, respectively).

Addresses from the microprocessor 10 are coupled to the ROM 17, PALS 23 and RAM address multiplexer 20. Some of the address signals, as indicated, are also coupled to the disk controller 18, serial communications controller 14 and interface adapter 15.

The RAM address multiplexer 20 permits the RAM to be addressed either by the microprocessor 10 or directly by the count stored in the video counter 22. During the time when the video signal is "painting" the screen, the multiplexer 20 selects the video counter 22, thus allowing the counter to directly address the RAM 11. (A signal from PALS 23 controls this selection.) During other times, the RAM address multiplexer 20 permits the microprocessor 10 to directly access the RAM 11. The second address multiplexer 21 as is multiplexer 20 is controlled by a signal from the PALS 23. During the last portion of the horizontal blanking signal, as will be described in conjunction with FIG. 2, multiplexer 21 selects the highest order 7 bits from the counter 22 and forces the memory to this address. This requires that the sound and disk speed data be stored in dedicated and consecutive locations of the RAM and permits easier access by the microprocessor when this data is updated. A latch, not shown, provides an additional bit input to the address lines of the multiplexer 21 to cause direct access to a second page of sound data in RAM 11.

The video counter 22 which consists of two Part Nos. 74LS393 provides a digital video count which corresponds to the beam's position on a raster scanned display and additional counts for the horizontal and vertical restore (blanking) periods. The timing signals which operate this counter along with the reset signals are generated by the PALS 23.

The PALS 23 consist of three program array logic chips. They receive the crystal controlled 16 mHz oscillator signal from oscillator 31. The PALS 23 generate from this signal the standard memory signals such as RAS/, CAS/, and the well-known timing signals used by the microprocessor. They also provide the horizontal synchronization signal (HSYNC/) and the vertical synchronization signal (VSYNC/). These signals are coupled to the display on lines 32. Other clocking signals used throughout the memory, such as the 8 MHz clocking signal used by the counters of FIG. 3 and the clocking signal used by the disk motor speed controller are generated within the PALS 23.

Two 32K×8 ROMS 17 are employed in the presently preferred embodiment. They provide storage for diagnostics, initialization and other functions not relevant to the present invention.

The disk controller 18 provides an interface to a floppy disk drive. The controller is described in more detail in copending application Ser. No. 573,067, filed Jan. 24, 1984, entitled Integrated Floppy Disk Drive Controller, and assigned to the assignee of the present invention.

The adapter communicates with the keyboard 24. A mouse 25 provides cursor input and switching information to both the controller 14 and adapter 15. A volume control knob is drawn on the graphics screen and is controlled by the mouse to provide three bits of binary data on lines 37. As will be described in conjunction with FIG. 6, these three bits are used for a static volume control for the audio signal.

VIDEO TIMING

In the presently preferred embodiment, horizontal scanning occurs at a rate of 22,256.84398 Hz. Vertical scanning occurs at a rate of 60 Hz. Each frame consists of 370 scan lines and there are 704 pixels, or dots, per horizontal scan. This corresponds to 44 sixteen bit words from the RAM 11. Thus, the main clock rate from oscillator 31 shown as 16 mHz is more accurately 15.6672 mHz.

Referring to FIG. 2, on the display itself there are 512 "live" pixels in the horizontal direction and 342 lines on the screen. The 192 remaining bits during each horizontal scan is the horizontal blanking period sometimes referred to as the "flyback" time. It is during this period of time that the beam current in a cathode ray tube is lowered and the beam brought back from one side of the screen to the other. In the vertical direction, in addition to 342 lines on the display, there are 28 additional periods during which time the vertical blanking occurs, that is, the beam current is again reduced and the beam returned from the lower part of the screen to the upper part of the screen.

In FIG. 2, time is shown from left to right by, for instance, the dotted line 39. On the first scan, after 512 bits have been displayed, the time represented by line 40 is reached, and blanking occurs. During blanking, it is not necessary for the RAM 11 to furnish data for the display. Prior to the time 40, referring to FIG. 1, the count from counter 22 accesses the RAM 11 through the RAM address multiplexer 20. This occurs for each of the lines in the display. (The counter 22 maintains both a horizontal and vertical count.) The counters do not increment in the normal sense during the horizontal blanking period. Rather, four bits of the video counter are reused for counting during this period. This eliminates address gaps for the sound data. When time 40 is reached for each of the scan lines, a timing signal from the PALS 23 causes the multiplexer 20 to accept addresses from the microprocessor 10. During the next 192 counts of the 16 mHz clock, except for the last count, the microprocessor is free to access the RAM and thus can perform tasks unrelated to the display. When the last count in each of the scan lines is reached, a signal from the PALS 23 causes the counter 22 through multiplexer 21 to directly access the memory 11. At this time, the sixteen bit word from RAM 11 (time 41 of FIG. 2) is read from the memory with 8 bits going to the disk motor speed controller 27 and 8 bits to the sound counters 29 (as will be seen, only six bits are used by the disk motor speed controller 27.) During the "screen time" shown in FIG. 2, the sixteen bit words from the memory are placed in the video shift register 28 and used to provide the video signal. The PALS 23, as mentioned, on line 32 provide the horizontal and vertical synchronization signal used in conjunction with the signal from the shift register 28 to control the video display.

When the 342nd scan line is reached (shown as line 43) and at time 40 along this line, the multiplexer 20 again allows the microprocessor 10 to access the RAM 11. However, at the end of line 43 and for the remaining period of the vertical blanking, the multiplexer 21 still forces 9 bits of address into the RAM 11 at time 41 to allow the 16 bit word to be supplied to the speed controller 27 and counters 29. (The lines RA0 to RA6 are time multiplexed to provide these address signals.) During the vertical blanking, the microprocessor 10 is able to access the RAM 11, except for the last count of each line. It is during this period of time as will be described that the disk motor speed control data and sound data stored in the RAM 11 is updated.

The multiplexer 21 with its nine bit address defines contiguous locations in memory, thus allowing all the sound and motor speed data to be more easily accessed and updated by the processor 10. Note that the storage location in the RAM 11 for the sound and speed control data will be in a different location than the screen data.

As currently implemented, during "live" video the microprocessor and video display signal transfers time share the data bus in alternating cycles. During horizontal blanking (for words 32 to 42) the microprocessor alone has access to the data bus. At time 41 of FIG. 2 (43rd word) the microprocessor and sound/speed data transfer time share the data bus in alternate cycles.

It is possible for the microprocessor to update the sound data and speed control data during the live video. The data is, in fact, updated during blanking periods. As currently implemented and preferred, the vertical synchronization signal (retrace signal) initiates the sound data updating. By using this signal and by updating the locations already accessed (e.g., beginning at the location used at line 39, time 41) updating does not interfere with the reading of the sound data. The software program assures that the updating remains ahead of the reading of the sound data. If the microprocessor updates the sound data without being synchronized with the display, data could be replaced before being used. Also this arrangement frees the software from the requirement of being time synchronized with the sound for updating the data.

AUDIO SIGNAL GENERATION

The eight sound data bits representing the audio signal are shifted in parallel into two four bit counters 46 and 47, shown in FIG. 3. These are commercial counters (Part No. 161). The counters are clocked by the 8 mHz clocking signal on line 48. Counting continues in these counters until overflow which is sensed on line 49. Thus, if all zeroes are placed in the counters, a longer period of time is required until overflow (approximately 32 μsec.) whereas overflow can occur as soon as one cycle of the 8 mHz clock if all ones are loaded into the counters.

The audio waveform is developed by first generating pulses the widths of which are a function of the time between the loading of the eight bits into the counters 46 and 47 and overflow. For instance, as shown by FIG. 4, the leading edge 52 of a pulse occurs upon loading of sound data into the counters. If all zeroes are loaded, then approximately 32 μsec. later, overflow occurs and the pulse ends as indicated by the trailing edge 54. One pulse is generated during each horizontal sweep since one eight bit sound data word is loaded into the counters during each sweep. Therefore, pulses are generated at a frequency of approximately 22,000 Hz, and in theory, this provides a bandwidth of approximately 11,000 Hz. In FIG. 4, a second pulse 56 is shown which has a substantially reduced width. This, of course, would occur when a larger number is placed into the counters 46 and 47. The pulse 57, which is shown occurring during a third sweep, has a width which falls between the first and second pulses.

The pulses are integrated using an ordinary integrator to provide the analog signal. The integrator 60 of FIG. 6 receives a load signal and the overflow signal; the waveform 61 shown in FIG. 4 is developed within the integrator 60. Waveform 61 represents the resultant integration of the pulses shown in FIG. 4.

The three bits of information (bits 37*a*, 37*b* and 37*c*) from the interface adapter 55 are used to allow a user to statically control volume. The amplifiers 63, 64 and 65 are switched (on or off) to permit the output amplitude on line 68 to be controlled.

CALCULATION OF THE SOUND DATA SIGNALS

Sound data from the memory which define the sound waveforms are calculated by the microprocessor 10. More specifically, they are "software" generated within the microprocessor. A higher order language, such as PASCAL, may be used to allow a user to more easily implement the flow diagrams which are discussed below. In general, the sound data are produced quite rapidly since the process takes advantage of the rapid adding capability of the 68000 microprocessor.

Referring to FIG. 5, assume that a single "pure" tone is to be generated. First, a look-up table is stored within the system memory; in the presently preferred embodiment the look-up table is 256×8 bits. Thus, for each eight bit address to the table an eight bit output results. For a pure tone, the look-up table contains points corresponding to a sinewave. This is illustrated by the look-up table 70 of FIG. 5. The process of generating the address for the subsequent value table is the repeated adding of some predetermined number shown in block 74 as ϕ_0 to a number stored in register 27. Initially, the 32 bit word stored in register 72 may have any value, for instance, all zeroes. The increment, ϕ_0 is added to it. The resultant sum is restored in register 72. The most significant eight bits are stripped from the sum as shown by block 76 and used as an address for the look-up table 70.

Assume for sake of discussion that $\Delta\phi_0$ is small. Each time this relatively small binary number is added to the number stored in register 72, the most significant bits will not change, but rather, numerous additions are needed for them to change. Consequently, each of the 256 locations in the look-up table 70 will be addressed several times and the eight bits of data from the look-up table which are stored within the RAM 11 will vary slowly. This, of course, will correspond to a low frequency. If, on the other hand, the increment $\Delta\phi_0$ is relatively large, the results from the look-up table will change more rapidly and thus, for instance, each of the consecutive eight bit data words from the look-up table 70 which are stored in the RAM 11 will be different. This would correspond to a high frequency. A new eight bit sound data word is obtained with each addition represented by block 74. Therefore, by varying the increment added on each cycle, the frequency of the tone is varied. All the sound data used during each frame can easily be calculated during a few scan line periods of the vertical blanking period.

To obtain envelope control or amplitude modulation, a set of tables may be used. Each table, for instance, of set 0–7, contains a sinewave with maximum peak to peak value of $2^{SET.NO.}$ By allowing a predetermined number of frame intervals to pass before switching between sets, envelope control is achieved.

Referring to FIG. 7, in the presently preferred embodiment, up to four 256×8 look-up tables may be used within the microprocessor 10. And, the contents of each look-up table can be user programmed and each may be different. For instance, look-up table 80 of FIG. 7 is shown as containing a sinewave, table 81 as a triangular wave, table 82 as a square wave, and table 83 as a ramp. The process described in conjunction with FIG. 5 is again used. However, this time (with four simultaneous tones being generated) 24 bits, rather than 32, are used. (This is shown by block 85 in FIG. 7.) Again, an increment shown as $\Delta\phi_1$ is added to the previous sum (block 86). The most significant bits are stripped from the sum (block 87) and used as an address for the corresponding eight bit word within the table 80. The same process is repeated for the number shown within block 87 where a different (or the same) increment $\Delta\phi_2$ is added shown at block 88, and again the most significant bits of the same are used to address look-up table 81. Similarly, different stored values and increments are generated to allow look-ups tables 82 and 83. The resultant eight bits from each of the tables are added as shown by blocks 80, 90 and 91 and the most significant eight bits are stripped from this sum as shown by block 92 and stored

within the memory 11. This process is repeated for each of the sound data words stored within the RAM 11 when four tones are generated. Once again, the fundamental frequency for each of the four tones is determined by the increment which is added, such as it blocks 86 and 88, and the harmonic content is determined by the "shape" stored within the look-up table.

Table I, attached, is a program written in 68000 assembly language for implementing the flow diagram of FIG. 7.

With the above-described sound generation apparatus, excellent tone control is achieved with up to 24 bits of "frequency control" being possible (for each tone) within the 11 khz band. This permits almost 17 million different tones to be generated within the band which is approximately equal to (or better) than the best discernability of the human ear.

The above-described processes are particularly suited for providing periodic functions which are harmonic in nature and provide a tonal quality representing music, and the like. For sounds such as voice, an "extended" look-up buffer may be used for initially storing a waveform representative of, for example, speech. This is shown as buffer 93 in FIG. 8. The buffer in fact can be within the RAM 11 and for practical reasons must be if a long waveform is to be stored. The eight bit values are again obtained by adding some increment $\Delta\phi_0$ shown in block 95 to a 32 bit word stored in register 96 with the most significant bits being used to address locations in the buffer 93. The results for the look-up in the extended buffer are stored and selected during the horizontal blanking period as was the case with the case of FIGS. 5 and 7.

Table 2, attached, contains a program written in 68000 assembly language for implementing the flow diagram of FIG. 8.

DISK MOTOR SPEED CONTROLLER

Most typically, floppy disk drives and other disk drives, include a mechanism for driving the disk at a constant, predetermined rate of rotation (speed). Upon fabrication of the disk drive, the speed control mechanism is calibrated to assure that data will be recorded and retrieved at a certain rate.

For the present invention the motor speed is controlled by a computer, and moreover, the motor speed is varied as a function of the track being accessed so that uniform flux densities result. That is, the motor turns slower when the outer tracks (greater radius) are being used and faster when the inner tracks (smaller radius) are being used.

In FIG. 9, the computer of FIG. 1 is shown as computer 97. A disk drive such as a floppy disk drive and in particular, a disk drive motor 98, is also illustrated. Line 99 provides the computer 97 with pulses which indicate the motor speed. In the presently preferred embodiment, the standard indexing pulses from the motor are used. The floppy disk drives employed are keyed to the motor hub, and thus no slippage occurs. Consequently, the index pulses themselves represent the actual rate of rotation of the floppy disk. If slippage is possible, then markers or bit streams from the disk itself may be used to obtain an accurate indication of the disk speed. The speed control signal on line 100 controls the motor speed. A predetermined signal level is used on line 100 and the motor speed sensed on line 99. This allows the computer 97 to record the characteristics of the motor 98. That is, the computer knows for each motor connected to it, the rate of rotation of the motor for a particular speed control signal. In this manner, the disk drive motor 98 itself need not be calibrated when being manufactured, and moreover, the speed control mechanism normally used within the disk drive is not needed since the speed control occurs from the computer 98. As is apparent from FIG. 9, closed loop operation occurs since the computer 97 senses the actual motor speed on line 99.

As currently implemented, the computer 97 examines the pulses 99 and, in effect, determines the characteristics of the motor 98 when a new disk is placed within the disk drive, before data is written or if errors occurred on reading or writing. Obviously, other arrangements may be used, for instance, the indexing pulses can be checked periodically, or for that matter, continually.

In the presently preferred embodiment, the motor operates at a speed from 700 rpms for the innermost track, to 350 rpms for the outer track. Obviously, the selected range of rate of rotation will be a function of the radius of the disk and will vary, depending upon the particular magnetic characteristics of the system and the size of the disk.

As previously mentioned, during each horizontal blanking period, 8 bits of data are provided to the sound counters 29 of FIG. 1, and 8 bits are provided to the speed controller 27. In the presently implementation only six of the bits on this bus are used for speed control. The bus is illustrated as bus 109 in FIG. 10 and these six lines from the bus are shown coupled to six stages of a shift register 102. The six bits from the bus 108 are loaded into the six stages of the register 102 when the sound data signals are loaded into the sound counters 2.

FIG. 10 implements a polynomial counter. The data placed into the six stages of the shift register 102 are shifted under the control of a clocking signal. The effective shift rate is approximately 1 mHz. Because of the various waiting stages involved in the shift register, the 8 mHz clocking signal is actually coupled to the register. The output of the last stage of the register is coupled to one input terminal of an exclusive OR gate 104 through line 103. The output of the first stage is coupled to the other input terminal of the gate 104 through line 105. This arrangement provides for counting in the "polynomial generator" in a manner known in the prior art. The stages of the shift register 102 are also coupled to a state detector 106. This detector determines when a predetermined binary state is reached within the shift register. When this state is reached, a signal is coupled over line 109 to stop the shifting within the shift register 102; this signal is used to generate the end of a pulse in the same manner as used for the sound signal.

Referring to FIG. 11, counting begins within the shift register 102 of FIG. 10 at the beginning of each horizontal sweep. At this time, the leading edge of a pulse is generated such as edge 115 of the pulse shown in FIG. 12. When the state detector 106 detects the predetermined state, the end of the pulse is generated such as shown by trailing edge 116 of FIG. 12. The pulses are integrated by the integrator 114 and the resultant signal on line 100 is used to control the speed of the motor in an ordinary manne:.

The 6 bits placed within the shift register 102 will always reach the state detected by the detector 106 before the end of each horizontal sweep. In practice, the state will be detected during the first 40 μsec. of the approximately 44 μsec. required for each horizontal sweep.

Ten horizontal sweeps are used for each speed control setting. This is chosen since the presently preferred embodiment employs 370 total scan lines which is evenly distributed by 10. Nonetheless, a pulse is generated for each horizontal sweep. (The time constant associated with the integrator 114 of FIG. 11 is slow enough that a continuous signal results on line 100.) The pulse width generated for each of the 10 sweeps used to define each speed control value is "dithered" to provide precise values. For instance, assume that a value corresponding to 6.5 is required on line 100. Referring to FIG. 12, for the 10 sweeps used to define this value, the first would have the value 6, the second the value 7, and so on for the 10 sweeps. This would cause the trailing edge 116 of the pulses to vary between the values 6 and 7. After being integrated, however, the value on line 100 would correspond to 6.5. By distributing the values and permitting the pulse dithering during the 10 sweeps used to define each speed control number, very accurate control occurs. Control accuracy beyond the 6 bits loaded into the shift register is obtained. In the present realization 400 unique levels or

$$\frac{\log(400)}{\log(2)}$$

bits are achieved.

Attached as Table 3, is the program used for the speed control, written in 68000 assembly language.

Thus, an improved apparatus has been described that permits both sound generation and motor speed control in a floppy disk drive, or the like.

TABLE I

```
     ;
     ; This code is executed every 16 sec at the vertical retrace
     ; interrupt. It computes the 370 values for the next sweep.
               MOVEM.L    (A6),D2-D7/A0-A5      ;get sound params into registers
               MOVE.L     SoundBase, A6         ;point to the buffer
               ADD.W      #370,A6               ;actually, point halfway into it
               MOVE.L     #$00FF0000,D1         ;set up mask in high part of D1
               MOVE       #2, -(SP)             ;init outer loop counter
               MOVE       #185,-(SP)            ;loop 185 times (half the buffer)
loop once for half of the 370 values, summing the waveform values for each voice
SoundLoop
          CLR.W      D1                    ;clear out summing register (not the mask
          ADD.L      D2,D3                 ;compute voice 1
          ADD.L      D4,D5                 ;compute voice 2
          ADD.L      D6,D7                 :compute voice 3
          ADD.L      A0,A1                 ;compute voice 4
;map voice 1 into D1
          MOVE.L     D5,D0
          AND.L      D1,D0                 ;mask off high bits
          SWAP       D0                    ;use bits 16-23
```

TABLE I-continued

```
        MOVE.B    0(A3,D0),D0       ;lookup in waveform tables
        ADD.W     D0,D1             ;add it in
;add voice 3 into D;
        MOVE.L    D7,D0
        AND.L     D1,D0             ;mask off high bits
        SWAP      D0                ;use bits 16-?3
        MOVE.B    0(A4,D0),D0       ;lookup in waveform table
        ADD.W     D0,D1             ;add it in
;add voice 4 into D1
        MOVE.L    A1,D0
        AND.L     D1,D0             ;mask off high bits
        SWAP      D0                ;use bits 16-23
        MOVE.B    0(A5,D0),D0       ;lookup in waveform table
        ADD.W     D0,Dl             ;add it in
;update the DMA sound buffer with the new value
        LSR.W     #2,D1             divide by 4(use most significant bits)
        MOVE.B    D1,(A6)           ;put it in the buffer
        ADDQ      #2, A6            ;bump, buffer pointer
;loop for half the values
        SUBQ      #1,(SP)           ;decrement counter
        BNE.S     SoundLoop         ;loop till done
;now do the second half of the buffer
        MOVE.L    SoundBase, A6     ;point to start of buffer
        MOVE      #185,(SP)         ;reset the counter
        SUBQ      #1,2(Sp)          ;decrement second counter
        BNE.S     SoundLoop         ;loop till done
;OK, all done, Update sound table, restore registers and return to caller
        ADDQ      #4,SP             ;pop off loop counter
        MOVE.L    SoundPtr,A6       ;get table address
        ADDQ      #2, A6
        MOVEM.L   D2-D7/A0-A1,(A6)  ;save back the sound registers
        MOVEM.L   (SP) + ,D0-D7/A0-A6   ;restore caller's registers
```

TABLE 2

```
    MOVE.L    SoundBase, A2   ;get sound base address
    ADD.W     #64,A2          ;start 32 bytes in
    LEA       676(A2),A4      ;compute the end address
    CLR.W     -(SP)           ;flag pass 1
    MOVE      #337,D2         ;338 bytes to move in 1st half
;OK, now that we have everything set up, start the main loop to fill the
buffer
    MOVE.B    (A1),(A2)       ;move it into the DMA buffer
    ADDQ      #2,A2           ;bump to next location
```

TABLE 2-continued

```
    ADD.L     D1,D3             ;bump cumulative index
    SWAP      D3                ;get high part in low territory
    ADD.W     D3,A1             ;bump to next entry (maybe)
    ADD.W     D3,D0             ;accumulative numDone
    CLR.W     D3                ;reset integer part
    SWAP      D3                ;restore D3
;have we exhausted our request?
    CMP.L     A1,A3             ;past the end of the buffer?
    DBLE      D2,Interpolate    ;if so, stop it
```

TABLE 3

```
;
;Routine:      SetSpeed, SetASpeed
;Arguments:    D6.W (input) - track number speed sbould be set for
;              Drive   (input) - current disk drive
;              TrkSpeedTbl (in) - speed code table for current drive
;              Wait       (output) - 0, or SpdChgTime if CurSpeed changed
;              registers other than A0-A2, D0-D2 are preserved
;Called By:    (SetSpeed): Seek,RWPower
;              (SetASpeed): MakeSpdTbl
;Function:     This routine determines the correct speed value
;              for Track and sets up the PWM memory buffer to
;              produce the desired output. The value of Wait is
;              set to SpdChgTime if the speed is changed. 0
;              otherwise. The TrkSpeedTbl for the current drive
;              is used. The drive enable is not changed, just the
;              PWM buffer in memory.
;              SetASpeed is an alternate entry point which simply
;              sets the pwm buffer according to a speed code in D2.
Set Speed
        BSR.S      GetDrvl            ; set up D1.A1
        MOVE.W     D6,D2              ; speed class is just track number
        LSR.W      #4,D2              ; divided by 16 . . .
        LSL.W      #3,D2              ; adjust to double-longword word index
        ADD.W      D1,D2              ; add drive specific offset
        MOVE.W     TrkSpeedTbl
                   (A1,D2),D2         ; get the speed we need
        ADD.W      OffSpeed(A1,D1),
                   2                  ; add in an adjustment (watch max,min
        BSL.S      @2                 ; don't go below 0
        MOVEQ      #0,D2
@2      CMP.W      #399,D2
        BLE.S      @3
        MOVE.W     #399,D2            ; don't go above 399
```

“APPLE_PAT_4_910_670_13” 131 KB 2000-02-21 dpi: 300h x 300v pix: 1925h x 2964v

TABLE 3-continued

```
        MOVE.W    PWMValue,D0          ; are we at that speed?
@3      BPL.S     @4                   ; if speed is invalid, wait power-on
time
        MOVE.W    PwrOnTime(A1),D0
        BRA.S     @6
@4      SUB.W     D2,D0
        BEQ.S     GetDrvl              ; if so, just exit
        BPL.S     @5
        NEG.W     D0                   ; positive speed difference
@5      LSL.W     #5,D0                ; multiply by 32 to get speed settle
time
        CMP.W     SpdChgTime(A1),D0    ; minimum wait time for speed change
        BGT.S     @6
        MOVE.W    SpdChgTime(A1),D)
@6      ADD.W     Wait(A1),D0          ; add in current wait time
        CMP.W     PwrOnTime(A1),D0
        BLT.S     @7
        MOVE.W    PwrOnTime(A1),D0
@7      MOVE.W    D0,Wait(A1)
;SetASpeed is an alternate entry point which simply sets up the speed code
in D2
SetASpeed
        MOVE.W    D2,PWMValue          ; note the speed for future reference
        MOVEM.L   D3-D6/A2,-(SP)       ; preserve A2-A7 D3-D7
        SUB.W     #399,D2              ; invert it (for sony)
        NEG.W     D2
        EXT.L     D2                   ; make it a long . . .
        DIVU      #10,D2               ; remainder in high word
        MOVEQ     #11, D0
@1      MOVE.B    D0,D1                ; main speed value
        MOVE.B    D0,D3                ; save bit 0
        LSR.B     #1, D0               ;
        EOR.B     D0,D3
        LSR.B     #1, D3               ; new bit 5 ->cy
        BCC.S     @2
        BSET      #5, D0
        DBRA      D2,@1
        SWAP      D2                   : remainder determines dither
        MOVE.B    DitherTbl(D2),D5     ; need 10 bits from dither table
        ASL       #8.D5
        MOVE.B    DitherTbl + 1(D2),D5 ; get 2 bits from next one
LoadPWMBuf
        MOVEQ     #36.D3               ; big loop goes 37 times
        LEA       PWMBuffer,A0         ; fill up PWM buffer
                                       (37 × 10 = 370 bytes)
        MOVE.L    PWMBuf2,A2           ; in case of alternate buffer
@1      MOVE.Q    #9,D2                ; inner loop goes 10 times
        MOVE.W    D5,D4                ; dither pattern
@2      LSL.W     #1,D4                ; carry bit = 1 means use higher value
        BCC.S     @3
        MOVE.B    D0,D6                ; use higher value
        BRA.S     @4
@3      MOVE.B    D1,D6                ; use main value
@4      MOVE.B    D6,(A2)
        ADDQ      #2,A0                ; every other byte is sound stuff
        ADDQ      #2,A2
        DBRA      D2,@2
        DBRA      D3,@1
        MOVEM.L   (SP) + ,D3-D6/A2     ; observe reg save conventions
SetSpdExit  BRA   GetDrv 1
DitherTbl                              ; used to dither the speed values evenly
   .Byte      $00,$20,$21,$24,$94
   .Byte      $AA,$B5,$B7,$7B,$FF,$40,$00
```

We claim:

1. In a computer system which includes a microprocessor and a random-access memory (RAM) and which provides a video signal for a raster scanned display, wherein said microprocessor accesses said RAM for loading data and said data is read from said RAM for presentation to said raster scanned display, an apparatus for generating an analog audio signal comprising:

a first counter for providing a digital count representative of timing of said video signal for said display, said first counter providing a vertical line count and a horizontal bit count for each frame of said display;

first address multiplexing means coupled to said first counter, microprocessor and said RAM for coupling either an address signal from said microprocessor or said digital count from said first counter to access a location of said RAM;

second address multiplexing means coupled to said first counter and said RAM, for coupling a portion of said digital count from said first counter to access said RAM by direct memory access;

said portion of said digital count coupled by said second address multiplexing means accesses audio data stored in said RAM, at least during a portion of the horizontal blanking period of said video signal, said audio data being programmed by said microprocessor and stored in said RAM;

said first address multiplexing means for coupling said digital count from said first counter as an address to said RAM during a video display cycle to access video data stored in said RAM and for coupling address signals from said microprocessor to said RAM at least during portions of the vertical blanking period to update said audio data in said RAM;

waveform means coupled to receive said stored audio data from said RAM during said portion of said horizontal blanking period and for converting said audio data to said analog audio signal; said waveforms means further having a second counter into which said audio data is loaded from said RAM, said second counter counting at a predetermined rate after said audio data is loaded; and said waveform means further including pulse generation means coupled to said second counter for initiating a pulse when said second counter is loaded and for ending said pulse when said second counter reaches a predetermined count, such that a frequency of said audio signal is determinea by a programmed value of said audio data.

2. The apparatus defined by claim 1 wherein ending of said pulse occurs when said second counter overflows.

3. The apparatus defined by claim 2 including integration means for integrating said pulses from said pulse generation means.

4. The apparatus defined by claim 1 or 3 wherein said horizontal blanking means occurs at a frequency of approximately 22,000 Hz.

5. The apparatus defined by claim 4 wherein said vertical blanking oceurs at a frequency of approximately 60 Hz.

6. The apparatus defined by claim 1 including an additional waveform means for converting data from said RAM, addressed during said horizontal blanking period by said first counter, to a speed control signal for a disk drive.

7. In a computer system which includes a microprocessor and a random-access memory (RAM) and which provides a video signal for a raster scanned display, a method for generating an analog audio signal from digital data signals which are stored in said RAM by said microprocessor, comprising the steps of:

generating digital data signals under control of said microprocessor by storing a base number, adding a predetermined number to said base number, using the most significant bits as a location in a lookup table, storing the sum as said base number, and using a value stored at said location accessed by the most significant bits as output for storage in said RAM;

storing said digital data signals in said RAM;

generating address signals from a first counter;

accessing said RAM by using said address signals from said first counter during a video display cycle to access video data stored in said RAM;

accessing said RAM by using said address signals from said first counter during portions of horizontal blanking periods by direct memory access to obtain said stored digital data signals representative of said audio signal;

converting said digital data signals into said analog audio signal by generating a pulse which pulsewidth is dependent on the value of said digital data signal and integrating said pulse to generate waveforms for said analog audio signal;

loading said RAM with new digital data signals representative of new audio signals during vertical blanking periods, wherein said microprocessor accesses said RAM for loading of said new digital data signals.

8. In a computer system which includes a microprocessor and a random-access memory (RAM) and which provides a video signal for a raster scanned display, wherein said microprocessor accesses said RAM for loading data and said data is read from said RAM for presentation to said raster scanned display, an apparatus for generating a speed control signal for a disk drive comprising:

a first counter for providing a digital count representative of timing of said video signal for said display, said first counter providing a vertical line count and a horizontal bit count for each frame of said display;

first address multiplexing means coupled to said microprocessor, RAM and said first counter for coupling either an address signal from said microprocessor or said digital count from said first counter to access a location of said RAM;

second address multiplexing means coupled to said first counter and said RAM, for coupling a portion of said digital count from said first counter to access said RAM by direct memory access;

said portion of said digital count coupled by said second address multiplexing means accesses disk speed data stored in said RAM, at least during a portion of the horizontal blanking period of said video signal;

said first address multiplexing means for coupling said digital count from said first counter as an address to said RAM during a video display cycle to access video data stored in said RAM and for coupling address signals from said microprocessor to said RAM at least during portions of the vertical blanking period to update said disk speed data in said RAM;

waveform means for receiving said stored disk speed data from said !ocations and for converting said disk speed data to said speed control signal; said waveform means having a second counter into which said disk speed data is loaded from said locations of said RAM, said second counter counting at a predetermined rate after said data is loaded; and said waveform means further including pulse generation means coupled to said second counter for initiating a pulse when said second counter begins counting and for ending said pulse when said second counter reaches a predetermined count, said pulse generation means coupled to said second counter.

9. The apparatus defined by claim 8 including integration means for integrating said pulses from said pulse generation means.

10. The apparatus defined by claim 8 wherein said computer system senses disk drive speed and varies said control signal as a function of said speed to provide dynamic calibration.

11. The apparatus defined by claims 8 or 10 wherein said speed control signal is varied as a function of the track being accessed on a disk.

* * * * *

United States Patent [19]

Ashkin et al.

[11] **Patent Number: 4,912,627**

[45] **Date of Patent: Mar. 27, 1990**

[54] **METHOD FOR STORING A SECOND NUMBER AS A COMMAND ADDRESS OF A FIRST PERIPHERAL DEVICE AND A THIRD NUMBER AS A COMMAND ADDRESS OF A SECOND PERIPHERAL DEVICE**

[75] Inventors: **Peter B. Ashkin**, Gatos; **Michael Clark**, Glendale, both of Calif.

[73] Assignee: **Apple Computer, Inc.**, Cupertino, Calif.

[21] Appl. No.: **132,123**

[22] Filed: **Dec. 14, 1987**

Related U.S. Application Data

[62] Division of Ser. No. 765,396, Aug. 14, 1985.

[51] **Int. Cl.**^{4} **G06F 13/42**

[52] **U.S. Cl.** **364/200;** 364/229.2; 364/240.8; 364/261; 364/284.3; 340/825.52

[58] **Field of Search** ... 364/200 MS File, 900 MS File, 364/514; 340/825.08, 825.07, 825.50, 825.52, 825.22; 370/85

[56] **References Cited**

U.S. PATENT DOCUMENTS

3,221,307	11/1965	Manning et al.	364/200
3,646,534	2/1972	Miller	360/40
3,715,725	2/1973	Kievit et al.	364/900
3,787,627	1/1974	Abramson et al.	370/67
3,836,888	9/1974	Boenke et al.	364/200
3,863,025	1/1975	Gonsewski et al.	375/55
3,979,723	9/1976	Hughes et al.	370/88 X
4,063,220	12/1977	Metcalfe et al.	340/825.5
4,071,908	1/1978	Brophy et al.	364/900
4,345,250	8/1982	Jacobsthal	340/825.5
4,360,870	11/1982	McVey	364/200
4,373,181	2/1983	Chisholm et al.	364/200
4,442,502	4/1984	Friend et al.	364/900
4,498,169	2/1985	Rozumus	370/85
4,562,535	12/1985	Vincent et al.	364/200
4,568,930	2/1986	Livingston et al.	340/825.5

(List continued on next page.)

FOREIGN PATENT DOCUMENTS

0051425	5/1982	European Pat. Off. .
0104545	4/1984	European Pat. Off. .
59-52331	3/1984	Japan .
1508854	4/1978	United Kingdom .
1518565	7/1978	United Kingdom .
2035636	6/1980	United Kingdom .
2070826	5/1984	United Kingdom .
0143160	6/1985	United Kingdom .
2167274	5/1986	United Kingdom .
0207313	1/1987	United Kingdom .

OTHER PUBLICATIONS

Hill et al., "Dynamic Device Address Assignment Mechanism", IBM TDB vol. 23, No. 8, Jan. 1981, pp. 3564–3565.

Search Report, dated May 21, 1986, for British Patent Application No. 8607632.

Primary Examiner—Thomas C. Lee

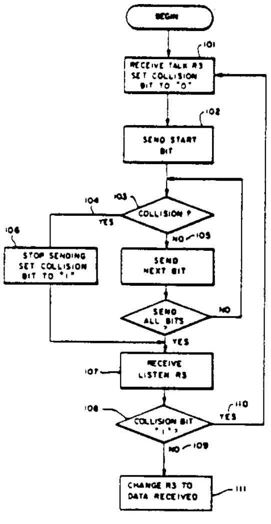
Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman

[57] **ABSTRACT**

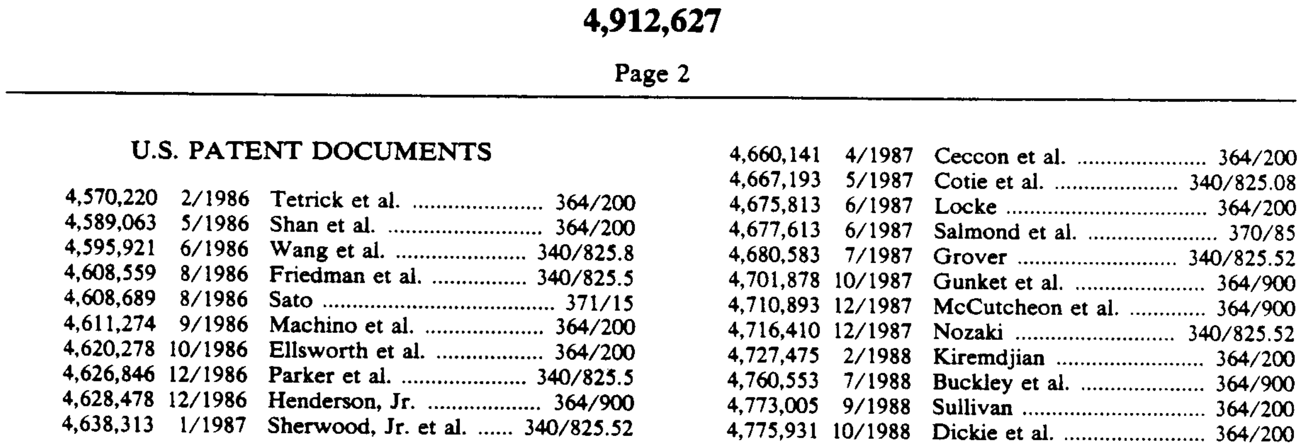
A method for transferring data on a communication medium between a host computer and a plurality of peripheral devices coupled to the medium at a plurality of first address locations, including first and second peripheral devices at a one of the first address locations. The host computer transmits a plurality of first signals to the one of the first address locations requesting any peripheral devices at the one of the first address locations to transmit data to the host computer. The first peripheral device senses the medium to determine if the medium is currently in use. The second peripheral device senses the medium to determine if the medium is currently in use. The first peripheral device transmit data to the host computer when the medium is not in use. The second peripheral device discontinues the transmission of data when the medium is in use and sets an internal collision flag. The host computer transmits a plurality of second signals, including a second address, to the one of the first address locations, the second address becoming an address for the first peripheral device, and the second address not becoming an address for the second peripheral device, given that the internal collision flag for the second peripheral device is set.

11 Claims, 4 Drawing Sheets

ADB



APPLE
DESKTOP
BUS



4,912,627

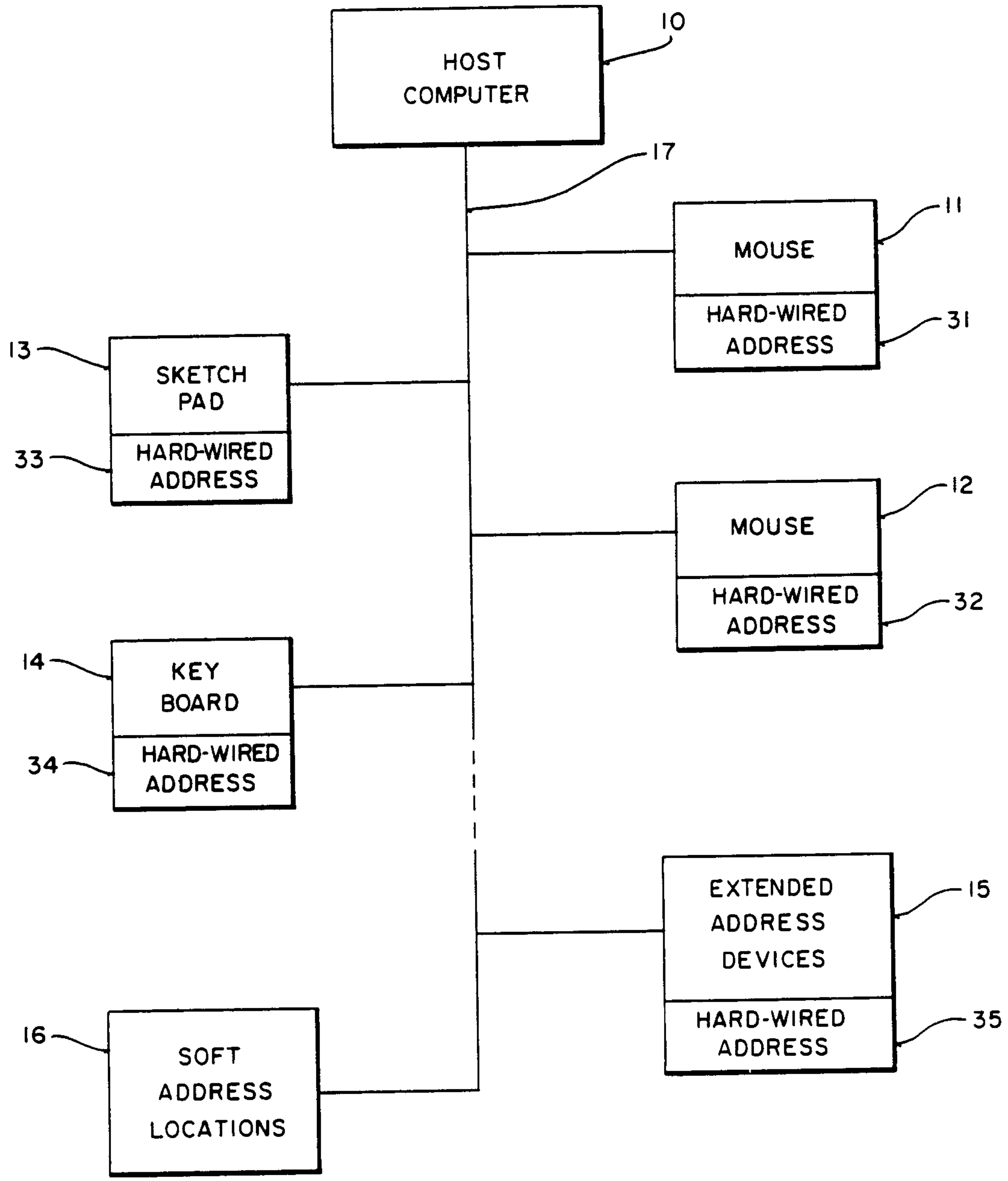
Page 2

U.S. PATENT DOCUMENTS

4,570,220	2/1986	Tetrick et al.	364/200
4,589,063	5/1986	Shan et al.	364/200
4,595,921	6/1986	Wang et al.	340/825.8
4,608,559	8/1986	Friedman et al.	340/825.5
4,608,689	8/1986	Sato	371/15
4,611,274	9/1986	Machino et al.	364/200
4,620,278	10/1986	Ellsworth et al.	364/200
4,626,846	12/1986	Parker et al.	340/825.5
4,628,478	12/1986	Henderson, Jr.	364/900
4,638,313	1/1987	Sherwood, Jr. et al.	340/825.52
4,660,141	4/1987	Ceccon et al.	364/200
4,667,193	5/1987	Cotie et al.	340/825.08
4,675,813	6/1987	Locke	364/200
4,677,613	6/1987	Salmond et al.	370/85
4,680,583	7/1987	Grover	340/825.52
4,701,878	10/1987	Gunket et al.	364/900
4,710,893	12/1987	McCutcheon et al.	364/900
4,716,410	12/1987	Nozaki	340/825.52
4,727,475	2/1988	Kiremdjian	364/200
4,760,553	7/1988	Buckley et al.	364/900
4,773,005	9/1988	Sullivan	364/200
4,775,931	10/1988	Dickie et al.	364/200

U.S. Patent Mar. 27, 1990 Sheet 1 of 4 4,912,627

FIG _ 1



“APPLE_PAT_4_912_627_03” 66 KB 2000-02-22 dpi: 300h x 300v pix: 1824h x 2546v

FIG_6

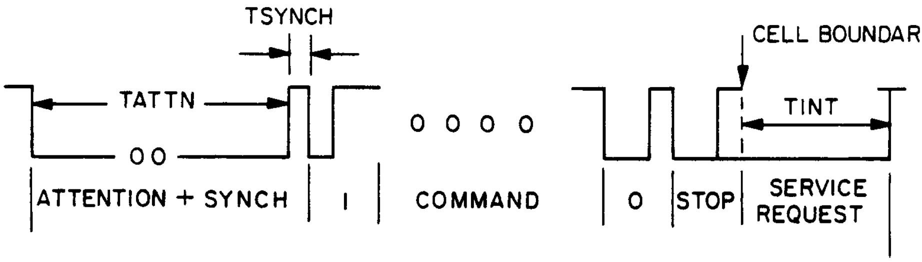


FIG_2

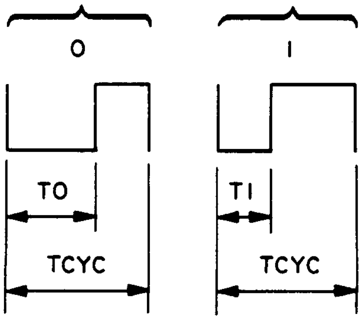


FIG_3

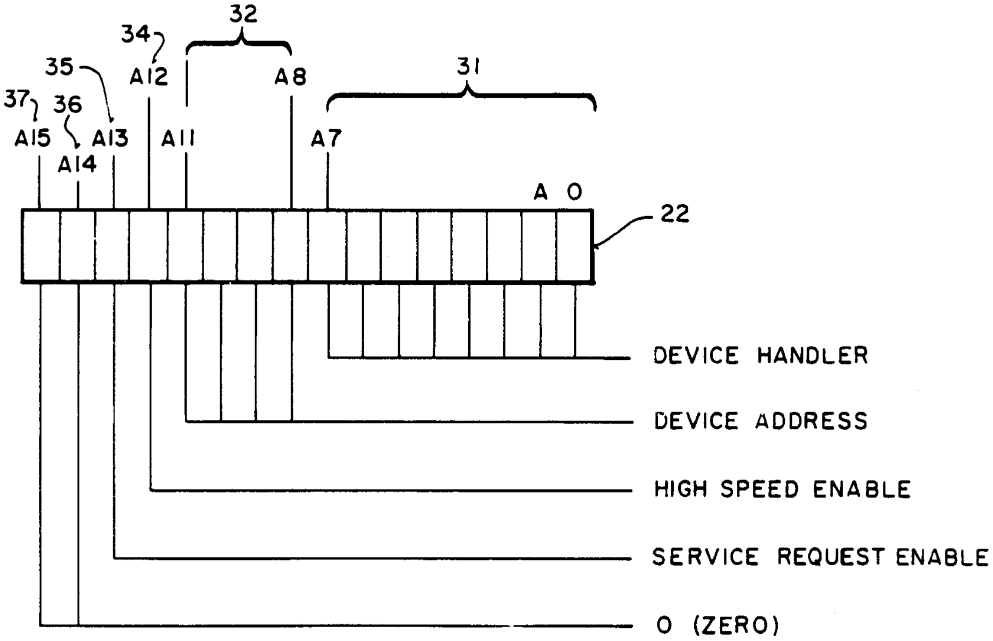
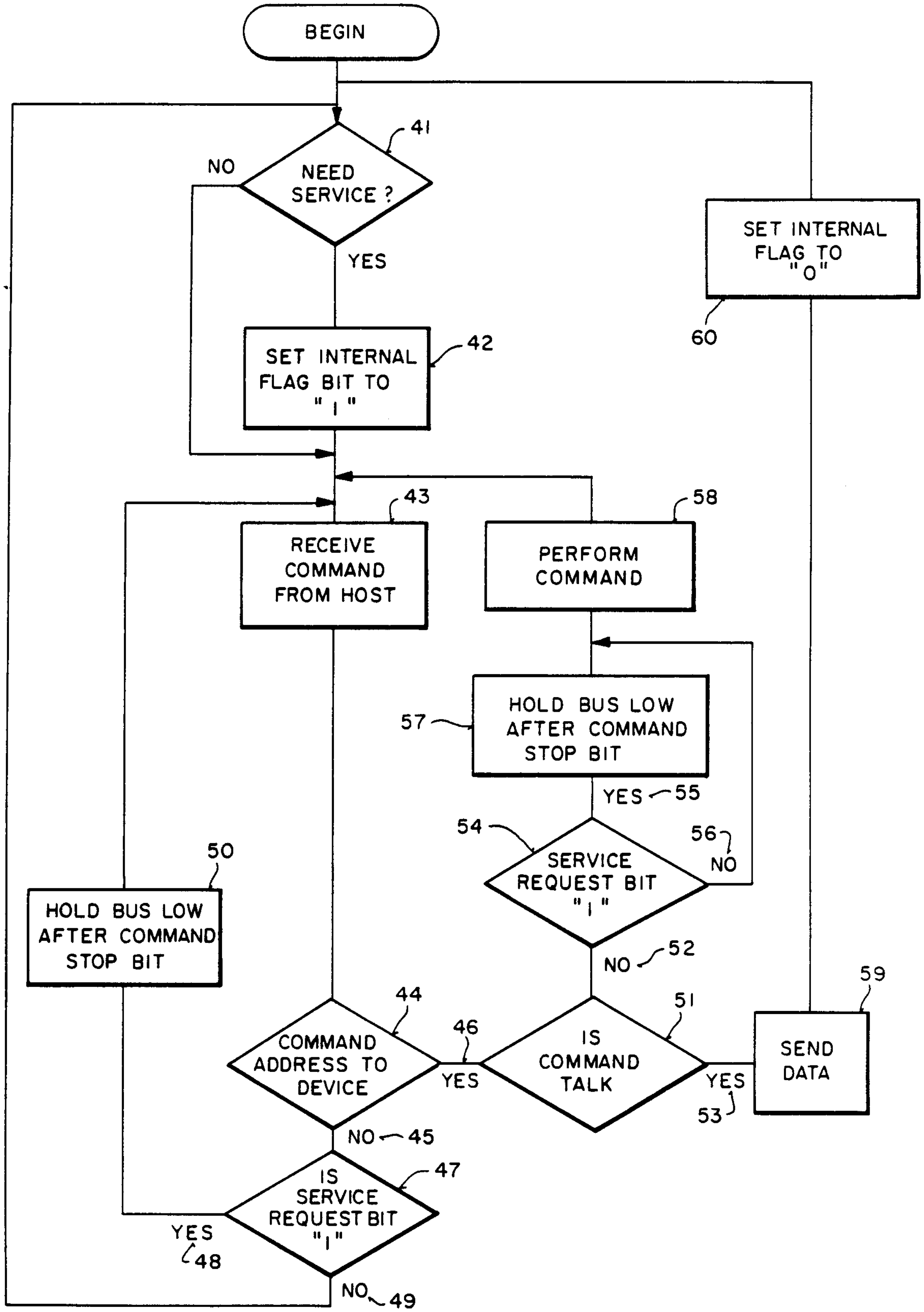
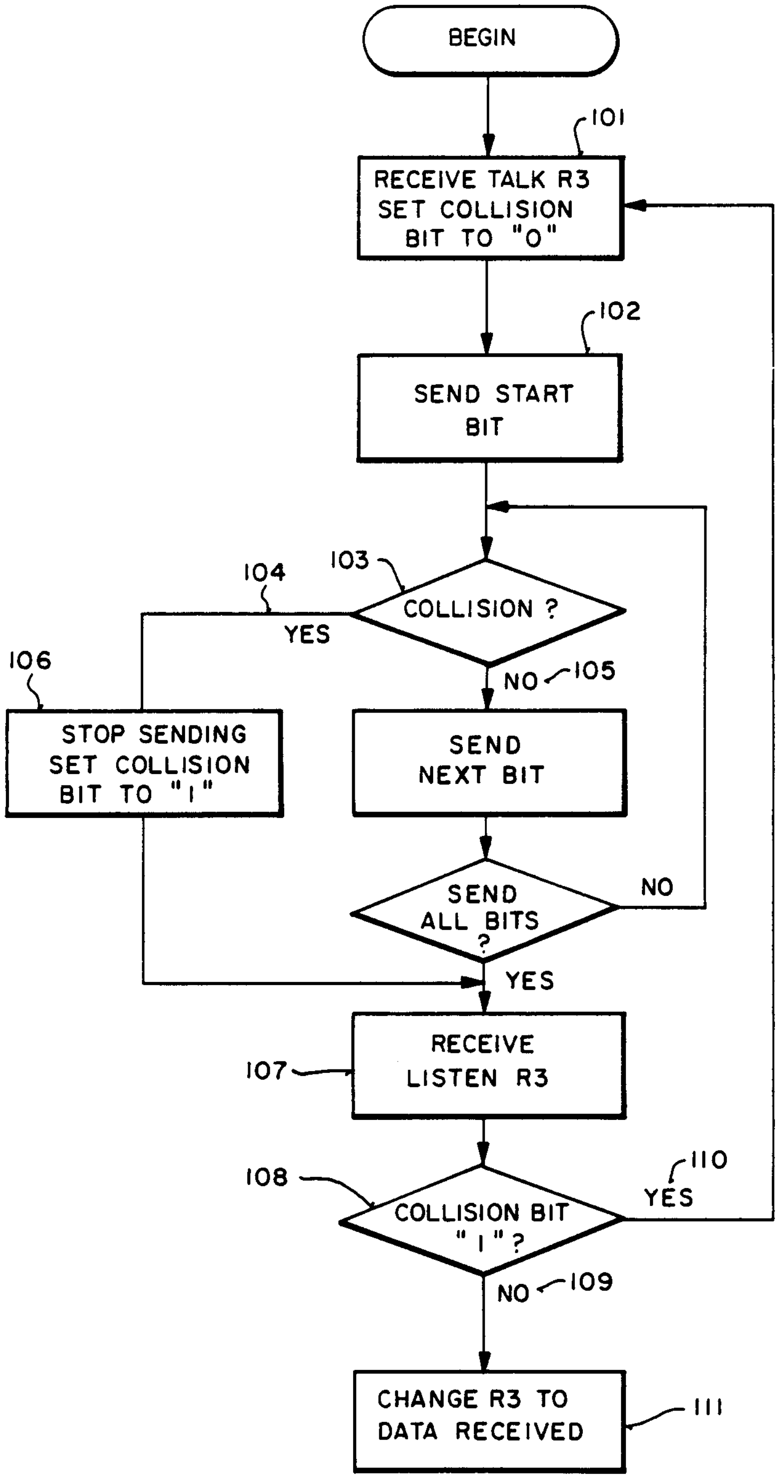


FIG_4


BEGIN
41
NEED SERVICE ?
NO
YES
SET INTERNAL FLAG BIT TO "1"
42
SET INTERNAL FLAG TO "0"
60
RECEIVE COMMAND FROM HOST
43
PERFORM COMMAND
58
HOLD BUS LOW AFTER COMMAND STOP BIT
57
YES
55
54
SERVICE REQUEST BIT "1"
56
NO
NO
52
50
HOLD BUS LOW AFTER COMMAND STOP BIT
44
COMMAND ADDRESS TO DEVICE
46
YES
51
IS COMMAND TALK
YES
53
SEND DATA
59
NO
45
IS SERVICE REQUEST BIT "1"
47
YES
48
NO
49

"APPLE_PAT_4_912_627_05" 119 KB 2000-02-22 dpi: 300h x 300v pix: 1898h x 2958v

FIG_5



“APPLE_PAT_4_912_627_06” 77 KB 2000-02-22 dpi: 300h x 300v pix: 1830h x 2854v

METHOD FOR STORING A SECOND NUMBER AS A COMMAND ADDRESS OF A FIRST PERIPHERAL DEVICE AND A THIRD NUMBER AS A COMMAND ADDRESS OF A SECOND PERIPHERAL DEVICE

This is a (divisional) of application Ser. No. 765,396 filed Aug. 14, 1985.

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention relates the field of communications media for transferring data between a source and a plurality of peripheral devices coupled to the source. More particularly, the present invention relates to data transfer along a peripheral device bus between a plurality of peripheral devices and a host computer.

2. Art Background

In the computing industry, it is quite common to transfer data and commands between a plurality of data processing devices, such as for example, computers, printers, memories and the like. The interconnection of computers and other peripheral devices principally developed in the early 1970's with the advent of computer networking systems, which permitted the distribution of access to computing resources beyond the immediate proximity of a main frame computer.

Networks, such as the ARPA network, were developed to provide access by various users to large timesharing systems and the transfer of data between such systems. In the case of geographically local networks, so-called "local area networks" (LANs) were developed to connect together a collection of computers, terminals and peripherals located, typically in the same building or adjacent buildings, and permitted each of these devices to communicate among themselves or wit devices attached to other networks. Local area networks permit the implementation of distributed computing. In other words, some of the devices coupled to the local area network may be dedicated to perform specific functions, such as file storage, data base management terminal handling, and so on. By having different machines perform different tasks, distributed computing can make the implementation of the system simpler and more efficient.

Presently, networking has only been applied to provide communications between data processing devices, which are machine input devices. However, it would also be useful to provide a networking means to provide communication between a single computer and a plurality of peripheral devices such as human input devices, listen only devices, appliances, etc. Human input devices include keyboards, cursor control devices (such as a "mouse"), and sketch pads, etc. Listen only devices include transaction logs, etc. In the prior art, such devices are attached to a host computer through a port dedicated to each device. Often, additional "cards" are required to allow a peripheral input device to be added. Further, the addition of cards requires that the host computer be powered down, with o mechanism for adding peripheral devices to a live system. Such prior art systems are inefficient since peripheral devices are not generally operated simultaneously. (for example, someone using a mouse is generally not using the keyboard or sketchpad at the same time). Thus, the devices could share a common line to the host computer without creating data traffic problems, eliminating the need for cards.

Prior art networking schemes also include elaborate methods for establishing control of the network to allow a device to transmit. Such systems are not needed for networking of peripheral devices, since only one is generally used at a time. In addition, prior art networking schemes provide for means for attached devices to identify themselves to each other through elaborate "handshaking" schemes. Again, such complexity is not required to connect peripheral devices since there is no need for these devices to identify themselves to other devices, only to the host computer.

Therefore, it is an object of the present invention to provide a communications medium for a plurality of peripheral devices, which provides a simple and efficient means for coupling those devices to a host computer.

It is a further object of the present invention to provide a communications medium by which all such peripheral devices can be coupled to a host computer at a single input.

It is still another object of the present invention to provide a communications medium which provides a means for peripheral devices to indicate a need for servicing to the host computer.

It is yet another object of the present invention to provide a communications medium which provides a means for determining if the communications medium is in use.

It is another object of the present invention to provide a communications medium which allows peripheral devices to be added during operation of the system.

SUMMARY OF THE INVENTION

A communications medium is disclosed including apparatus and methods for transferring data between a plurality of peripheral devices and a host computer. In the preferred embodiment, a plurality of peripheral devices such as human input devices (including mice, keyboards, sketchpads, etc.), appliances, listen only devices, etc., are coupled to a common cable for data transmission and reception of commands. A peripheral device coupled to the cable may signal the host computer when it requires servicing. This peripheral device will continue to request service until the host computer commands it to transmit its data. All peripheral devices of the same generic type (e.g., all keyboards), may have an identical hard wired address used as an identification number. In this manner, the host computer can identify the generic type of device communicating on the cable. If more than one of the same type of device is coupled to the cable (e.g., 2 mice), the host computer will assign new addresses in the status registers of the mice so they can be differentiated.

In the preferred embodiment, a return to zero modulation scheme is used to transmit data and commands over the cable. As a result, a peripheral device will assume a collision if it attempts to transmit a high signal on the cable and the cable is pulled low by another device. In order to simplify the protocol of the system, only the computer can initiate communication.

The present invention permits the addition of peripheral devices to a computer while the computer is in use, without the need to power down the computer system. The present invention can be embodied in a narrow band medium, as well as broad band, fiber optic, infrared and other media.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is block diagram illustrating the networking system of the present ,invention.

FIG. 2 is a timing diagram illustrating the present invention's use, of return to zero encoding.

FIG. 3 illustrates a register of a peripheral device of the present invention.

FIG. 4 is a flow chart illustrating the sequence of operations utilized by a peripheral device to request service by the host computer.

FIG. 5 is a flow chart illustrating the sequence the operations utilized to provide new addresses to devices sharing the same hard-wired address.

FIG. 6 is a timing diagram illustrating a command transaction of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

A peripheral device bus including apparatus and methods for transferring data between a plurality of peripheral devices coupled to a host computer is disclosed. In the following description numerous specific details are set forth, such as specific numbers, registers, addresses, times, signals, and formats, etc., in order to provide a thorough understanding of the present invention. However, it will be apparent to one skilled in the art that the present invention may be practiced without these specific details. In other instances, well-known circuits and devices are shown in block diagram form in order not to unnecessarily obscure the present invention.

Referring to FIG. 1, the preferred embodiment of the present invention may be seen. A plurality of peripheral devices, generally identified by numbers 11 through 16 are coupled through a single cable 17 to a host computer 10. In the preferred embodiment, all devices communicate with the host computer by a mini-phono jack with the following connecter assignments; tip-power, ring-data, sleeve-power return. A "high" signal (1) is 2.4 volts minimum. A "low" signal (0) is 0.8 volts maximum. Although a single cable is contemplated in the preferred embodiment of the present invention, other communications media, such as broad band methods, fiber optic systems, and infrared signals, are contemplated.

The bus of the present invention supports coded devices, (for which a keystroke represents a symbol or a function, such as a keyboard 14), relative devices, (in which movement of a display cursor in response to a control device, such as a mouse 11 or 12, may be from any starting point), and absolute devices (for which there is a constant and direct relationship between display position and device position, such as sketch pad 13).

The system also permits the networking of extended address devices. Extended address devices share a common hard wired address 35, but further include an address unique to the individual device which the host computer must recognize before the device can be accessed. For example, it is contemplated that appliances may be coupled to the host computer and controlled by the host computer. In such a situation, all appliances would have an identical hardwired fixed address. The host computer, on a first level, would simply address the hard wired address for appliances. At this time, all appliances coupled to that address are inactive. An individual appliance may be activated by the host computer if the host computer sends a signal to that appliance which matches the extended address of the appliance. An extended address i an individual identification number, which, in the preferred embodiment, may be up to 64 bytes long. Once the host computer has provided the extended address, the device having that address is active. Subsequent commands to the appliance address location will be executed by that device without the need for providing the extended address each time. An activated appliance will respond to all commands to the appliance address, while unactivated devices remain passive. To deactivate an active extended address device, the host computer provides the extended address of another extended address device, activating it and deactivating the previously active device. It is contemplated that any device which could be controlled by the host computer is suitable for the present networking scheme, such as lights, ovens, sprinkler systems, phone answering machines, etc. It is contemplated that at leas one other hardwired address for extended address devices be provided in the present system. Such an address would be used for system protection schemes or user identification schemes. For example, a device at this location could contain an extended address which must be provided by the system user before the system could be enabled. In other instances, individual operations could require that the extended address of other security devices be provided by the host computer prior to performance. Such security devices could function as "keys" to lock the entire system or certain operations performed on the system.

Also reserved for use on the network of the present invention are soft address locations 16. Soft address locations ar reserved for duplicates of peripheral devices coupled to the bus. When more than one mouse is coupled to the bus, for example, the host computer assigns new addresses to each mouse, those addresses being at tee soft address locations.

Although specific examples have been given for each type of device coupled to the bus, there may be more than one kind of each type of device with that address. For example, a sketch pad has been given as an absolute device but a touch screen would also be considered an absolute device and be assigned the same fixed command address as the sketch pad. In those situations, the host computer will assign new addresses from the soft address locations to each device.

In the preferred embodiment of the present invention, the various peripheral devices have been assigned addresses as shown below:

Address	Device Types	Example
0000 (zero)	extended address device	security systems, user ID
0001 (one)	extended address device	appliances
0010 (two)	coded devices	keyboard
0011 (three)	relative devices	mouse, track ball
0100 (four)	absolute devices	sketchpad, touch screen
0101 (five)	reserved	none
0110 (six)	reserved	none
0111 (seven)	reserved	none
1000 (eight)	soft addressed	duplicate peripheral devices
. . .	. . .	. . .
. . .	. . .	. . .
1111 (15)	soft addressed	duplicate peripheral devices

It will be appreciated by one skilled in the art that other addresses may be assigned to these devices containing more or less bits than in the preferred embodiment. Fixed hard-wired addresses **31**, **32**, **33** and **34** are shown in FIG. **1** for mouse **11**, mouse **12**, sketch pad **13**, and keyboard **14**, respectively.

All peripheral devices have four registers in the preferred embodiment to receive data and send data. For each device, register **3** talk and register **3** listen have status information such as device address and handler information. The remaining registers are data registers which are device specific except register **2** listen which contains the extended addresses for extended address devices or device specific contents for soft addressed devices.

In the preferred embodiment of the present invention, there are three types of communication on the peripheral bus: commands, data and global signals. Commands are sent from the host computer to the peripheral devices, data is sent from the host computer to the devices or from the devices to the host computer, and global signals are special messages sent to the entire system.

In the preferred embodiment data is encoded as the ratio of low time to high time of each bit cell A bit cell boundary is defined by a falling edge on the bus. A "zero" is encoded as a bit cell in which the low time is greater than the high time. This is shown in FIG. **2** by bit cell **20**. Therefore, a "1" is defined as a bit cell in which the low time is less than the high time as shown by cell **21** of FIG. **2**. In the present preferred embodiment, a start bit is defined as a "1". A stop bit is a "0" which does not have an additional falling edge to define the bit cell time. The stop bit is used to synchronize the stopping of transactions on the bus.

The period for each bit cell of command signals and low speed data transmission is approximately 100 microseconds plus or minus 30%. For high speed data transmission, the bit cell is 50 microseconds plus or minus 1%. The format of a data transaction is a start bit (**1**), followed by up to 256 bits of data and ending with a stop bit. It will be appreciated that when other communications media are utilized, other signaling methods may be utilized.

Commands are sent only by the host. In the preferred embodiment of the present invention, there are three commands; talk, listen, and flush. As shown in FIG. **6**, to signal the start of a command, an attention pulse is sent out. An attention pulse is generated by the host computer by transmitting a bus low for a period of "T-attn". In the preferred embodiment, T-attn is approximately 560–1040 microseconds. The attention pulse is followed by a synch pulse to give the initial bus timing. The following edge of the synch pulse is used as a timing reference for the first bit of the command. The command is followed by a stop bit, (in the preferred embodiment a "0"). After the stop bit, the bus returns to its normally high state unless a device requests service.

The command is an 8 bit value in the preferred embodiment. The command includes a 4 bit device address field which specifies the fixed hardwired address of the desired peripheral device (e.g., 0011 for a mouse). The next 2 bits form the command and the final 2 bits form a register address field which allows a specific register, R0–R3 within an addressed peripheral device to be specified. In the preferred embodiment, the commands have the following bit code:

Command	Code
Flush	01

-continued

Command	Code
Listen	10
Talk	11

The talk command orders the addressed device to provide its data to the host computer. The listen command orders the addressed device to accept data from the host computer and place it in one of its registers. The flush command has an effect on each device which is defined by the individual device. It can be used for such functions as clearing a register or resetting all keys on a keyboard so that they will be sent again.

When a peripheral devices is addressed to talk, it must respond within a certain period, called the "time out" period. The time out, "Tlt", is approximately 140 to 260 microseconds (2 bit cells). The selected device, if it does not time out, becomes active on the bus and performs its data transaction, and then "untalks" itself and goes inactive on the bus.

Global signals are used for transactions which are neither commands nor data transactions. Global signals include: attention and synch, which is used to signal the start of a command and to give initial bus timing; service request, a transaction that devices use to signal the host that they require service; and reset, used to issue a break on the bus by holding the bus low for a minimum of "Tres", which is approximately 2.8 to 5.2 milliseconds, (40 bit cells). Global signals will be described in more detail in conjunction with other transactions.

Since a peripheral device can only send data when it has been commanded to talk by the host computer, the present system provides a means for a device to notify the host computer that it needs servicing. This is accomplished by having the device send a service request signal to the host computer. In the present invention, a service request is sent by holding the bus low after the stop bit of any command transaction. Each of the peripheral devices coupled to the bus include a number of registers (in the preferred embodiment four registers). FIG. **3** shows one of the registers for a peripheral device. Bit A13 has been identified as the service request enable bit. When this bit is set high by the host computer, the device is enabled to hold the bus low after the stop bit of a command transaction, as shown in FIG. **6**, if the device needs service. A device will keep requesting service until it receives a talk command from the host. The flow chart in FIG. **4** shows the steps followed by a device requiring service.

Initially the device determines if it requires servicing Block **41**, that is, if it has data to send to the host. If it does, it sets an internal flag bit Block **42**. When the next command is sent out from the host Block **42**, the device checks to see if the command is addressed to the device Block **44**. If the command was not addressed to the device Branch **45**, the device checks to see if its service request enable bit, (bit A13 of register **3**), is set high Block **47**. If so Branch **48**, it holds the bus low after the command stop bit Block **50**. (See FIG. **6**) The device then waits until the next command is received from the host to see if it will be addressed to talk Block **43**. If the command is addressed to the device Branch **46**, the device determines if it is a command to talk Block **51**. If it is not a command to talk Branch **52** the device sends a service request Block **57**, performs whatever command is instructed Block **58**, and awaits the next command Block **43**. If the command is to talk Branch **53**, the device sends its data Block **59** and considers its service request to be satisfied Block **60**. The device continues to monitor itself to determine when it needs service Block **41**. By allowing the host computer to control the service request enable bit, more efficient operation of the bus is realized. When a service request is received, the host computer need only ask those devices whose service request bit was enabled whether they need servicing. Additionally, the host computer can disable certain devices that are not required for particular applications.

When sending data, the device is able to detect collisions. If a peripheral device tries to output a 1 and the data line is or goes to a 0, the device assumes it has lost a collision to another device. This means that another device is also sending on the bus. When this happens the losing device untalks itself from the bus and preserves the data which was being sent for retransmission. The device sets an internal flag bit if it loses a collision. Prior art peripheral devices were unable to detect collisions. This novel feature of the present invention permits more efficient operation of the communications medium. By having the device sense a collision, it can preserve the data that is transmitted and indicate to the host computer that it requires serving. Additionally, the collision detection scheme of the present invention does not require a waiting period before a collision is assumed. A device will end its transmission if the line is modulated by another device or simply not begin its transmission if he line is already in use. Further, this collision detection scheme is useful in locating multiple devices at a single hardwired address location, such as mouse **11** and mouse **12** of FIG. **1**.

In such a situation, the host will change the address of the devices by forcing a collision of devices sharing the same address. The host achieves this by issuing a talk **R3** command addressed to those devices. As shown in FIG. **3**, Register **322** (one of the registers of the device) contains the following information. Bits **A0** through **A731** contain a device handler which tells the host computer the function of a device and the use of data provided by the device. Bits **A8** through **A1132** are an address field which can be changed when more than one device, having the same command address, is coupled to the bus. In that situation, one of the soft address locations are assigned to bits **A8** through **A1132** which then serve as the command address for that device. Until that time, those bit locations contain a random number which aids in the detection of collisions. For example, if two mice received a talk **R3** talk command and both began talking at the same, neither would detect a collision. However, by having random numbers in the address field **32** of register **322**, the output of the two devices will eventually differ. When that occurs, one of the devices will detect a collision and stop talking. Bit **A1234** is a high speed enable bit which if set, provides for data transmission at the higher modulation rate (50 microseconds per bit frame). The high speed enable bit is set by the host computer. If the host computer is unable to receive data at the higher modulation rate, it sets the high speed enable bit low in each of the devices. If the host computer is able to accept data at the higher modulation rate, and the device is able to transmit at the higher rate, (that information being contained in the handler bits **31** of register **3**), the host computer sets the high speed enable bit **34** high for the device. As previously mentioned, bit **A13 35** is service request enable which is set by the host to enable the device to perform a service request transaction. Bit **A1436** and **A1537** are reserved for future use and are set to 0.

When a device receives a talk **R3** command the device provides its status (handler and address) to the host computer. If there are two devices of the same type coupled to the bus, only one can respond since the other will detect a collision. FIG. **5** shows the method of assigning new addresses on the bus.

After receiving a talk **R3** signal Block **101** the device sends its status from Register **3**. If the line goes low, the device determines that there has been a collision Branch **104**, it stops sending (untalks itself) and sets an internal flag bit to indicate a collision Block **106**. The host sends a listen **R3** to tee mouse address Block **107**. Each command resets the internal collision flag of the device. The device checks to see if its collision bit is set Block **108**. If the collision bit is not set Branch **109** the device changes **A8** through **A11** to the soft address provided by the listen **R3** command Block **111**. In this manner the address of the winning device is changed with the host computer keeping track of the new address of the device. If a collision bit is detected by the device after a listen **R3** command Branch **110**, the device does no change the soft address bits, but may change other fields in **R3**. The host computer sends out another talk **R3** command Branch **101** to see if any devices remain at the mouse address. In this situation the remaining mouse will send its start bit Block **102**, not detect a collision Branch **105**, and send its status from register **3** Block **112**. The host computer will send back a listen **R3** command to the mouse address Block **107**. The remaining mouse will not detect a collision bit being set in this instance Branch **109** so it will change bits A8 through **A11** of register **3** to the soft address received from the host computer Block **111**. The host computer then sends out another talk **R3** command to the mouse address Block **101**. This time, since no mouse remains at that address, the bus is timed out and the host computer knows that it has assigned new addresses to each of the mice sharing the mouse address.

In one embodiment of the present invention, peripheral devices have a device on them to indicate activity called the activator. The activator can be a special key on a keyboard or a button on a mouse. When more than one of a device is coupled to the bus, the host computer can display a message requesting one of the devices to use the activator. The host can then issue a listen R3 command which will change the address of the device which is activated. In this manner individual devices can be located and assigned new addresses in multiuser applications.

Thus, a peripheral device bus has been described which allows a plurality of peripheral devices to be coupled to a host computer through a single port.

What is claimed is:

1. A method for transferring signals and data, wherein the signals and data are transferred under the control of a host computer between the host computer and first and second peripheral devices, wherein the signals and data are transferred over a bus coupling the first and second peripheral devices to the host computer, wherein the bus is normally in a logical first state, and wherein the first and second peripheral devices each initially contain an identical first number as a command address for both the first and second peripheral devices, comprising the steps of:

the host computer transmitting a plurality of first signals over the bus, wherein the plurality of first signals include a first talk command requesting any peripheral device with the first number as its command address to (1) transmit data to the host computer and (2) reset a collision detect bit of that peripheral device to a logical second state;

given that the first number is the command address of the first peripheral device, the first peripheral device responding to the first talk command by attempting to transmit data over the bus to the host computer and, at the same time, the first peripheral device looking for a collision that indicates that the bus is currently in use, wherein a collision is detected if the first peripheral device attempts to transmit data in the logical first state on the bus but the bus is in or goes to the logical second state;

given that the first number is the command address of the second peripheral device, the second peripheral device responding to the first talk command by attempting to transmit data over the bus to the host computer and, at the same time, the second peripheral device looking for a collision that indicates that the bus is currently in use, wherein a collision is detected if the second peripheral device attempts to transmit data in the logical first state on the bus but the bus is in or goes to the logical second state;

the first peripheral device not detecting a collision, and the first peripheral device transmitting its data to the host computer over the bus;

the second peripheral device detecting a collision and, as a result, (1) the second peripheral device setting the collision detect bit to the logical first state from the logical second state and (2) stopping the transmitting of data to the host computer from the second peripheral device over the bus;

the host computer transmitting a plurality of second signals over the bus, wherein the plurality of second signals include a first listen command requesting any peripheral device with the first number as its command address to accept data sent by the host computer;

the host computer sending a second number stored at a first soft address location as data over the bus;

given that the first number is the command address of the first peripheral device, the first peripheral device responding to the first listen command by storing the second number as the command address of the first peripheral device;

the second peripheral device not accepting the second number from the bus because the collision detect bit of the second peripheral device is set to the logical first state;

the host computer transmitting a plurality of third signals over the bus, wherein the plurality of third signals include a second talk command requesting any peripheral device with the first number as its command address to (1) transmit data to the host computer and (2) reset the collision detect bit of that peripheral device to the logical second state;

the first peripheral device not responding to the second talk command, given that the first number is no longer the command address of the first peripheral device;

given that the first number is the command address of the second peripheral device, the second peripheral device responding to the second talk command by attempting to transmit data over the bus to the host computer and, at the same time, the second peripheral device looking for a collision that indicates that the bus is currently in use, wherein a collision is detected if the second peripheral device attempts to transmit data in the logical first state on the bus but the bus is in or goes to the logical second state;

the second peripheral device not detecting a collision, and the second peripheral device transmitting its data to the host computer over the bus;

the host computer transmitting a plurality of fourth signals over the bus; wherein the plurality of fourth signals include a second listen command requesting any peripheral device with the first number as its command address to accept data sent by the host computer;

the host computer sending a third number stored at a second soft address location as data over the bus;

the first peripheral device not responding to the second listen command, given that the first number is not longer the command address of the first peripheral device;

given that the first number is the command address of the second peripheral device, the second peripheral device responding to the second listen command by storing the third number as the command address of the second peripheral device.

2. The method of claim **1** for transferring signals and data, further comprising the steps of:

the host computer transmitting a plurality of fifth signals on the bus, wherein the plurality of fifth signals include a third talk command requesting any peripheral device with the first number as its command address to (1) transmit data to the host computer and (2) reset the collision detected bit of that peripheral device to the logical second state;

the first peripheral device not responding to the third talk command given that the first number is no longer the command address of the first peripheral device;

the second peripheral device not responding to the third talk command given that the first number is no longer the command address of the first peripheral device;

a time out period elapsing without the first and second peripheral devices responding, which indicates to the host computer that the host computer has completed assigning new command addresses to the first and second peripheral devices.

3. The method of claim **2** for transferring signals and data, wherein

the plurality of first signals comprise an attention signal, a synchronization signal, the first talk command, and a stop signal;

the plurality of second signals comprise an attention signal, a synchronization signal, the first listen command, and a stop signal;

the plurality of third signals comprise an attention signal, a synchronization signal, the second talk command, and a stop signal;

the plurality of fourth signals comprise an attention signal, a synchronization signal, the second listen command, and a stop signal; and

the plurality of fifth signals comprise an attention signal, a synchronization signal, the third talk command, and a stop signal.

4. The method of claim **3** for transferring signals and data, further comprising the step of any of the first and second peripheral devices requiring service generating a service request signal by holding the bus at a logical second state for a period of time after transmission of a plurality of signals from the host computer on the bus, wherein the service request signal indicates to the host computer that at least one of the peripheral devices has data to send to the host computer and requests a command from the host computer that would permit the peripheral device to transmit the data to the host computer, and wherein transmission of the service request signal is selectively enabled or disabled by the host computer.

5. The method of claim **4** for transferring signals and data, wherein at least one of the first and second peripheral devices comprises a mouse.

6. The method of claim **4** for transferring signals and data, wherein at least one of the first and second peripheral devices comprises a sketch pad.

7. The method of claim **4** for transferring signals and data, wherein at least one of the first and second peripheral devices comprises a keyboard.

8. The method of claim **4** for transferring signals and data, wherein the data is transferred on the bus in a return-to-zero encoding scheme.

9. The method of claim **4** for transferring signals and data, wherein the logical first state is a logical high state and the logical second state is a logical low state.

10. The method of claim **9** for transferring signals and data wherein a third peripheral device is coupled to the host computer by the bus, wherein when the host computer sends over the bus a first address of the third peripheral device and a signal that matches an extended address of the third peripheral device, the third peripheral device is initially activated, and wherein when the host computer then sends over the bus a subsequent command to the first address of the third peripheral device, the command is executed by the third peripheral device without the host computer sending the extended address of the third peripheral device.

11. The method of claim **10** for transferring signals and data, wherein a fourth peripheral device is coupled to the host computer by the bus, wherein the fourth peripheral device has the same first address as the third peripheral device, wherein the fourth peripheral device has an extended address different from the extended address of the third peripheral device, wherein when the host computer sends over the bus the extended address of the fourth peripheral device after the third peripheral device has been activated, the fourth peripheral device is initially activated and the third peripheral device is deactivated, and wherein when the host computer then sends over the bus a subsequent command to the first address of the fourth peripheral device, the command is executed by the fourth peripheral device without the host computer sending the extended address of the fourth peripheral device.

* * * * *

United States Patent [19]
Sander et al.
[11] Patent Number: 4,916,556
[45] Date of Patent: Apr. 10, 1990

[54] DISK DRIVE CONTROLLER

[75] Inventors: Wendell Sander; Brian Sander, both of Campbell, Calif.

[73] Assignee: Apple Computer, Inc., Cupertino, Calif.

[21] Appl. No.: 280,665

[22] Filed: Dec. 6, 1988

Related U.S. Application Data

[62] Division of Ser. No. 55,443, May 28, 1987.

[51] Int. Cl.^{4} G11B 5/09
[52] U.S. Cl. 360/45
[58] Field of Search 360/45, 51, 40

[56] References Cited

U.S. PATENT DOCUMENTS

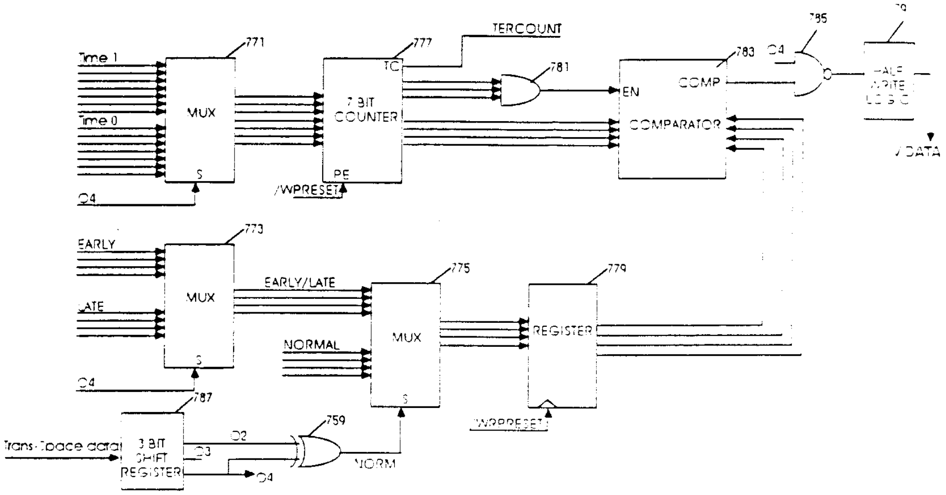
4,081,756	3/1978	Price et al.	360/45
4,281,356	7/1981	Sousa	360/45
4,327,383	4/1982	Holt	360/45
4,344,093	8/1982	Huber	360/45

Primary Examiner—Vincent P. Canney
Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman

[57] ABSTRACT

The invented controller uses a programmable parameter scheme which makes it possible to read and write 3½ inch variable and fixed speed drives, as well as standard 5¼ inch drives. Additionally, the present invention uses a plus/minus rate multiplier to correct for symmetry and frequency errors. Also provided is a form of read post compensation which corrects for peak shift effects on disks with insufficient precompensation. Other advanced features of the present invention include the use of half clock circuits to provide half clock resolution in the signal being written to and read from the disk and the capability of operating at continuously variable clock speeds and data rates dynamically programmable by the computer.

2 Claims, 13 Drawing Sheets



SWIM CHIP
SUPER WOZ/WENDELL INTEGRATED MACHINE

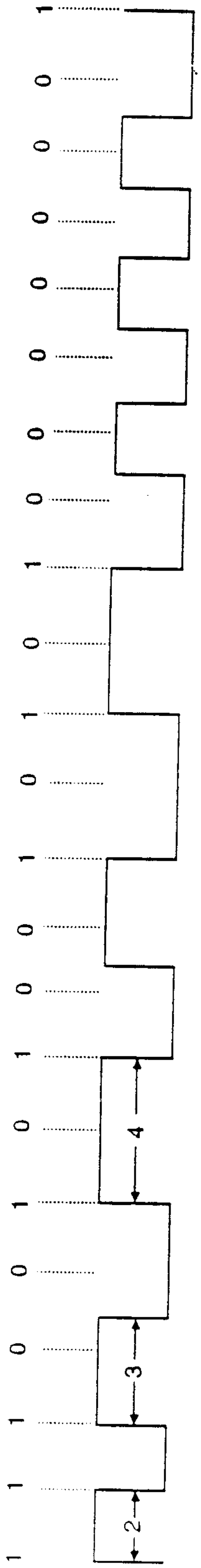


Fig. 1

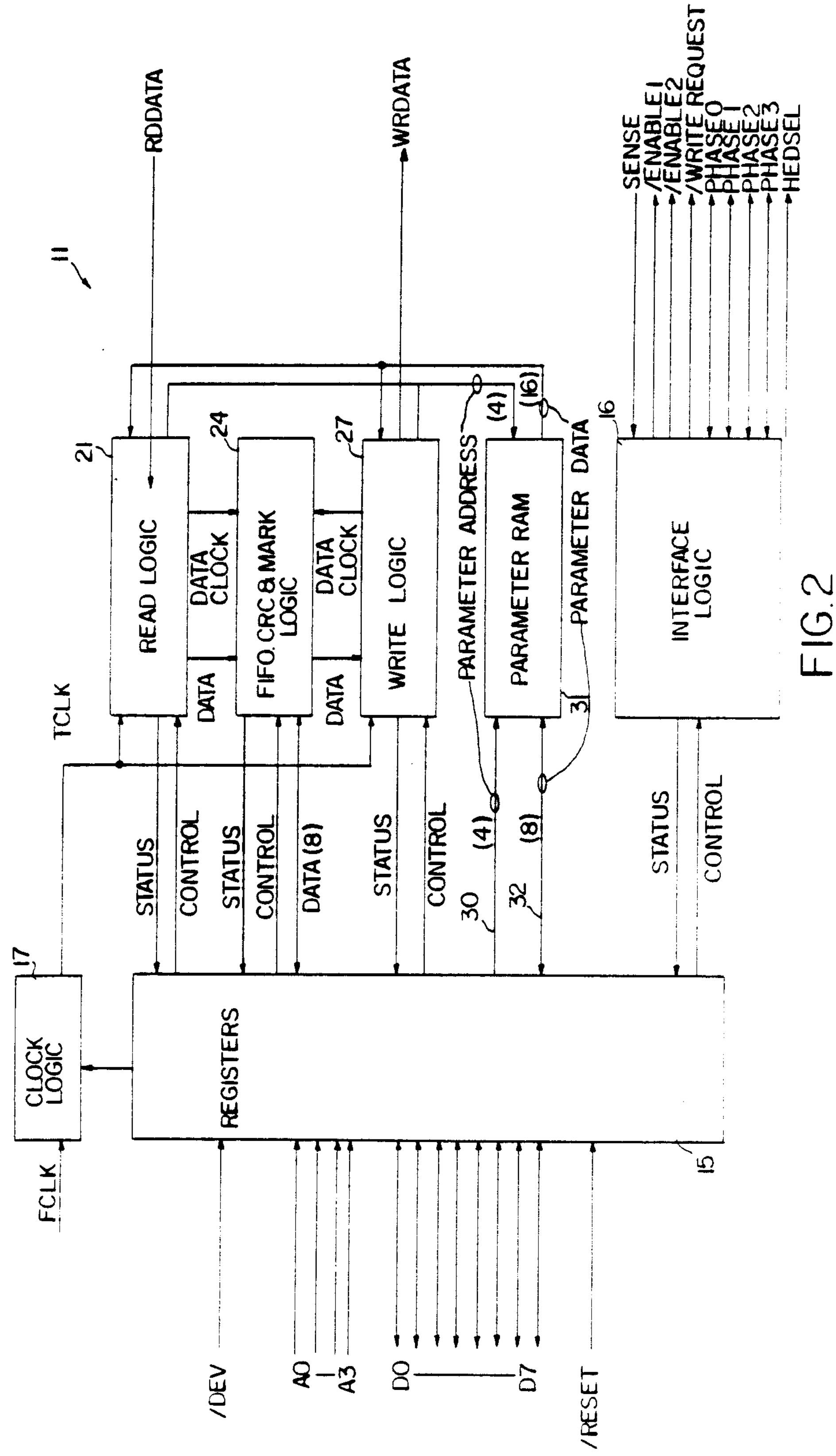

11
RDDATA
WRDATA
SENSE
/ENABLE1
/ENABLE2
/WRITE REQUEST
PHASE0
PHASE1
PHASE2
PHASE3
HEDSEL
21
24
27
16
READ LOGIC
DATA CLOCK
FIFO, CRC & MARK LOGIC
DATA CLOCK
WRITE LOGIC
PARAMETER ADDRESS
PARAMETER RAM
PARAMETER DATA
(4)
(16)
31
INTERFACE LOGIC
DATA
DATA
TCLK
STATUS
CONTROL
STATUS
CONTROL
DATA (8)
STATUS
CONTROL
(4)
(8)
30
32
STATUS
CONTROL
17
CLOCK LOGIC
REGISTERS
15
FCLK
/DEV
A0 — A3
D0 — D7
/RESET

FIG. 2

U.S. Patent Apr. 10, 1990 Sheet 3 of 13 4,916,556

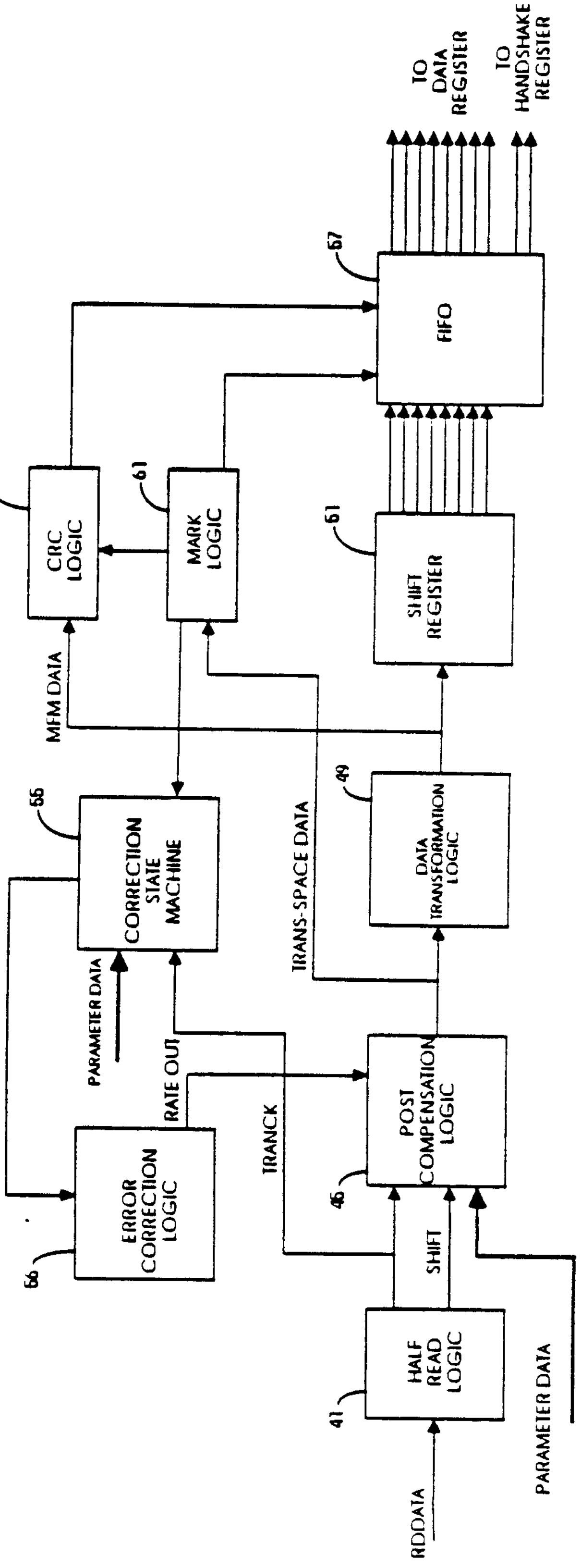


FIG. 3

“APPLE_PAT_4_916_556_04” 88 KB 2000-02-22 dpi: 300h x 300v pix: 1855h x 2779v

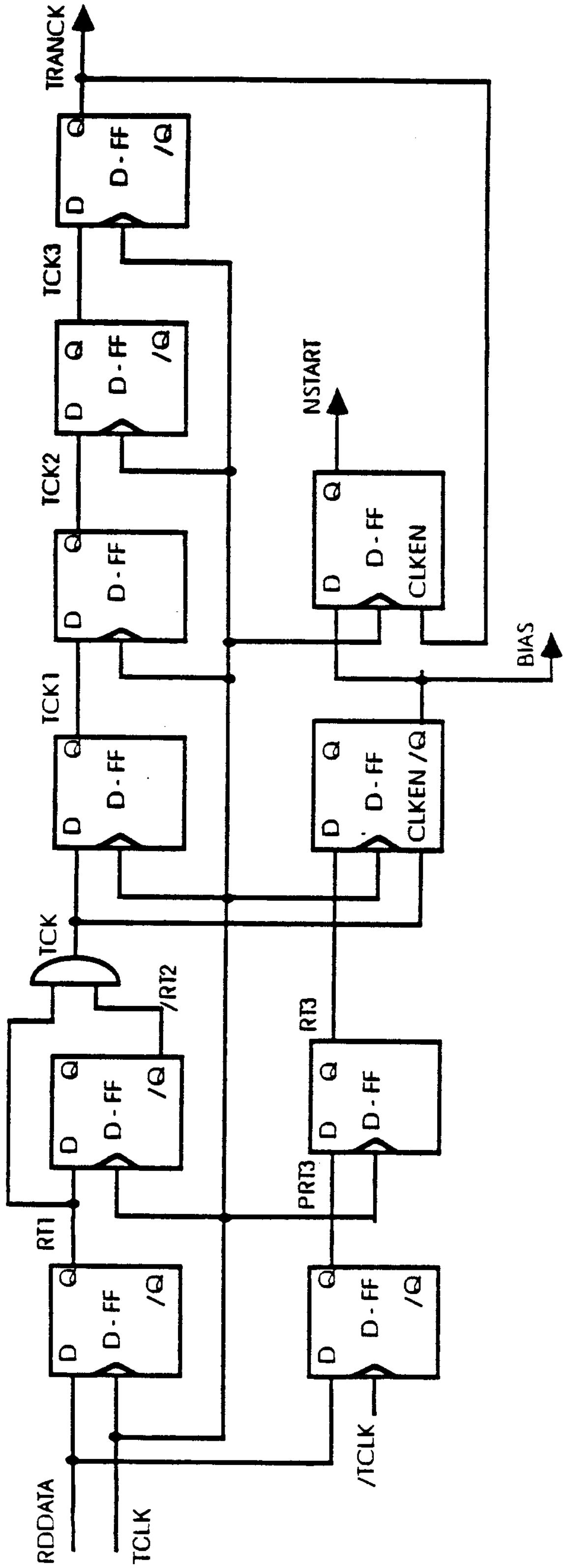

TRANCK
TCK3
TCK2
TCK1
TCK
/RT2
RT1
RDDATA
TCLK
/TCLK
PRT3
RT3
NSTART
BIAS
D-FF
D
Q
/Q
CLKEN
CLKEN /Q

FIG. 4a

U.S. Patent Apr. 10, 1990 Sheet 5 of 13 4,916,556

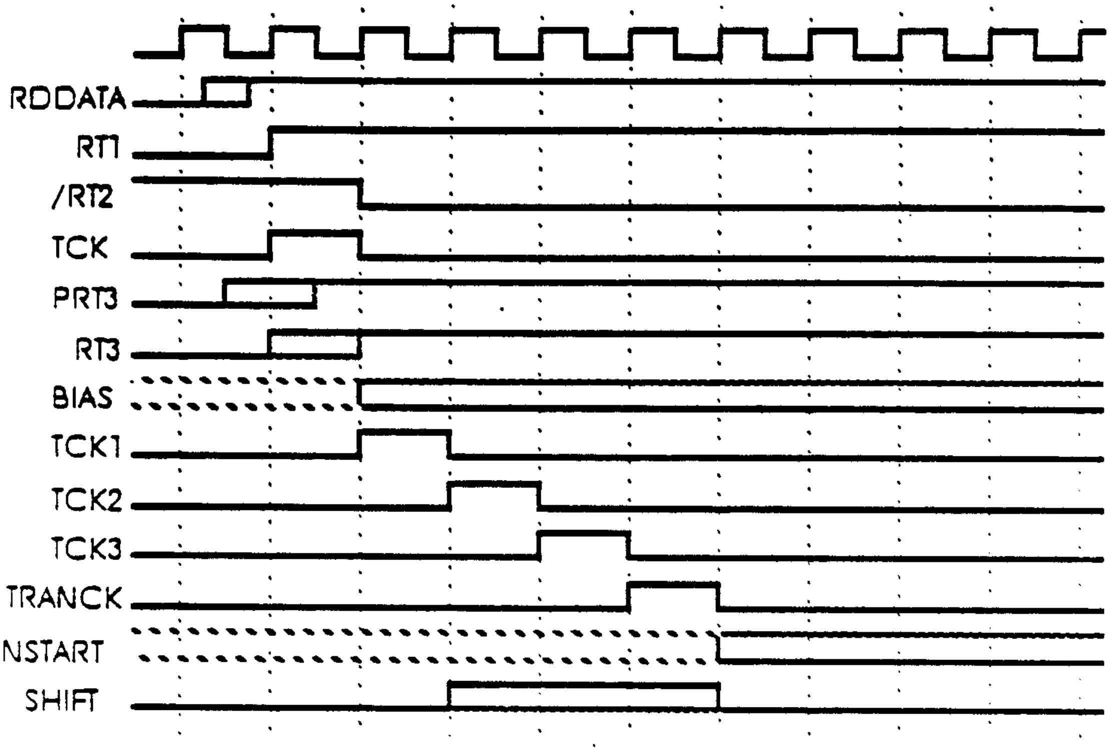


FIG. 4b

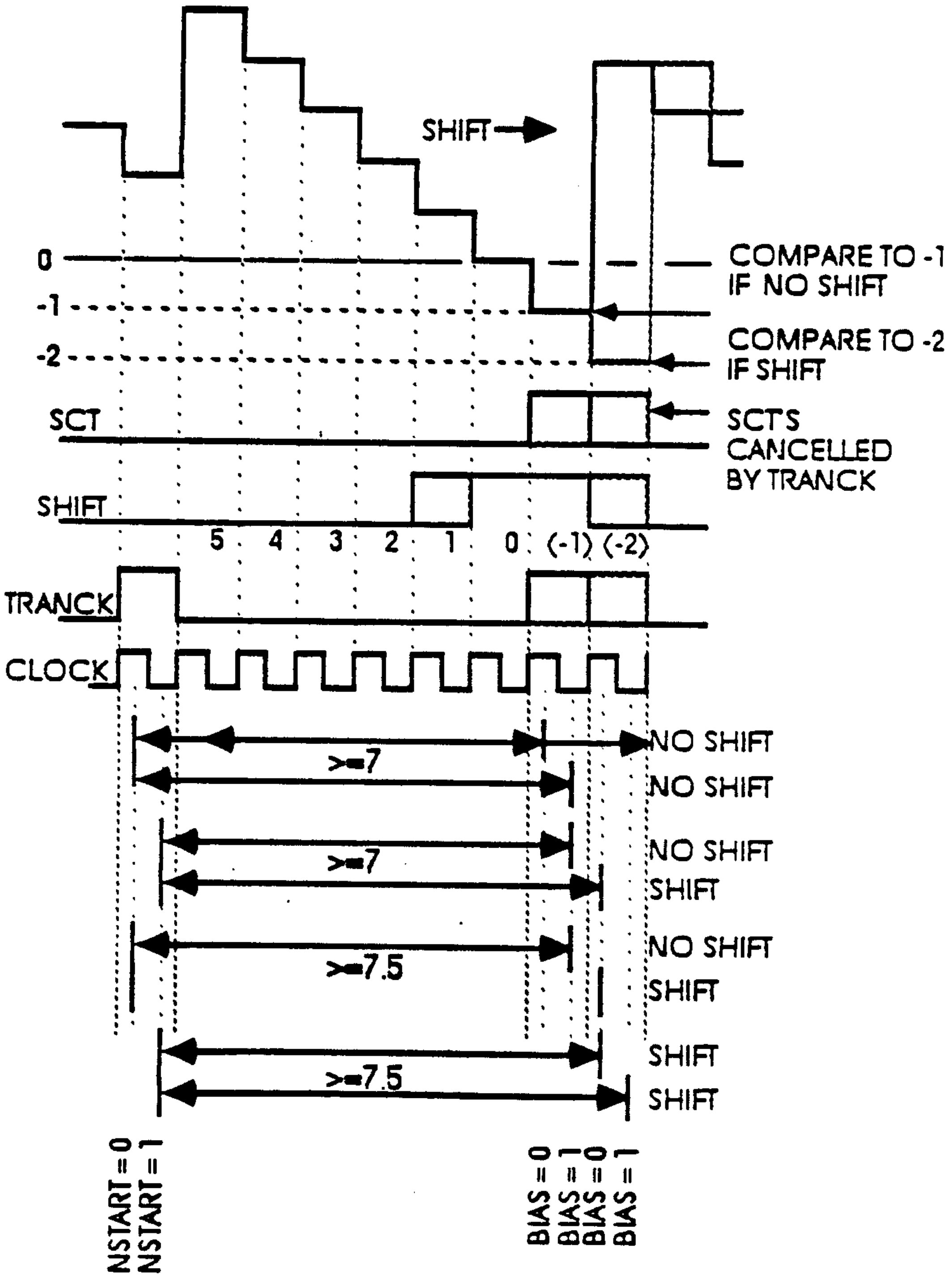

SHIFT
0
-1
-2
COMPARE TO -1
IF NO SHIFT
COMPARE TO -2
IF SHIFT
SCT
SCT'S
CANCELLED
BY TRANCK
SHIFT
5
4
3
2
1
0
(-1)
(-2)
TRANCK
CLOCK
>=7
NO SHIFT
NO SHIFT
>=7
NO SHIFT
SHIFT
>=7.5
NO SHIFT
SHIFT
>=7.5
SHIFT
SHIFT
NSTART = 0
NSTART = 1
BIAS = 0
BIAS = 1
BIAS = 0
BIAS = 1

FIG. 4c

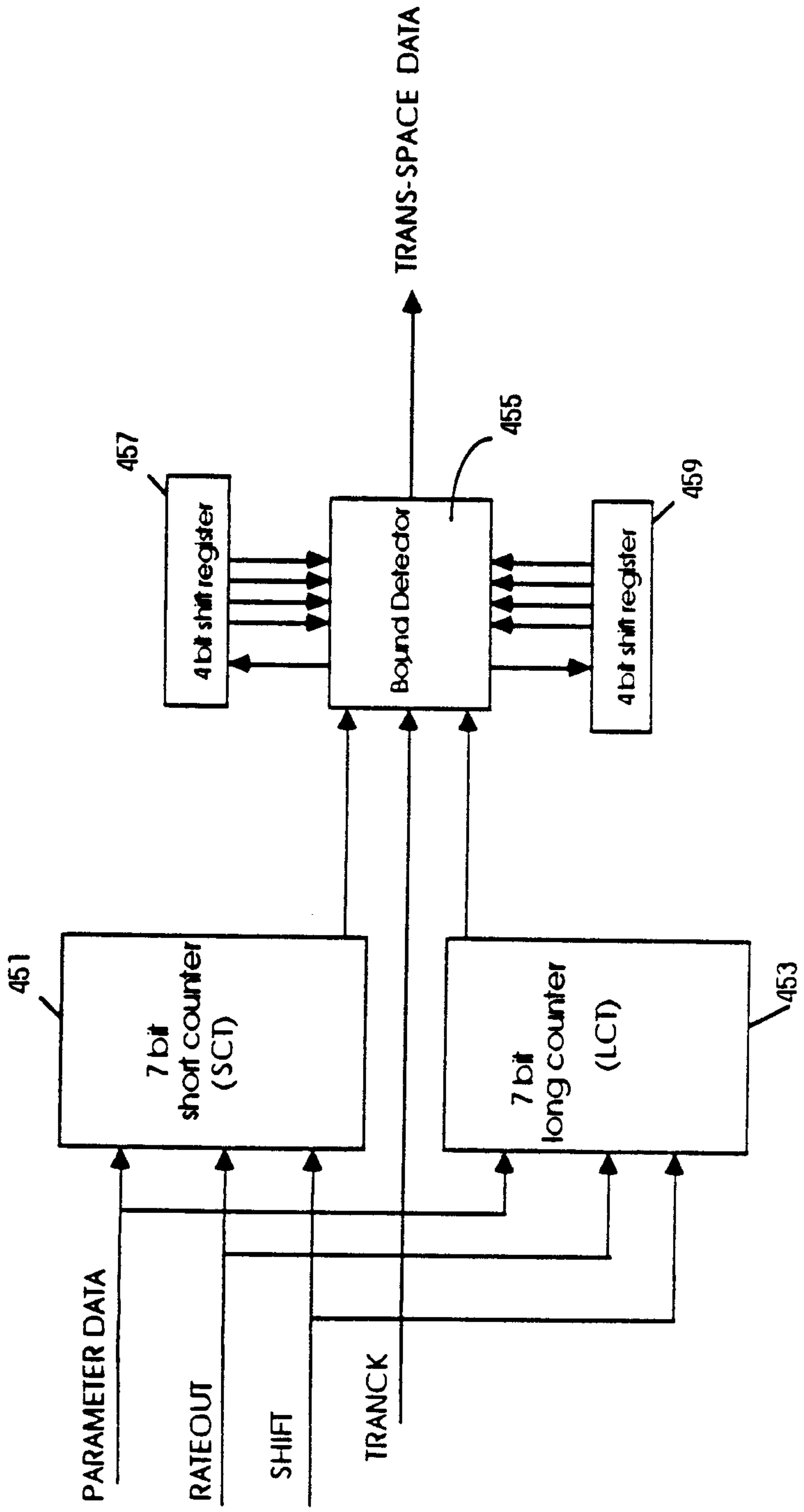

TRANS-SPACE DATA
457
455
459
4 bit shift register
Bound Detector
4 bit shift register
451
7 bit
short counter
(SCT)
7 bit
long counter
(LCT)
453
PARAMETER DATA
RATEOUT
SHIFT
TRANCK

FIG. 5

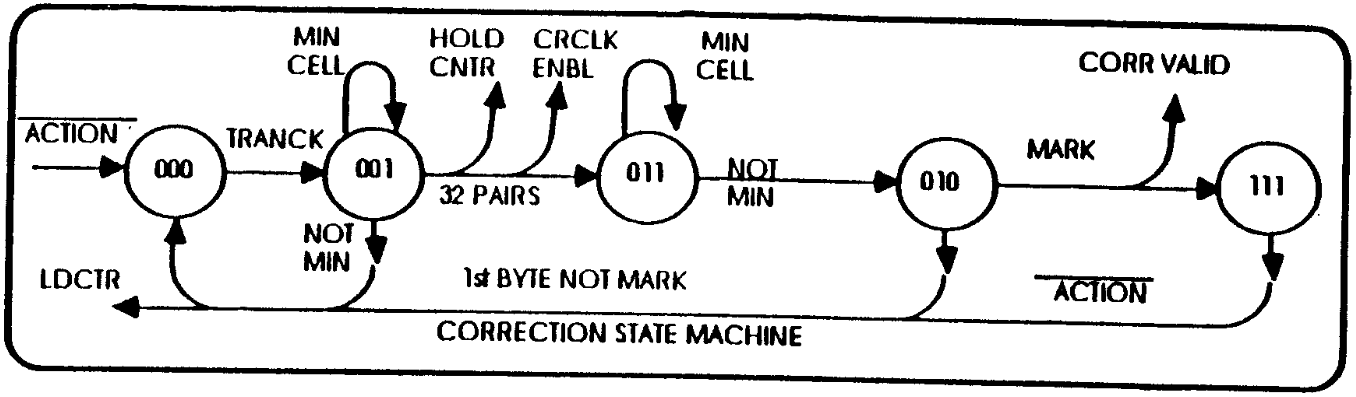

ACTION
000
TRANCK
MIN CELL
001
NOT MIN
HOLD CNTR
CRCLK ENBL
32 PAIRS
MIN CELL
011
NOT MIN
010
MARK
CORR VALID
111
LDCTR
1st BYTE NOT MARK
ACTION
CORRECTION STATE MACHINE

FIG. 6

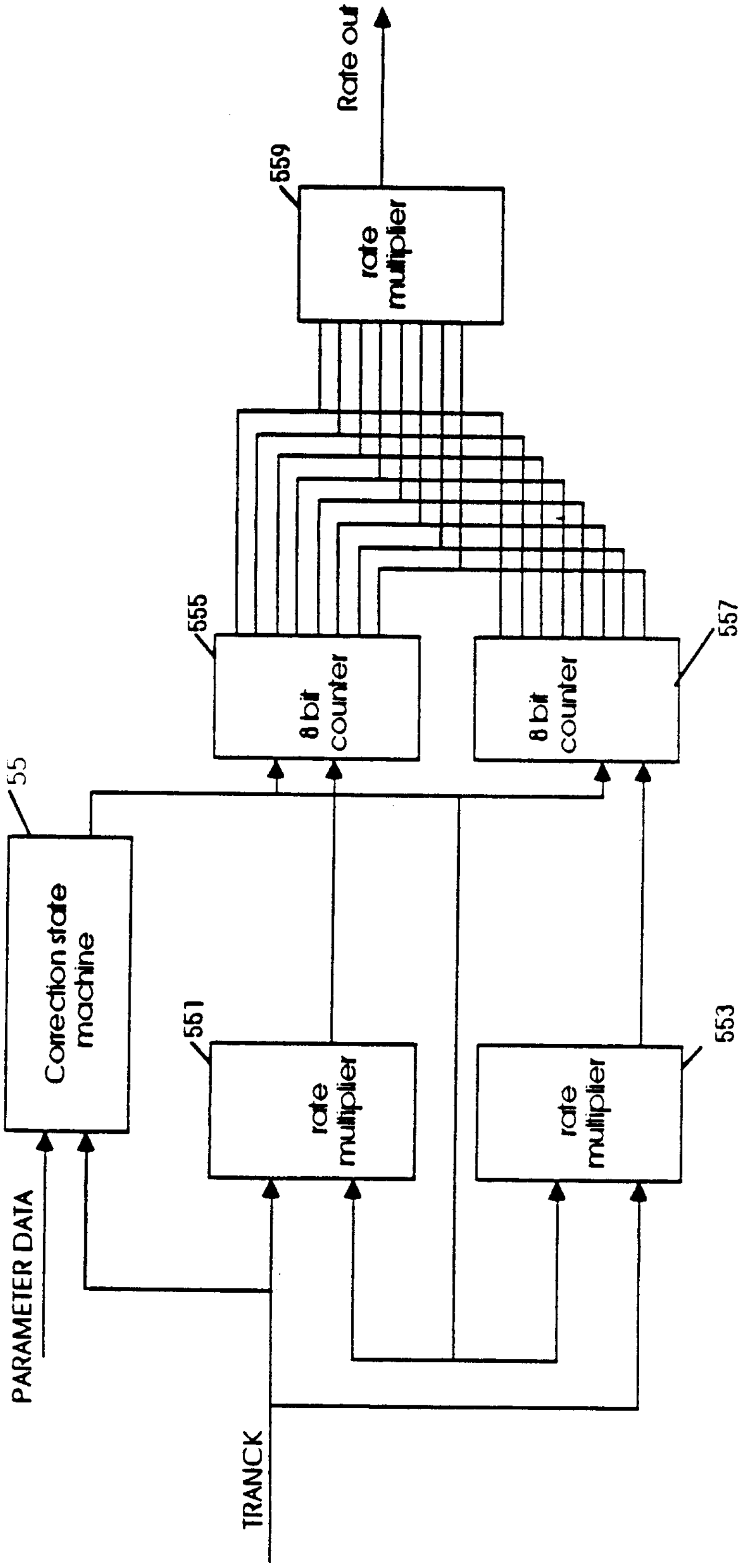

Rate out
559
rate multiplier
555
8 bit counter
557
8 bit counter
55
Correction state machine
551
rate multiplier
553
rate multiplier
PARAMETER DATA
TRANCK

FIG. 7

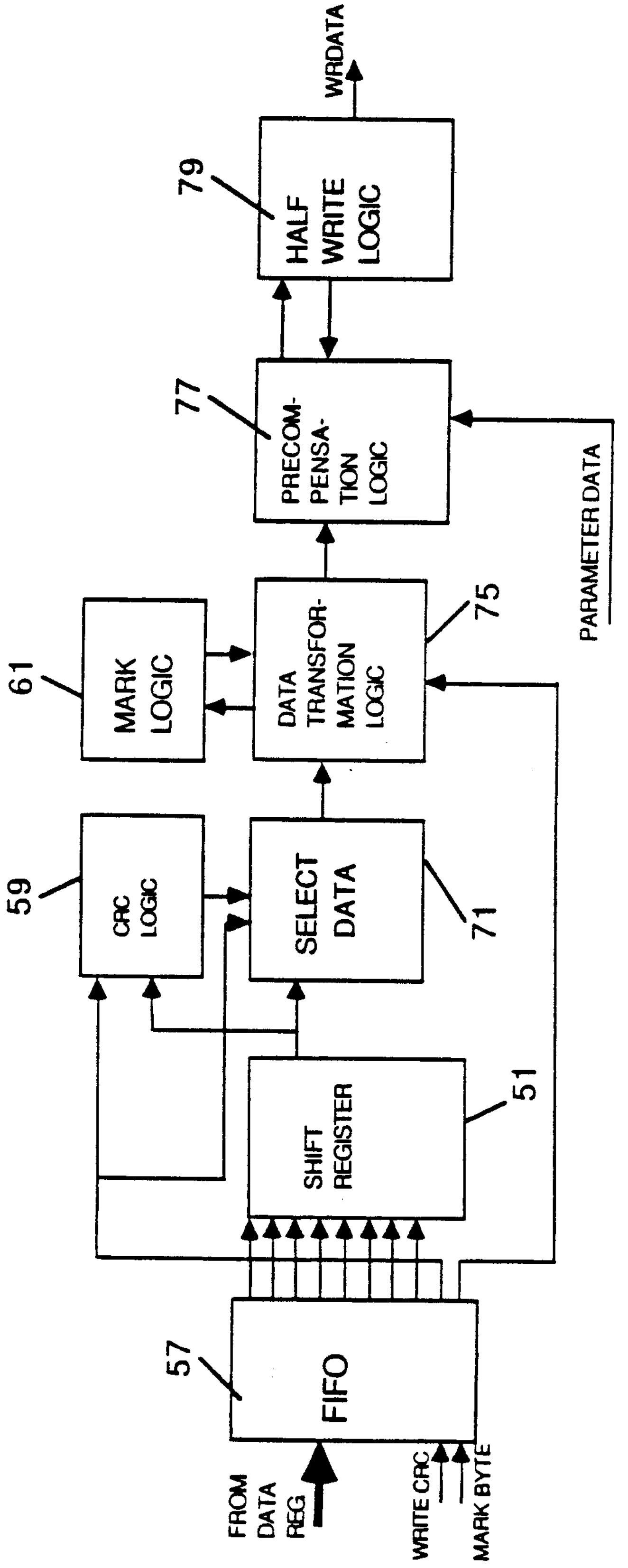

WRDATA
79
HALF WRITE LOGIC
77
PRECOM-PENSA-TION LOGIC
PARAMETER DATA
61
MARK LOGIC
DATA TRANSFOR-MATION LOGIC
75
59
CRC LOGIC
SELECT DATA
71
SHIFT REGISTER
51
57
FIFO
FROM DATA REG
WRITE CRC
MARK BYTE

FIG 8

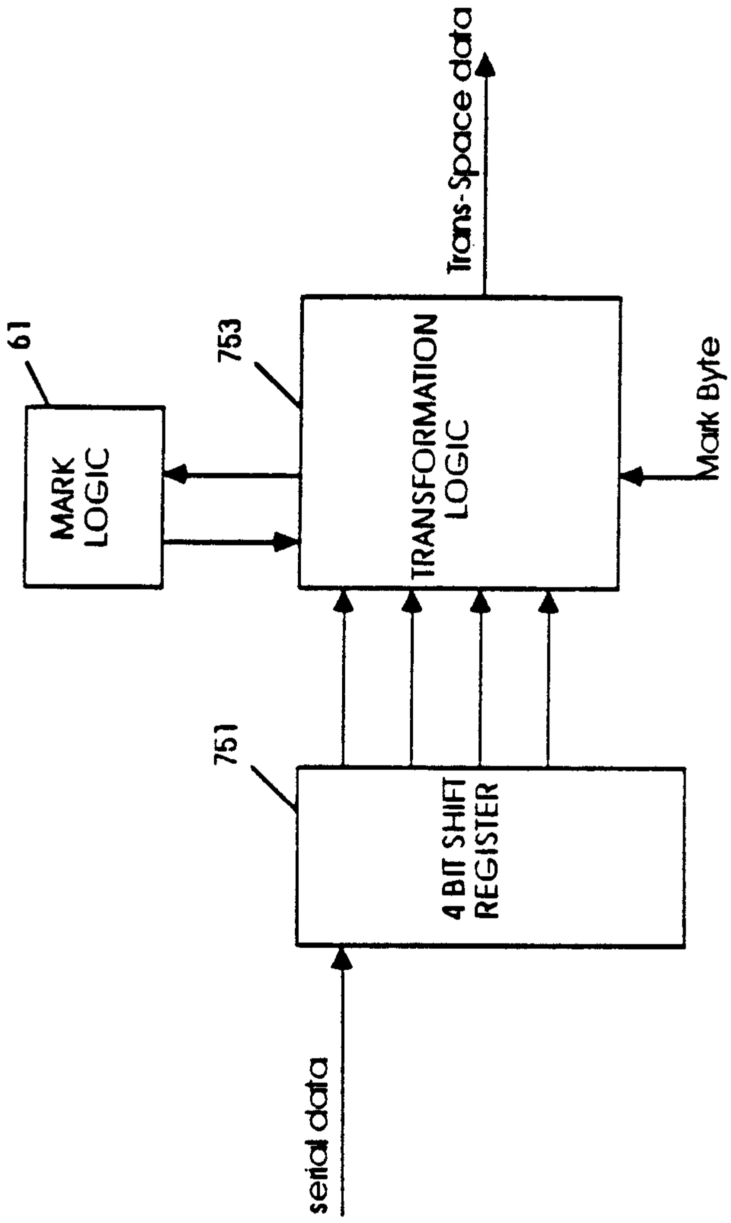

Trans-Space data
61
MARK LOGIC
753
TRANSFORMATION LOGIC
Mark Byte
751
4 BIT SHIFT REGISTER
serial data

FIG. 9

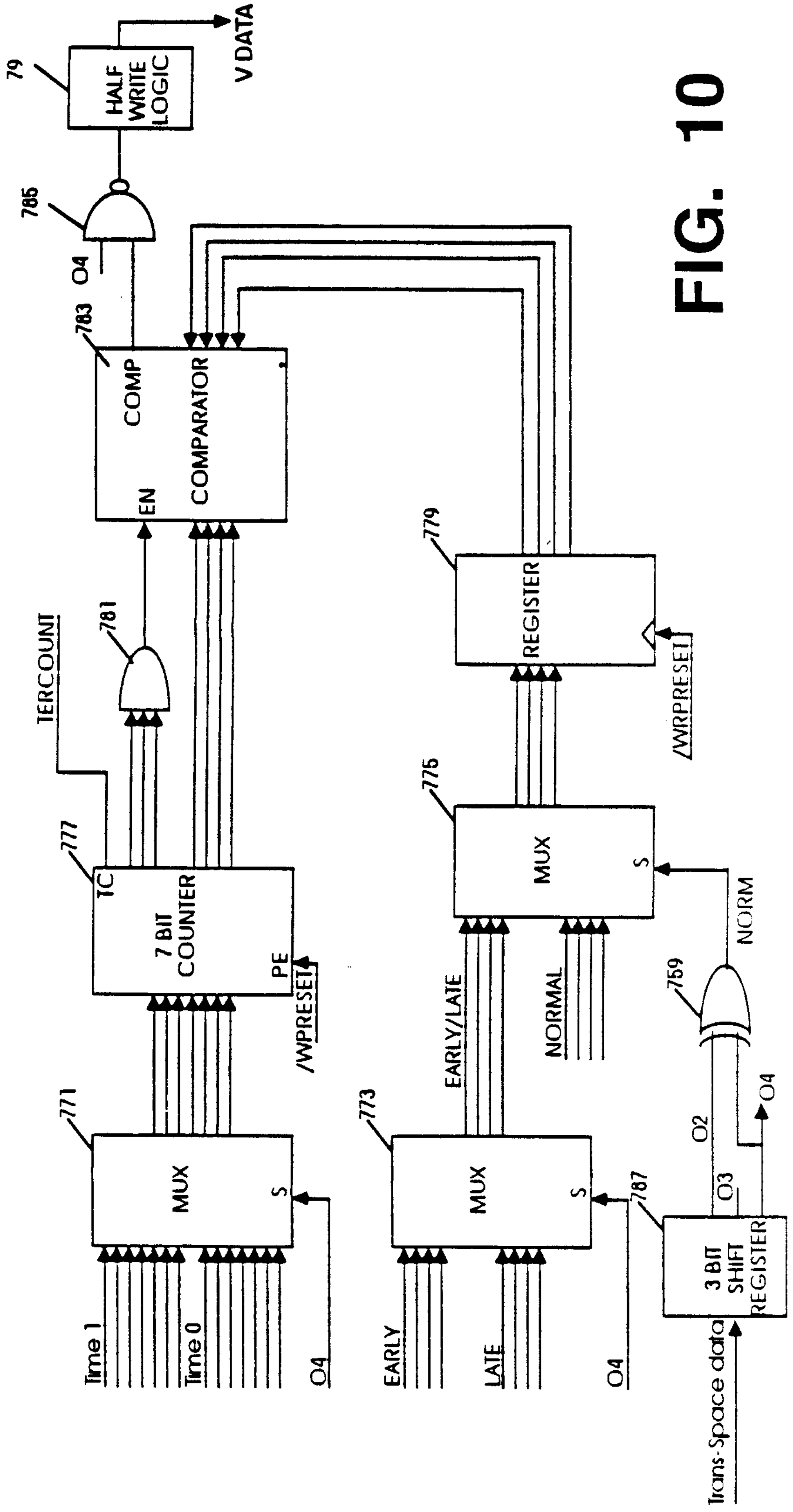

79
HALF WRITE LOGIC
V DATA
785
O4
783
COMP
COMPARATOR
EN
781
TERCOUNT
777
TC
7 BIT COUNTER
PE
/WPRESET
771
MUX
S
Time 1
Time 0
O4
773
MUX
S
EARLY
LATE
O4
EARLY/LATE
NORMAL
775
MUX
S
NORM
759
O2
O4
O3
787
3 BIT SHIFT REGISTER
Trans-Space data
779
REGISTER
/WRPRESET

FIG. 10

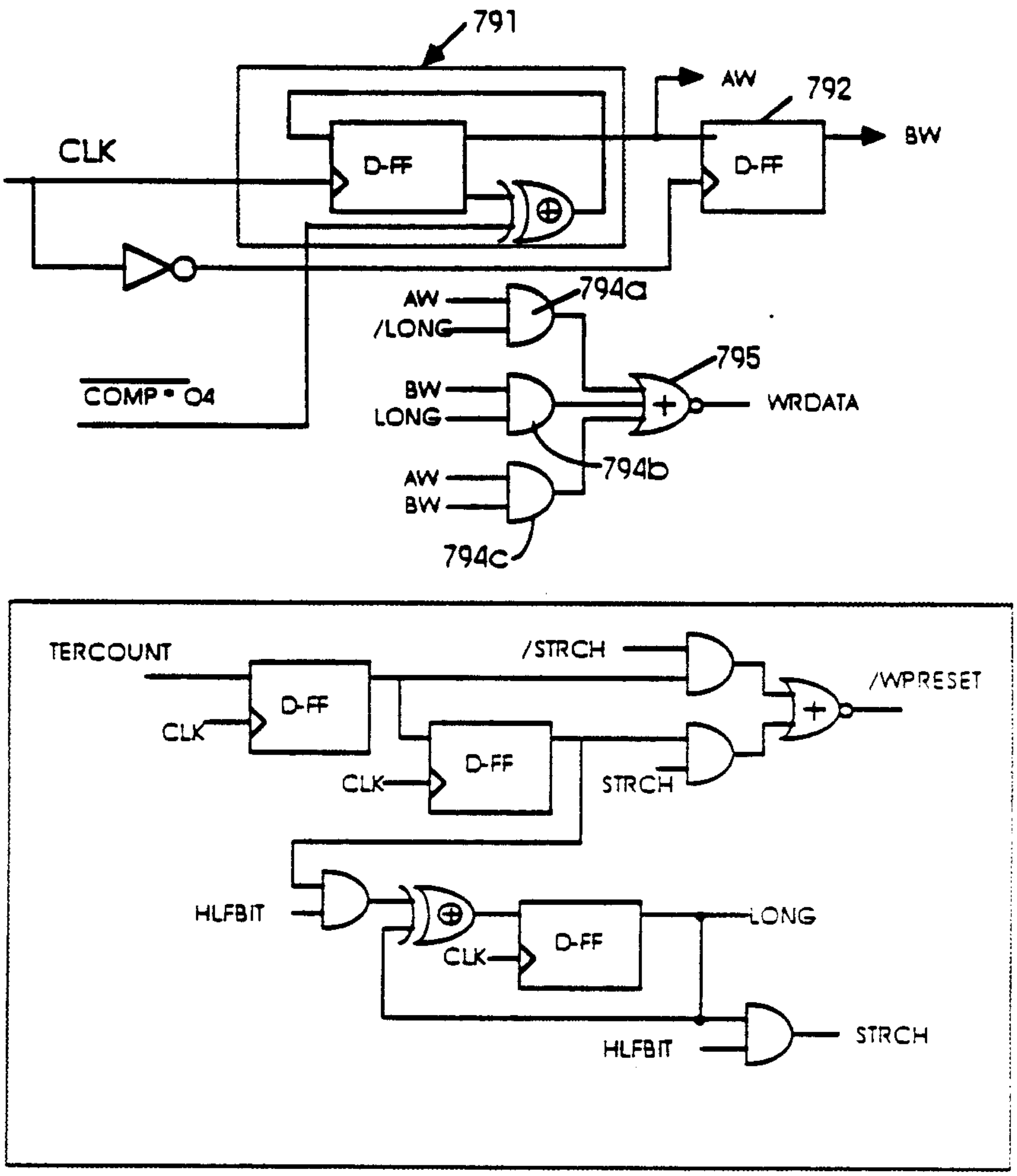

791
AW
792
BW
CLK
D-FF
D-FF
794a
AW
/LONG
$\overline{\text{COMP} \cdot \text{O4}}$
BW
LONG
795
WRDATA
794b
AW
BW
794c
TERCOUNT
/STRCH
/WPRESET
D-FF
CLK
D-FF
CLK
STRCH
HLFBIT
LONG
CLK
D-FF
HLFBIT
STRCH

FIG. 11

DISK DRIVE CONTROLLER

This is a division of application Ser. No. 055,443, filed May 28, 1987.

SUMMARY OF THE INVENTION

An integrated disk controller chip is disclosed which is designed to read and write Manchester ("MFM") and Group Code Recording ("GCR") formatted disks and other formats under program control.

The invented controller uses a programmable parameter scheme which makes it possible to read and write 3½ inch variable and fixed speed drives, as well as standard 5¼ inch drives.

Thus, with the present invention, it is possible to read and write both MFM formatted disks, such as used by IBM personal computers and GCR formatted disks, such as used by Apple personal computers on the same disk drive. It is also possible to write MFM format on a 3½ inch variable speed drive in such a way that it can be read back on fixed speed 3½ inch drives.

The invented controller provides the ability to perform write precompensation to correct for peak shift effects which occur in magnetically stored media.

Also provided is a form of read post compensation which corrects for peak shift effects on disks with insufficient precompensation. A two byte read and write FIFO is used to provide software flexibility.

The invented controller allows the phase lines to be programmed as either inputs or outputs which makes it possible to interface with a wide variety of drives. Additionally, rather than using a fixed rate multiplier, as frequently employed in prior art controllers, the present invention uses a plus/minus rate multiplier to correct for symmetry and frequency errors. Other advanced features of the present invention include the use of half clock circuits to provide half clock resolution in the signal being written to disk and the capability of operating at continuously variable clock speeds and data rates dynamically programmable by the computer.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** shows a particular pattern of 1's and 0's in MFM format.

FIG. **2** is an overview block diagram of the invented controller.

FIG. **3** is an overview block diagram of read logic **21**.

FIG. **4*a*** is logic diagram showing a portion of half read logic **41**.

FIG. **4*b*** is a graphical representation of the signals generated by half read logic **41**.

FIG. **4*c*** is a graphical representation illustrating particular examples of when the signal SHIFT is generated.

FIG. **5** is a detailed block diagram of post compensation logic **45**.

FIG. **6** is a state machine diagram of correction state machine **55**.

FIG. **7** is a detailed block diagram of error correction logic **56**.

FIG. **8** is an overview block diagram of write logic **27**.

FIG. **9** is a block diagram of write data transformation logic **75**.

FIG. **10** is a detailed block diagram of precompensation logic **77**.

FIG. **11** is a detailed block diagram of half write logic **79**.

DETAILED DESCRIPTION OF THE INVENTION

Although the present invention uses various codes for reading from and writing data to disks, it will be described with reference to the most frequently utilized coding scheme, namely Manchester or MFM code. The MFM code follows two basic rules: first, a transition occurs any time that a one is encountered in the data pattern; and second, a transition occurs between any two adjacent zeroes. As shown in FIG. **1**, MFM code produces a series of 2, 3 and 4 unit distances (cells) between transitions which, based on the these distances, when read back, can be resolved into the actual data represented. Details regarding the reading and writing of GCR formatted disks may be found in U.S. Pat. No. 4,210,959 and copending application Ser. No. 943,839.

In the following description, numerous specific details are set forth such as specific word or byte lengths, etc., to provide a thorough understanding of the present invention. However, it will be obvious to one skilled in the art that the present invention may be practiced without such specific details. In other instances, well known circuits have been shown in block diagram form in order not to obscure the present invention in unnecessary detail.

MFM Sector Format

The concept of writing 2, 3 and 4 unit cells provides the mechanism by which the data is translated and written on the disk. But there must be some method for organizing the data so that a specific group of data can be easily located. This is done by writing the data in a sector format. A sector consists of (1) information which allows a controller to find the start of the sector, (2) details about which sector is being read, (3) which side of the disk is being read, (4) which track is being read (a track is a group of sectors), (5) the length of the sector, and (6) cyclical redundancy check (CRC) error detection information. Table 1 shows the organization of an MFM sector.

TABLE 1

	NO. OF BYTES	DATA WRITTEN
	*80	4E
	*12	00
TRACK ID	*3	C2 (Mark Byte)
	*1	FC (Index Mark)
	*50	4E
	12	00
	3	A1 (Mark Byte)
	1	FE (ID Address Mark)
SECTOR ID	1	TRACK NUMBER
	1	SIDE NUMBER
	1	SECTOR NUMBER
	1	SECTOR LENGTH
	2	CRC INFORMATION
	22	4E
	12	00
	3	A1 (Mark Byte)
DATA FIELD	1	FB (Data Address Mark)
	256	DATA
	2	CRC INFORMATION
	54	4E
	**1-	4E

*These bytes are only written at the beginning of a track.

**These bytes are only written at the end of a track.

Note: The sector ID and data field bytes are repeated for each sector on a track.

The beginning of a track or sector consists of a number of bytes of 4E's (hexidecimal) which serve as a buffer zone between regions of meaningful information. The next bytes in the pattern that are written are twelve bytes of zeroes (2 unit cells), known as the "bytes of zeroes". These bytes are used to locate the beginning of either a track, a sector ID or a sector data field. Following the bytes of zeroes are three mark bytes. A mark byte is a special byte containing a pattern which violates the basic rules of MFM (i.e., has a missing transition). This illegal pattern can be recognized, and provides two very important functions: first, since it is always in the byte that follows the bytes of zeroes, it serves as verification that the zeroes are indeed the beginning of a track, sector ID or sector data field and not data (1's and 0's) in a data field and second, the mark byte provides a reference point or synchronization from which the MFM rules may be applied to decode the data. (Without synchronizing on a known pattern, it is impossible to tell the difference between a string of 1's and a string of 0's.) After the mark byte, the next byte encountered in the format pattern is the information byte. This byte is used to determine whether the region being read is the track information, the sector ID, or the sector data field. The next four bytes in the sector ID contain the track number, side number, sector number and sector length.

The next bytes are the cyclical redundancy checks or CRC bytes which are used to detect errors according to well-known techniques.

With the basic concepts of the MFM pattern and MFM sectors in mind, the functions and structure of the invented controller will now be set forth, namely how it handles the problems of reading, writing and interfacing with a processor.

An overview block level diagram of the invented controller is shown in FIG. 2. Controller **11** comprises register block **15** which serves as an interface between the processor and the controller logic; interface logic **16** which serves as an interface between the controller and one or more disk drives; clock logic **17**, which generates a signal TCLK used by the controller from the signal FCLK generated by the processor; read logic **21**; FIFO, CRC and mark logic **24**; write logic **27**; and parameter RAM **31**. The device select signal $\overline{DEV}$ must be asserted by the processor in order for the controller to utilize the signals on address lines A0–A3 and data lines D0–D7. The controller is reset whenever the processor asserts $\overline{RESET}$.

Register Block **15**

Register block **15** comprises registers which may be accessed by the processor and by the controller logic. Some of the registers are read only, some are write only and some are read/write. In this connection, from a software point of view, there are a total of sixteen eight bit registers as follows: data register (read/write); mark register (read/write); error register (read); write CRC register (write); parameter data register (read/write); phase register (read/write); setup register (read/write); handshake register (read); mode register zeroes (write); mode register ones (write); and read status register (read).

Data Register

The data register is the location where data is read from or written to a FIFO in FIFO, CRC and mark logic **24**. If a mark byte is read from this location, an error will occur. A read from this location when Action (data bit **3** in the mode register) is not set, will provide two bytes of error correction information. The register is set up to toggle between the two bytes on successive reads, thus providing both bytes of information. If there is still valid data to be read when Action is not set, it can be read by reading the mark register.

Mark Register

This location is used for reading and writing mark bytes. Writing to this location will cause the missing transition between two zeroes to occur. Reading from this location will allow a mark byte to be read without causing an error.

Error Register

This location provides information on the type of error that has occurred. If any of its bits are set, an error flag will be set in the handshake register as described below. Once any error bit has been set, no other error bit can be set until the register is cleared. Reading the error register will cause the register to clear. This register must be cleared prior to beginning a read or write operation. The possible error conditions are as follows.

In write mode, when bit **0**=**1**, the FIFO is being underrun by the processor. In other words, the FIFO is empty and the processor has not acknowledged the handshake by writing another byte. In read mode, when bit **0**=**1**, the FIFO has two bytes to be read, but the processor is not reading them fast enough.

When bit **1**=**1**, a byte which was read from the data register was a mark byte.

In write mode, when bit **2**=**1**, the processor is writing faster than the FIFO is requesting bytes. In read mode, when bit **2**=**1**, the processor is reading bytes faster than they are available.

When bit **3**=**1**, the correction number obtained in the correction state machine (described in conjunction with FIG. **5** hereinbelow) is so large that the error cannot be corrected.

When bit **4**=**1**, the transition occurred before the first short counter (SCT) pulse (described in conjunction with FIG. **4** below) which indicates that the cell was too narrow to be a legal cell.

When bit **5**=**1**, the fourth SCT pulse occurred before the transition which implies that the transition was too wide to be a valid cell.

When bit **6**=**1**, there were three marginal transitions in a row which implies that the transitions cannot be resolved.

Bit 7 of the error register is not used.

Write CRC Register

A write to the CRC register will set a status bit in the FIFO which will cause the CRC bytes to be written on the disk.

Since the status bit moves through the FIFO, the CRC bytes will shift out after the last bit of data is written.

Parameter Data Register

The parameter data register is where sixteen bytes of parameter data from parameter RAM **31** are written and read. This register comprises a counter which increments the address parameter RAM **31** each time a write or read to the register occurs. The sixteen bytes of data can be written or read by successively writing to or reading from this register. Thus, the four bit address placed on parameter address line **30** accesses sixteen locations in RAM **31** and the data from the accessed location is placed on the eight bit parameter data bus **32**. The increment counter presets the addresses to zero each time a write to the mode register zeros occurs. The data is stored in RAM **31** in the following sequence (the meanings of the various parameters will be set forth below):

RAM Address	Parameter
0000	MIN CELL TIME (MIN)
0001	CORRECTION MULTIPLIER (MULT(K))
0010	SSL
0011	SSS
0100	SLL
0101	SLS
0110	RPT
0111	CSLS
1000	LSL
1001	LSS
1010	LLL
1011	LLS
1100	EARLY/NORM
1101	TIME0
1110	LATE/NORM
1111	TIME1

The MIN parameter is the minimum number of clocks needed to determine a valid transition.

The MULT(K) parameter is a weighting factor for normalizing drive speed to an ideal speed. The SSL, SLL, SLS, RPT, CSLS, LSL, LSS, LLL and LLS parameters are eight bit fields used during post compensation. The EARLY/NORM and LATE/NORM parameters are eight bit fields used during precompensation (four bits for each of EARLY, LATE and NORM.) TIME1 is an eight bit field containing the time delay associated with a transition sent to the drive. TIME0 is an eight bit field containing the additional time delay associated without sending a transition to the drive. TIME1 and TIME0 are 7 bits long. The low order bit of each (HLFBIT) is used by the half write logic, to lengthen WRDATA by one-half clock when desired.

Each of the foregoing parameters is dynamically programmable by the computer. In this manner, the controller can be programmed to run at a clock speed and data rate determined by the computer. Such programmable parameters enable the controller to interchangeably read and write constant angular velocity drives and constant linear velocity drives.

Phase Register

The phase register is used to read and write the four phase lines (phase 0, phase 1, phase 2 and phase 3) which are used to control or read status from the disk drive. The four phase lines can be independently programmed as either inputs or outputs depending on the state of the other four bits in the register. The phase lines default to low outputs on reset. The function of each of the eight bits in the phase register is as follows:

Bit 0 is used to set the polarity of the phase 0 line when programmed as an output.

Bit 1 is used to set the polarity of the phase 1 line when programmed as an output.

Bit 2 is used to set the polarity of the phase 2 line when programmed as an output.

Bit 3 is used to set the polarity of the phase 3 line when programmed as an output.

Bit 4=0 indicates that the phase 0 line is an input.

Bit 4=1 indicates that the phase 0 line is an output.

Bit 5=0 indicates that the phase 1 line is an input.

Bit 5=1 indicates that the phase 1 line is an output.

Bit 6=0 indicates that the phase 2 line is an input.

Bit 6=1 indicates that the phase 2 line is an output.

Bit 7=0 indicates that the phase 3 line is an input.

Bit 7=1 indicates that the phase 3 line is an output.

Setup Register

The setup register is used to set the controller into its various modes. This register will reset to all zeroes when a reset occurs. The function of each of the eight bits in the register is as follows:

Bit 0=1 will cause HEDSEL to be output

Bit 1=0 3.5 inch drive not selected

Bit 1=1 3.5 inch drive selected

Bit 2=0 normal operation.

Bit 2=1 sets the controller into GCR mode.

Bit 3=0 normal operation.

Bit 3=1 causes the internal clock frequency to be divided by two.

Bit 4=0 disables the correction state machine.

Bit 4=1 enables the correction state machine.

Bit 5=0 sets up the read and write signals for Apple type drives.

Bit 5=1 sets up the read and write signals for IBM type drives.

Bit 6=0 normal operation.

Bit 6=1 causes the read and write data transformation logic (described below) to be bypassed. This bit must be set whenever the GCR or 3.5 inch drive modes are set.

Bit 7=0 will produce no timeout when turning off Motoron (mode register, bit 7).

Bit 7=1 causes the Motoron bit to stay on for ½ second (at 16 Mhz) after the drive is disabled.

Handshake Register

The handshake register performs the following functions.

When bit 0=1 the next byte to be read from the FIFO is a mark byte.

When bit 1=0, the CRC register became all zeroes when the second CRC byte passed through the register. This bit is valid when the second CRC byte is the next to be read from the FIFO.

Bit 2 is used to read the read data signal from the drive.

Bit 3 is used to read the SENSE input from the drive.

Bit 4 is used to read the status of Motoron (Mode Register bit 7.

Bit 5=1, indicates one of the bits in the error register has been set to a one. This bit is cleared by reading the error register.

When bit 6=1, in write mode, there are two bytes of available space in the FIFO. In read mode, when bit 6=1, there are two bytes to be read from the FIFO.

When bit 7=1, in write mode, there is one byte of available space in the FIFO. In read mode, when bit 7=1, there is one byte to be read from the FIFO.

Mode Register (Write Zeroes and Write Ones)

The mode register is used to set the various status bits of the controller. A bit can be set to zero by writing to the Write Zeroes location with the corresponding bit set to a one. A bit can be set to a one by writing to the Write Ones location with the corresponding bit set to a one. This scheme is used in order to make it possible to modify a particular bit without having to rewrite the entire register. The register is cleared to zeroes when a reset occurs. The Action bit (bit 3) will be cleared anytime there is any error while writing.

Bit 0 is used to clear the FIFO. This bit must be set and then cleared on successive operations. Read or Write mode (bit 4) must be established prior to setting bit 0 since the FIFO will clear to opposite states depending upon whether a write or read operation is about to take place.

When bit 1=0, drive 1 is not enabled.

When bit 1=1, drive 1 is enabled.

When bit 2=0, drive 2 is not enabled.

When bit 2=1, drive 2 is enabled.

When bit 3=0, Action is not set.

When bit 3=1, Action is set.

Bit 3 is used to start the read and write operation. This bit should only be set after everything else has been setup. When writing, two bytes of data should be written into the FIFO prior to setting this bit in order for the FIFO to start shifting immediately.

When bit 4=0, the controller is placed into Read mode.

When bit 4=1, the controller is placed into Write mode. ·

When bit 5=0, the side 0 head is selected (HEDSEL is reset.)

When bit 5=1, the side 1 head is selected (HEDSEL is set.)

Bit 6 is not used and always reads back as set.

When bit 7=0, Motoron is disabled.

When bit 7=1, $\overline{\text{Enable1}}$ and $\overline{\text{Enable2}}$ signals are asserted, for enabling drive 1 and drive 2. This bit must not be cleared until after the Action bit is cleared.

Read Status Register

This register is used to read back the status of the mode register.

The registers in register block 15 communicate with the other blocks in controller 11 by signals on the various STATUS (for inputs) and CONTROL lines (for outputs), as will be set forth in detail below.

Interface Logic 16

The registers in register block 15 communicate with the drive by signals on the STATUS lines (for inputs) and CONTROL lines (for outputs) using conventional and well known techniques.

Clock Logic Block 17

The inputs to clock logic block 17 are the system clock signal FCLK from the processor which typically is a 7-24 Mhz clock and a signal from register block 15 which causes the clock to run at its full speed or half speed (bit 3 of the Setup Register). Clock logic block 17 outputs the clock signal TCLK which is used by the invented controller. Thus, TCLK is either FCLK or one-half of FCLK.

Read Logic Block 21

FIG. 3 is an overview block diagram of read logic 21, including the applicable portions of FIFO, CRC and mark logic block 24 which are shared with write logic block 27.

Data is read from a disk by means of a signal called RDDATA generated by the drive as the read head passes over the magnetic media. This signal consists of pulses which are spaced at 2, 3 and 4 units apart, which of course is the data in its MFM translated form. If all conditions were ideal, to convert the MFM formatted data into its actual data, it would be a relatively simple matter to determine whether a cell is 2, 3, or 4 units long, then decode the data, and transfer the data through a serial to parallel shift register for use by the processor. However, conditions are rarely, if ever, ideal. A first problem is known as peak shift which occurs due to the non-ideal nature of the properties of magnetic media. Specifically, it is known that a 2 unit cell on a disk is crowded together more than a 3 or a 4 unit cell, in a relative sense. The effect of this crowding is that 2 unit cells will tend to push out their transitions into the region of a 3 or 4 unit cell, when a 2 unit cell is adjacent to a 3 or 4 unit cell. This pushing out causes such a 2 unit cell to be longer than it should be, and a 3 or 4 unit cell to be shorter than it should be when the data is read back.

When the data is written, it is known in the art to use a technique known as precompensation to correct for this problem, wherein a transition is caused to occur earlier or later when writing. That is, precompensation makes 4 and 3 unit cells longer and 2 unit cells shorter when they are next to each other during disk writes.

However, if the disk that is being read was not written by a controller which uses precompensation, or the precompensation used was not enough, errors may occur reading back the data due to effects of peak shift. This problem is solved in the present invention by using post compensation which will be described in detail below. Other problems that can occur are that the speed of the disk drive or the frequency of the clock can be off, or there can be some other form of systematic error in the data. Such errors can also make it very difficult to read back the data reliably. Such errors are corrected in the present invention by use of a correction state machine. The discussion of the read logic will set forth how the post compensation and correction state machine work, along with a description of how the beginning of a track or sector is located, how the mark byte is detected, and what starts the process of transferring data into the FIFO.

Read logic block 21 comprises half read logic 41, post compensation logic 45, data transformation state machine 49, shift register 51, correction state machine 55 and error correction logic 56. Also shown in FIG. 3 are FIFO 57, CRC logic 59 and mark logic 61, which elements are from FIFO, CRC and mark logic block 24, as shown in FIG. 2.

Half Read Logic 41

Half read logic 41 causes 2 unit cell, 3 unit cell and 4 unit cell input signal RDDATA which is asynchronous with respect to the internal clock TCLK to become synchronous with TCLK and transformed so that each RDDATA pulse is precisely one TCLK wide. The synchronized and transformed output is referred to herein as TRANCK.

In particular, half read logic 41 detects whether a RDDATA pulse occurred in the first or second half of the clock cycle thereby providing half clock resolution cf the input pulse. Depending on the combination of which half of the clock cycle the current RDDATA pulse occurred in, and in which half the previous RDDATA pulse occurred in, there might have been an error in resolving RDDATA into TRANCK. Thus, the half read logic will stretch the bounds which are determining the cell time by one clock. This will effectively shorten the distance between TRANCK pulses by one clock, thereby correcting for the error in the one clock sample time.

If the cell times of the data coming from drive are very accurate, there is no problem resolving the data because the parameters can be set to fit in the middle of each region and there is sufficient margin between the SCT and LCT pulses generated by SCT and LCT counters (described below with the description of FIG. 5) and TRANCK pulses. However, in reality due to drive and noise error there can be some error in the values of the cell times. This can cause the SCT and LCT pulses and the TRANCK pulses to fall very close to each other making it difficult to tell the difference between two different cell times.

Without halfclock resolution, what is intended to be 2 3 4 pattern can be transformed into a 3 3 3 pattern. Such error can occur since data can only be sampled on the rising edge of the clock. Thus, if a first RDDATA pulse occurs just after the rising edge of the clock and a second RDDATA pulse occurs just prior to the rising edge of the clock, almost one full clock of error has been introduced in the length of the cell. This problem can be reduced by determining which half of the clock cycle the RDDATA pulse occurred in and shifting the SCT and LCT pulses (as described below) by one count to compensate. Shifting the SCT and LCT pulses will effectively change the distance between TRANCK pulses. The overall effect is that the distance between RDDATA pulses can be resolved to within one half clock of the actual distance instead of one clock. The effective half clock shift of SCT and LCT can take place in two manners. First to compensate for the problem just mentioned and second to allow for better resolution in calculating the parameters for the SCT and LCT counters. FIGS. 4*b* and 4*c* show a schematical representation of how a shift signal used by the counters is generated.

Specifically, FIGS. 4*b* and 4*c* shows that the TRANCK signal is formed such that it is delayed for four clocks. This pipelining is necessary to be able to know when the TRANCK is going to occur four clocks before it occurs. The RDDATA signal is synchronized to the nearest half clock and then delayed by one clock to generate the signal RT3 as shown in FIG. 4*a*, which shows a particular implementation of half read logic **41**. When the TCK signal becomes valid, RT3 is sampled. If RDDATA occurred in the first half of the clock cycle, RT3 would be a one. If RDDATA occurred in the second half of the clock cycle, RT3 would be zero. This information is then latched in as signal called BIAS. The signal BIAS is set to a zero if RDDATA occurred in the first half of the clock cycle, and is set to a one if it occurred in the second half of the clock cycle. The signal NSTART is used to latch BIAS when TRANCK occurs. This is used on the next RDDATA to determine what has just occurred since the BIAS signal will change on the next TCK. As mentioned above, to avoid introducing errors resulting from the asynchronous nature of the clock signal and RDDATA, it must be known, in advance, whether SCT and LCT should or should not be shifted near a TRANCK. This can now be resolved using the information generated. Since it is known when the TRANCK is going to occur four clocks prior to it actually occurring, and it is known which half of the clock cycle the RDDATA pulse that generated the TRANCK occurred in, and the same information about the previous RDDATA pulse is known, a signal called SHIFT can be generated which will cause the comparison point in the SCT and LCT counters **451** and **453** to be altered by one count thereby correcting to the nearest half clock. The equation for generating SHIFT is $\overline{\text{FRACTION}}$ *NSTART*$\overline{\text{BIAS}}$+FRACTION * $\overline{\text{NSTART*BIAS}}$. SHIFT is set with TCK1 and reset with TRANCK. FRACTION is the low order bit of the parameter loaded in each of SCT counter **451** and LCT counter **453**.

Post Compensation Logic **45**

Post compensation logic **45** corrects errors caused by the effects of peak shifting. A detailed block diagram of post compensation logic **45** is shown in FIG. 5.

Post compensation logic comprises two 7-bit counters **451** (SCT) and **453** (LCT), a bound detector **455** and two 4-bit shift registers **457** and **459**. The counters are used to place pulses at certain time intervals between transitions. The presets of these counters are the parameters SSL, SSS, SLL, SLS, RPT, CSLS, LSL, LSS, LLL and LLS which are programmed by the software and enable the controller to handle various cell times. The SCT counter **451** loads parameters which are calculated to represent a cell which has a short cell (i.e. 2 unit) following it. The LCT counter **453** loads parameters which are calculated to represent a cell time which has a long cell (i.e. 3 or 4 unit) following it. Additionally, the parameters loaded depend on the previous cell time. In this connection, the counter parameters SSS, LSS, SLS and LLS are used by the SCT counter and the SSL, LSL, SLL and LLL parameters are used by the LCT counter. (The letters represent Long or Short previous/current/next cell times; e.g., the SSL parameter is used when the previous, and current cell times are short and next cell time is long.) RPT is the maximum number of clocks which may occur before a valid transition. CSLS is an addition correction used by the post-compensation logic under certain conditions. The following describes how the parameters are calculated.

The parameters are calculated based on the clock frequency and cell times. Therefore it is required to know both of these factors before calculating parameters. For calculating post compensation parameters, it is required to know the amount of peak shift. This factor can be expressed as a percentage of the minimum cell time MIN. The first step in calculating the parameters is to determine the number of clocks (Nclks) for each of the three cell times. This is done as follows:

Nclks=length of cell (in s) * clock frequency (in Mhz).

The three different cell times will be defined as Nclk1, Nclk2 and Nclk3. The MIN parameter is defined to be the minimum value that a cell must be. This value is arbitrarily placed at the midpoint between between zero and the first transition time. Therefore,

MIN=Nclk1/2

The rest of the parameters are calculated in a similar fashion such that the bounds will be placed at the midpoint between two cell times. The only difference is that there is a different amount of peak shift for different combinations of cell times next to each other making it necessary to compensate differently for each. The amount of peak shift per edge can be calculated as follows:

peak shift = PS = % peak shift (per minimum cell time)
*Nclk1

This number represents the number of clocks that an edge is affected if a 2 unit cell is next to a 3 or 4 unit cell or vice versa. With this in mind, the remaining parameters can be calculated as follows:

SSS = (Nclk1 – Nclk2)/2 – INT(MIN) – PS
SSL = (Nclk1 – Nclk2)/2 – INT(MIN)
LSS = (Nclk1 – Nclk2)/2 – INT(MIN)
LSL = (Nclk1 – Nclk2)/2 – INT(MIN) – PS
SLS = (Nclk2 – Nclk3)/2 – INT(SSS) – 2*PS
SLL = (Nclk2 – Nclk3)/2 – INT(SSL) – PS
LLS = (Nclk2 – Nclk3) 2 – INT(LSL) – PS
LLL = (Nclk2 – Nclk3)/2 – INT(LSL)

-continued

CSLS = SLL − INT(LSL)

The RPT parameter is simply a maximum bound check. Therefore, its value is not constrained to a particular value, but it must meet the following requirement:

$$RPT = >(Nclk3 - Nclk2) + 2^*PS$$

These values must be converted to hexidecimal (Hex) since they represent presets to binary counters. This is done by rounding each value to the nearest half and converting the integer portion into its Hex equivalent value. This value is mapped into the upper 7 bits of the corresponding 8 bit parameter. The low order bit (FRACTION) is set to a one if the fractional part of the number is one-half, otherwise it is set to a zero.

The use of the parameters will now be described with reference to a particular example.

Assume:

Fclk = 16 Mhz.

Cell times are 4, 6 and 8 s.

Post Comp = 3% of 4 s cell time.

This implies:

NCLK1 = 4 * 16 = 64 Clocks

NCLK2 = 6 * 16 = 96 Clocks

NCLK3 = 8 * 16 = 128 Clocks

PS = 3% * 64 Clocks = 1.92 Clocks

Therefore the parameters are:

MIN = 64/2 =	32.00 Clocks
SSS = (64 + 96)/2 − 32 − 1.92 =	46.08 Clocks
SSL = (64 + 96)/2 − 32 =	48.00 Clocks
LSS = (64 + 96)/2 − 32 =	48.00 Clocks
LSL = (64 + 96)/2 − 32 + 1.92 =	49.92 Clocks
SLS = (96 + 128)/2 − 46 2*1.92 =	62.16 Clocks
SLL = (96 + 128)/2 − 48 − 1.92 =	62.08 Clocks
LLS = (96 + 128)/2 − 48 − 1.92 =	62.08 Clocks
LLL = (96 + 128)/2 − 49 =	63.00 Clocks
RPT = 128 − 96 + 2*1.92 =	35.84 Clocks

Converting these parameters to Hex yields the following:

MIN = $40

SSS = $5C

SSL = $60

LSS = $60

LSL = $64

SLS = $7C

SLL = $7C

LLS = $7C

LLL = $7E

RPT = $48

The other dynamically programmable parameters are calculated as follows:

MULT (K) =	(256*256)/(32*Nclk1)
TIME1 =	Nclk1
TIME0 =	Nclk1 /2
NORM =	Arbitrary
LATE =	NORM + Pre Comp * Nclk1
EARLY =	NORM − Pre Comp * Nclk1

Pre Comp is selectable by the software as a percentage of the MIN cell time.

Bound detector **455** counts the number of pulses which occur between TRANCK transitions. If one pulse occurs between transitions, then the cell must be a two unit cell, if two pulses occur between transitions then the cell must be a three unit cell, and if three pulses occur between transitions then it must be a four unit cell.

The reason for having two counters is that depending on whether the next cell is long (a 3 or 4 unit cell) or short (a 2 unit cell) the pulses may occur in different positions because, for example, a 3 unit cell will be shorter when next to a 2 unit cell than when next to a 3 or 4 unit cell. If both counters generate the same number of pulses between transitions, then bound detector **455** simply generates a space (a 0) for each pulse and a transition (a 1) at the end of the transition time. Such output is referred to herein as the trans-space data pattern. If the two counters generate a different number of pulses between transitions, then the length of the current cell cannot be determined until the next transition time is determined.

Two 4-bit shift registers **457** and **459** keep track of what has happened until the next cell has been determined, thus making it possible to determine the length of the uncertain cell.

Correction State Machine **55**

Correction state machine **55** corrects systematic errors such as those caused by a drive that runs too fast or too slow or by an inaccurate clock. A state machine diagram of correction state machine **55** is shown in FIG. **6**.

In MFM format, the beginning of a sector or track can be located by finding the 12 bytes of zeros followed by the mark byte. In the present invention, correction state machine **55** is used to sync-up on the bytes of zeroes followed by the mark byte.

Specifically, the state machine looks for a string of minimum cells by looking at the number of SCT pulses that occur between TRANCK pulses. If the state machine sees 64 cells which have only one SCT pulse between transitions, then it knows that it has found a region of minimum cells. The machine then looks to see if the first non-minimum cell is part of a mark byte. If this is the case then the rest of the bits start shifting into the shift register **51** and FIFO **57** will begin functioning. Otherwise the state machine will go back into the state which looks for a string of minimum cells.

The state diagram of FIG. **6** shows how correction state machine **55** works. It starts out in the **000** state and stays there until it gets a transition. At this point it goes into the **001** state where it stays until it encounters 32 minimum cells. If 32 pairs of minimum cells are then counted, the machine proceeds on to the **010** state, otherwise it goes back to look for another transition. Once it has encountered the 32 pairs, it waits for the first non-minimum transition to occur in state **011**. If this non-minimum cell is part of a mark byte, then it proceeds on to the **111** state where it remains until the processor is finished reading bytes. If the non-minimum cell is not part of a mark byte, the state machine goes back to state **000**.

Error Correction Logic **56**

Referring now to FIG. **7**, during the sync-up period, rate multiplier **551** and **553** count the number of clocks for 32 MIN CELL TIMEs. Upper counter **555** counts the even cells and lower counter **557** counts the odd cells. This make it possible to correct for asymmetry as well as frequency errors. The amount by which the 8-bit counters vary from 256 counts represents the amount of error over the sample. This error number is then applied to post compensation logic SCT counter **451** and LCT counter **453** by stretching or shortening the counts using the output of rate multiplier **559** RATEOUT.

Data Transformation State Machine 49

Data output from post compensation logic **45** is input to read data transformation state machine **49** which converts the data into actual MFM data. Table 2 shows the results of the operation of data transformation state machine **49** for all combinations of trans-space and previous data.

TABLE 2

PREVIOUS DATA	CURRENT TRANS-SPACE	RESULT
1	(1)	1
1	(01)	0
1	(001)	01
0	(1)	0
0	(01)	01
0	(001)	00

The actual data (i.e., after transformation by data transformation state machine **49**) is input to serial-to-parallel shift register **51** which shifts out CRC bytes, mark bytes and data bytes as parallel data. The eight bit actual data, is transferred to FIFO **57** which is a two byte FIFO comprising two 10 bit registers. CRC logic **59** is implemented as the CRC polynomial $X^{16}+X^{12}+X^5+1$. Mark logic **61** is implemented as a state machine which generates a logic 1 when a mark byte is detected.

Write Logic Block 27

FIG. **8** is an overview block diagram of write logic **27**, including the applicable portions of FIFO, CRC and mark logic block **24**.

The following will describe how data from the processor is translated into 2, 3 and 4 unit cells for writing to the disk.

The write process begins when a processor writes a byte into the data register and sets the Action bit in the mode register. The byte which is written in the data register is loaded into FIFO **57**. FIFO **57** is a two byte FIFO consisting of three ten bit registers. The first ten bit register is used to grab the data from the data register and the other two are used as FIFO registers. The ten bit FIFO consists of eight bits of data, a bit which indicates whether the data is a mark byte and a bit which tells the controller to write the CRC bytes.

As shown in FIG. **8**, write logic **27** includes FIFO **57**, shift register **51**, CRC logic **59**, and mark logic **61**. While each of the foregoing components can be separately implemented, in the preferred embodiment of the subject invention, such elements are shared between read logic **21** and write logic **27** as part of FIFO, CRC and mark logic block **24**. Of course, in performing a write, shift register **51** is a parallel to serial shift register rather than a serial to parallel shift register as is the case when doing a read. Similarly, during a write, CRC logic **59** calculates a CRC byte to be written rather than calculating a CRC byte to compare with one which has been read. Similarly, mark logic **61**, when in write mode causes a mark byte to be written.

Select Data 71

The serial data output from shift register **51** is input to select data block **71** which, in effect, multiplexes between the actual data and the CRC byte produced by CRC logic **59**, outputting the data or CRC byte to data transformation logic **75**.

Write Data Transformation Logic 75

Write data transformation logic **75** translates the data stream into a form in which a 1 represents a transition and a 0 represents a space which is the form suitable for writing on a magnetic disk.

A block diagram of write data transformation logic **75** is provided in FIG. **9**. As shown in FIG. **9**, the front end of data transformation logic **75** is a four bit shift register **751** which makes it possible to know what the last two bits were, the current bit is, and the next bit will be. Most of the time, the only information needed is what the current bit is and what the next bit will be. The exception is when writing the mark byte. In this instance, more information is needed because it must be determined when to leave out the transition. As noted above, the only time a transition needs to be skipped is when there is a 1 0 0 0 pattern. Thus, all four bits of information are needed. Table 3 shows the desired transformation of the data performed by transformation logic **753**.

TABLE 3

CURRENT BIT	NEXT BIT	TRANSFORMED DATA	MARK
0	0	1	00
0	1	01	01
1	0	0	0
1	1	1	1

Precompensation Logic 77

Precompensation logic **77** compensates for the problems created by peak shifts as described above with respect to read logic **21**. Precompensation logic **77** performs the analog of post compensation logic **45** for write operations. A detail block diagram of precompensation logic **77** is shown in FIG. **10**.

Precompensation logic **37** comprises multiplexors **771**, **773** and **775**, 7-bit counter **777**, latch register **779**, AND gate **781**, comparator **783**, AND gate **785**, shift register **787** and XOR gate **789**.

The 7-bit counter **777** shown in FIG. **10** is used for counting out the desired cell times. The counter is preset to either TIME1 if a transition is occurring, or TIME0 if a space is occurring. When the counter reaches the value of the comparison number, then the transition is fed to the half-write logic **79** if the high bit of the shift register **787** is a 1 (indicating a transition). By changing the comparison point, it is possible to stretch or shorten a cell time thereby performing precompensation. The decision whether the transition should be early or late is decided by whether a transition is about to take place. The decision whether the transition should be placed at its nominal value or at a corrected value is made by looking at what the next cell is going to be, thus knowing whether the next cell is a similar type cell. Shift register **787** provides the ability to look at what is coming next in order to determine what to do with the current transition. The outputs of shift register **787** are O4 (current data), O3 (next data), and O2 (next, next data).

Inasmuch as the length of cells to be written are not exact multiples of the clock frequency, additional errors may be introduced. For example, in a four microsecond cell, when a 7.16 Mhz clock is used, the number of clocks in the cell is 28.64. While the output from precompensation logic 77 can be used to write to the disk, it is necessary to round the cell length to an integral number of clocks. This round off forces the cell times to vary from the desired values. Depending upon the clock used, it is possible for significant errors to result. In order to reduce this round off effect, the present invention utilizes half write logic 79 which works on both edges of the clock and creates the effect of having half clock resolution. Writing using half clocks can be very difficult because of the high effective clock speeds generated. For this reason, half write logic 79 is performed just prior to writing the data to the disk.

Half Write Logic 79

A detailed block diagram of half write logic 79 is shown in FIG. 11.

Once a comparison point has been reached and the trans-space data is a 1, then a transition is generated by toggling the WRDATA line to the drive. This is done by toggling T-counter 791. This toggle is subsequently delayed by one half clock using D-flip-flop 792. The resulting WRDATA signal is then generated by selecting either the half clock signal BW or non-half clock signal AW thus producing half clock resolution in the WRDATA signal according to the logic performed by and gates 794*a*, 794*b* and 794*c* and NOR gate 795. The HLFBIT signal is what determines whether to cause half clock shifts or not. In particular, logic circuit 796 will cause the LONG signal to toggle on each transition only allowing half-shifts on alternate edges.

What is claimed is:

1. In an improved disk drive controller for controlling the transfer of data between a computer and a disk drive, said computer including a clock for generating clocking signals, an address bus and a data bus, said controller including read logic means for converting data received from a signal generated by the drive to data for placement on the data bus, and write logic means for converting data on the data bus to a signal for recording on magnetic media by the drive, the improvement wherein said read logic means includes means for processing the signal received from the drive to compensate for the effects of peak shift and wherein said peak shift compensation means comprises:

(a) first counter means and second counter means for placing pulses at predetermined time intervals between transitions in said signal from said drive, said predetermined time intervals being determined by setting said first and second counter means with values generated by said computer as a function of the time between previous transitions in said signal from said drive, wherein said first counter means is set with a value corresponding to the shortest expected time between the next two transitions and the second counter means is set to a value greater than the shortest expected time between the next two transitions and less than the maximum expected time between the next two transitions;

(b) bound detector means coupled to said first and second counter means for counting the number of pulses generated by said first and second counter means between transitions in said signal from said drive; and

(c) first and second shift registers coupled to said bound detector means for storing the number of pulses generated by said first and second counters respectively to enable said bound detector means to generate peak shift compensated pulses from said signal from said drive.

2. A method for performing symmetry and frequency correction on a signal from a disk drive, said disk drive for coupling to a controller, said controller for coupling to a computer, said signal having transitions which are converted by the controller into data usable by the computer, said method comprising the steps of:

(a) summing the distances between the leading edges of alternate pairs of said transitions:

(b) normalizing said summed distances;

(c) subtracting said normalized distances from predetermined values to produce a correction magnitude and direction;

(d) using said correction magnitude and direction to generate a pulse to correct the symmetry and frequency of said signal from said disk drive.

* * * * *

United States Patent [19]

Ashkin et al.

[11] **Patent Number: 4,918,598**

[45] **Date of Patent: Apr. 17, 1990**

[54] **METHOD FOR SELECTIVELY ACTIVATING AND DEACTIVATING DEVICES HAVING SAME FIRST ADDRESS AND DIFFERENT EXTENDED ADDRESSES**

[75] Inventors: **Peter B. Ashkin,** Los Gatos; **Michael Clark,** Glendale, both of Calif.

[73] Assignee: **Apple Computer, Inc.,** Cupertino, Calif.

[21] Appl. No.: **132,124**

[22] Filed: **Dec. 14, 1987**

Related U.S. Application Data

[62] Division of Ser. No. 765,396, Aug. 14, 1985, Pat. No. 4,910,655.

[51] **Int. Cl.**^{4} **G06F 13/42**

[52] **U.S. Cl.** **364/200;** 364/229.2; 364/240.8; 364/261; 364/284.3; 340/825.03; 340/825.52

[58] **Field of Search** ... 364/200 MS File, 900 MS File, 364/137, 138, 514; 340/825.02, 825.03, 825.07, 825.52, 825.53; 370/85

[56] **References Cited**

U.S. PATENT DOCUMENTS

3,221,307	11/1965	Manning	364/200
3,646,534	2/1972	Miller	360/40
3,715,725	2/1973	Kievit et al.	340/825.07 X
3,787,627	1/1974	Abramson et al.	370/67
3,836,888	9/1974	Boenke et al.	364/200
3,863,025	1/1975	Gonsewski et al.	375/55
3,979,723	9/1976	Hughes et al.	370/31
4,063,220	12/1972	Metcalfe et al.	340/825.5
4,071,908	1/1978	Brophy et al.	340/825.02
4,345,250	8/1982	Jacobsthal	340/825.5
4,360,870	11/1982	McVey	364/200
4,373,181	2/1983	Chisholm et al.	364/200
4,442,502	4/1984	Friend et al.	364/900
4,498,169	2/1985	Rozmus	370/85
4,562,535	12/1985	Vincent et al.	364/200
4,568,930	2/1986	Livingston et al.	340/825.5
4,570,220	2/1986	Tetrick et al.	364/200
4,589,063	5/1986	Shah et al.	364/200
4,595,921	6/1986	Wang et al.	340/825.08
4,608,559	8/1986	Friedman et al.	340/825.5
4,608,689	8/1986	Sato	371/15
4,611,274	9/1986	Machino et al.	364/200
4,620,278	10/1986	Ellsworth et al.	364/200
4,626,846	12/1986	Parker et al.	340/825.5
4,628,478	12/1986	Henderson, Jr.	364/900
4,638,313	1/1987	Sherwood, Jr. et al.	340/825.52
4,660,141	4/1987	Ceccon et al.	364/200
4,667,193	5/1987	Cotie et al.	340/825.08
4,675,813	6/1987	Locke	364/200
4,677,613	6/1987	Salmond et al.	370/85
4,680,583	7/1987	Grover	340/825.52
4,701,878	10/1987	Gunkel et al.	364/900
4,710,893	12/1987	McCutcheon et al.	364/900
4,716,410	12/1987	Nozaki	340/825.52
4,727,475	2/1988	Kiremidjian	364/200
4,760,553	7/1988	Buckley et al.	364/900
4,773,005	9/1988	Sullivan	364/200
4,775,931	10/1988	Dickie et al.	364/200

FOREIGN PATENT DOCUMENTS

0051425	5/1982	European Pat. Off. .
0104545	4/1984	European Pat. Off. .
59-52331	3/1984	Japan .
1508854	4/1978	United Kingdom .
1518565	7/1978	United Kingdom .
2035636	6/1980	United Kingdom .
2070826	5/1984	United Kingdom .
0143160	6/1985	United Kingdom .
2167274	5/1986	United Kingdom .
0207313	1/1987	United Kingdom .

OTHER PUBLICATIONS

Hill et al., "Dynamic Device Address Assignment Mechanism," IBM Technical Disclosure Bulletin, vol. 23, No. 8, Jan. 1981, pp. 3564–65.

Search Report, dated May 21, 1986, for British Patent Application No. 8607632.

Primary Examiner—Thomas C. Lee

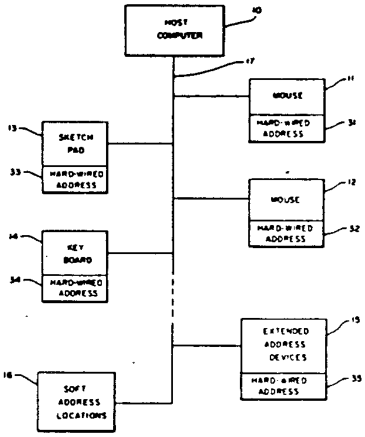
Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman

[57] **ABSTRACT**

A method for activating one of a plurality of devices coupled to a communications medium at a first address location and controlled by a host computer. A plurality of extended addresses are provided to the devices, each of the devices having a unique extended address. A command is transmitted from the host computer to the first address. One of the plurality of devices is activated by transmitting the unique extended address of the device on the communications medium, the activated device responding to further commands sent to the first address location.

3 Claims, 5 Drawing Sheets

ADB



(Apple Desktop Bus)

FIG_1

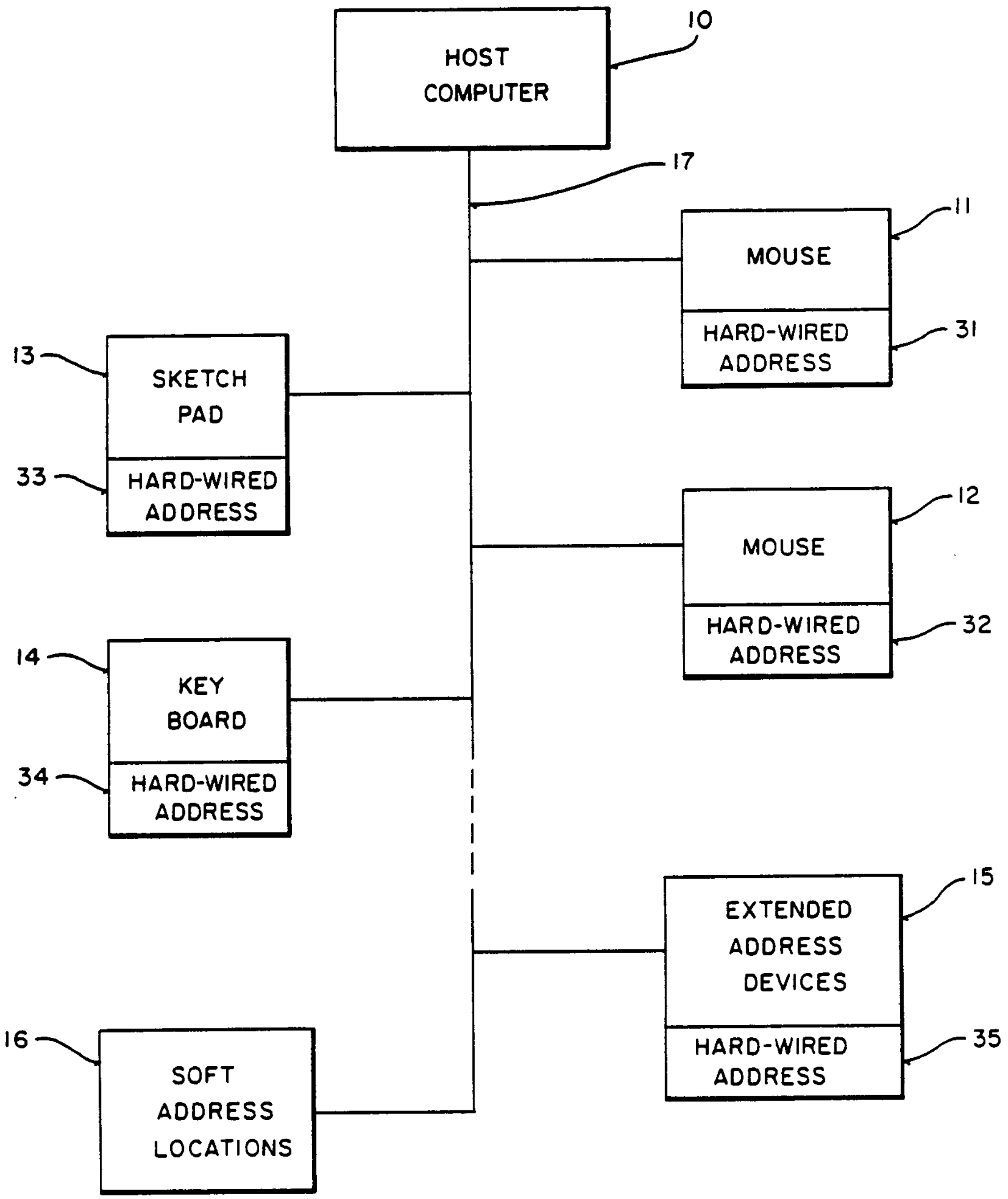

HOST COMPUTER
10
17
MOUSE
11
HARD-WIRED ADDRESS
31
13
SKETCH PAD
33
HARD-WIRED ADDRESS
MOUSE
12
HARD-WIRED ADDRESS
32
14
KEY BOARD
34
HARD-WIRED ADDRESS
EXTENDED ADDRESS DEVICES
15
HARD-WIRED ADDRESS
35
16
SOFT ADDRESS LOCATIONS

FIG_6

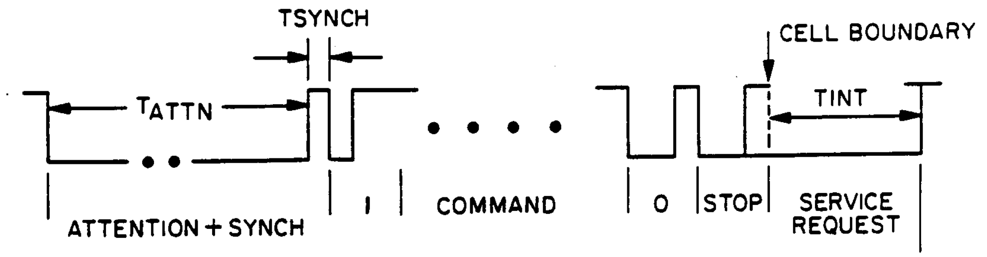


FIG_2

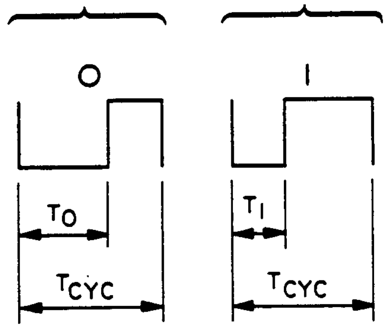


FIG_3

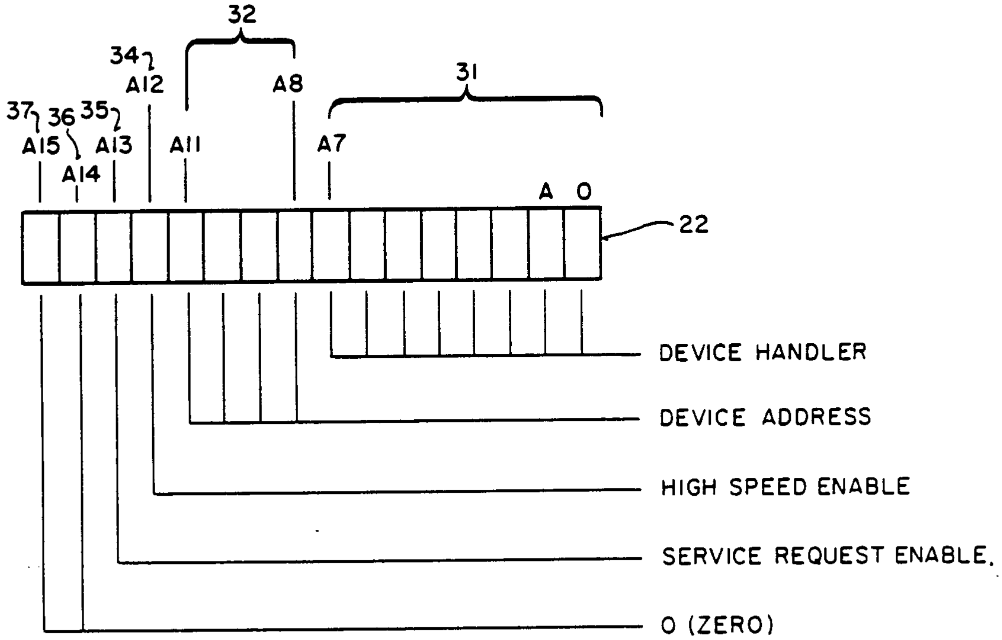
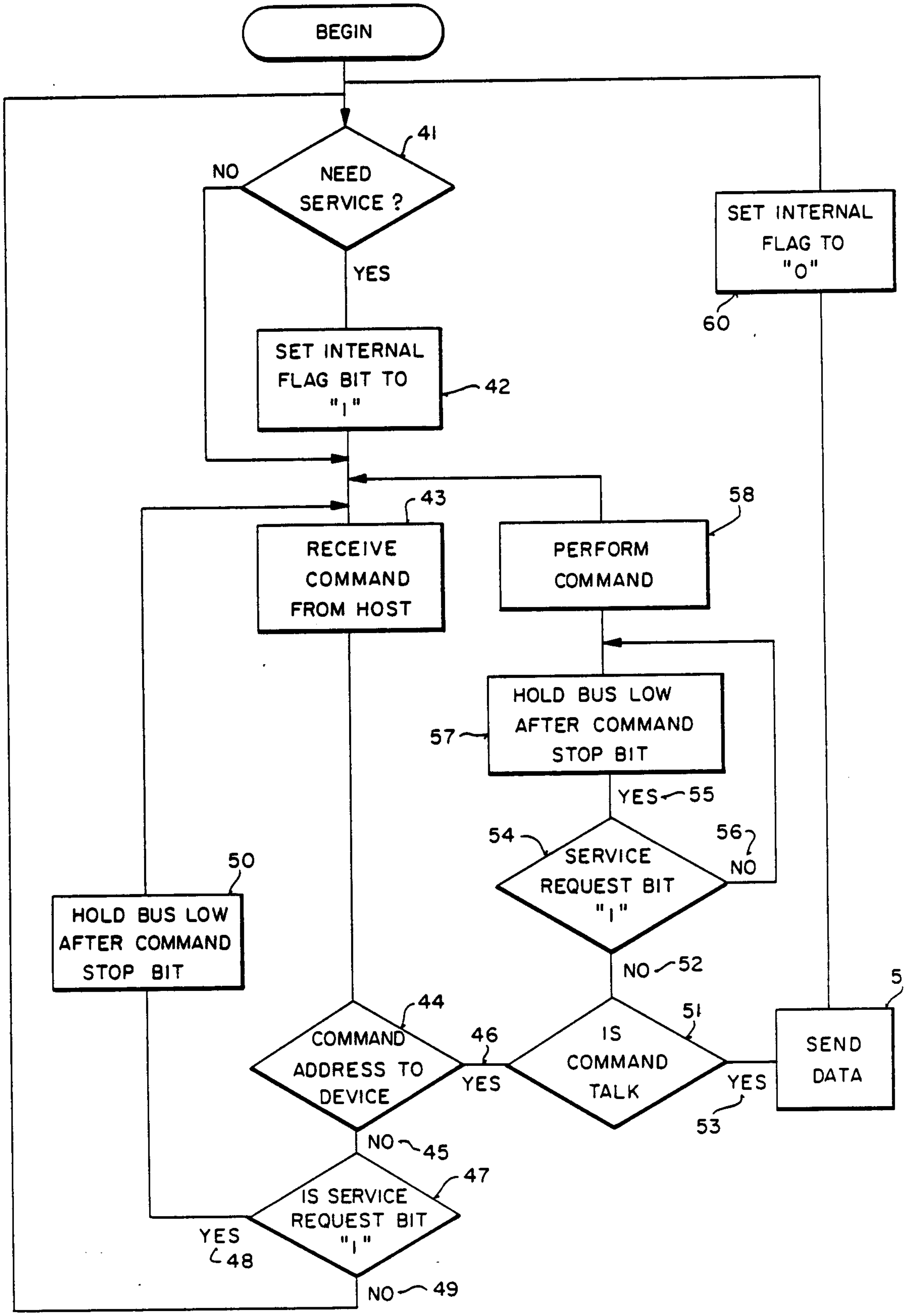
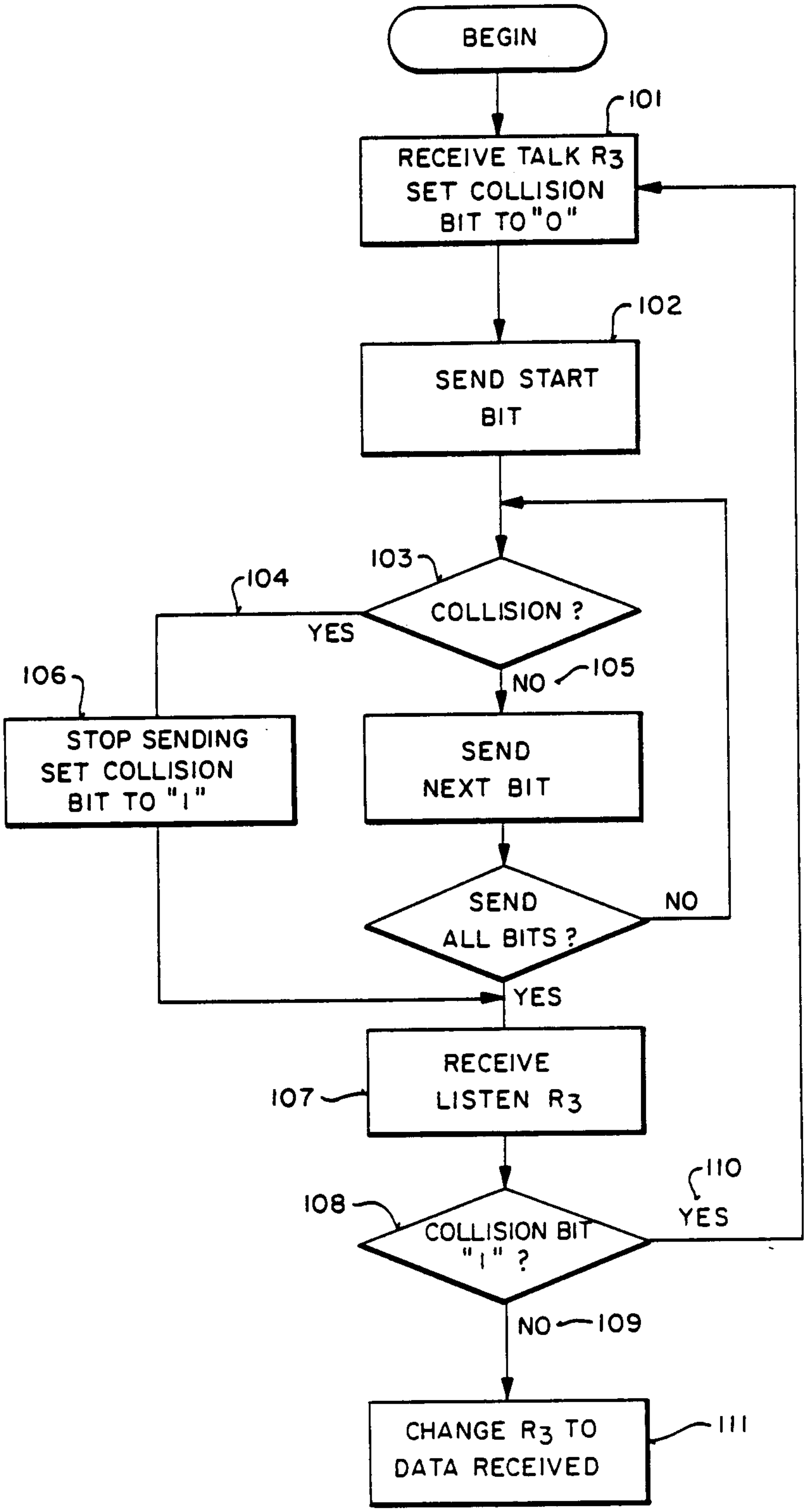


FIG _ 4


BEGIN
41
NEED SERVICE ?
NO
YES
SET INTERNAL FLAG BIT TO "1"
42
43
RECEIVE COMMAND FROM HOST
58
PERFORM COMMAND
SET INTERNAL FLAG TO "0"
60
57
HOLD BUS LOW AFTER COMMAND STOP BIT
YES
55
54
SERVICE REQUEST BIT "1"
56
NO
NO
52
50
HOLD BUS LOW AFTER COMMAND STOP BIT
44
COMMAND ADDRESS TO DEVICE
46
YES
IS COMMAND TALK
51
YES
53
SEND DATA
59
NO
45
47
IS SERVICE REQUEST BIT "1"
YES
48
NO
49

"APPLE_PAT_4_918_598_04" 116 KB 2000-02-22 dpi: 300h x 300v pix: 1904h x 2903v

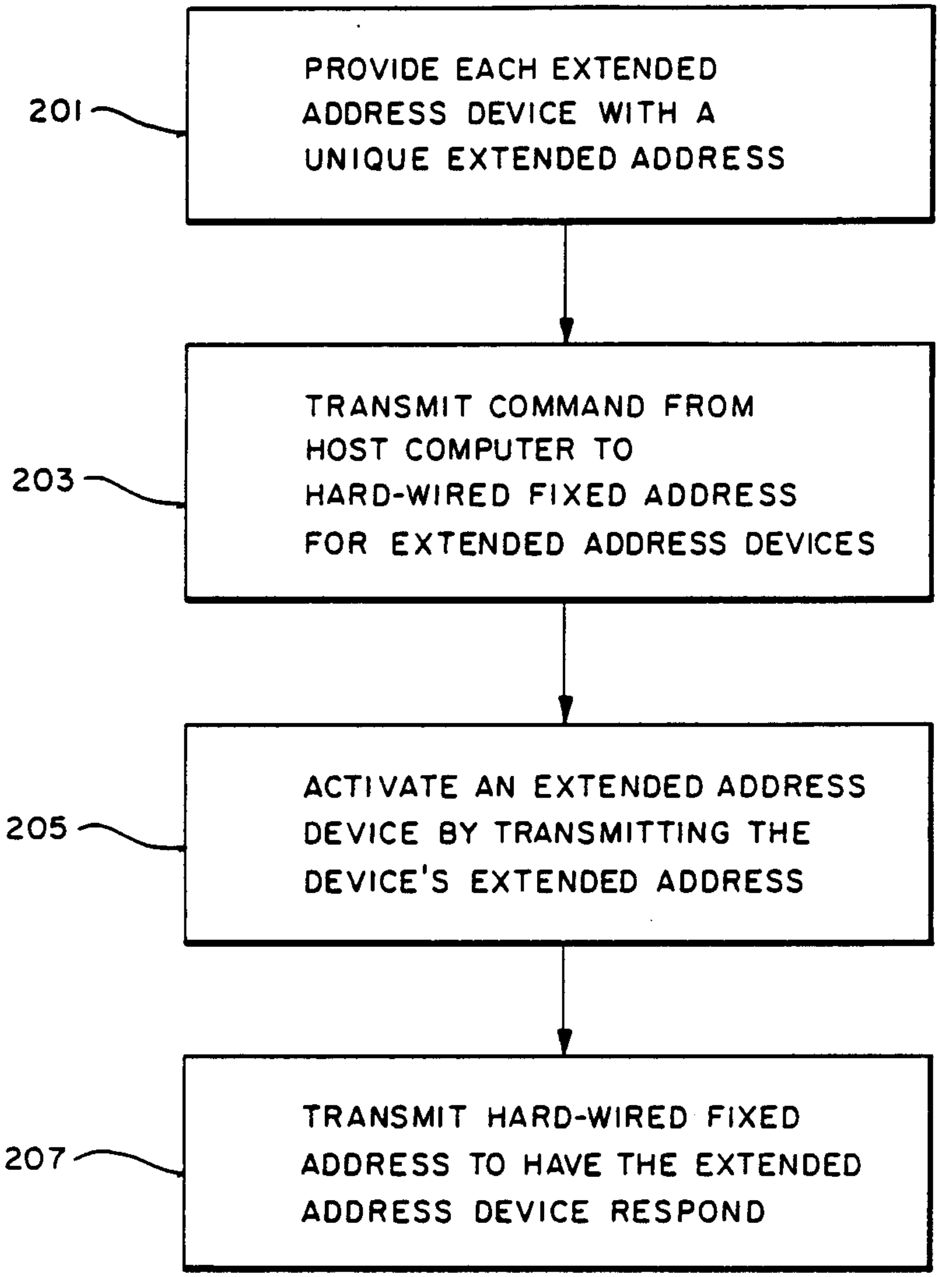
FIG_5



"APPLE_PAT_4_918_598_05" 78 KB 2000-02-22 dpi: 300h x 300v pix: 1843h x 2749v

U.S. Patent Apr. 17, 1990 Sheet 5 of 5 4,918,598

FIG _ 7


201
PROVIDE EACH EXTENDED ADDRESS DEVICE WITH A UNIQUE EXTENDED ADDRESS
203
TRANSMIT COMMAND FROM HOST COMPUTER TO HARD-WIRED FIXED ADDRESS FOR EXTENDED ADDRESS DEVICES
205
ACTIVATE AN EXTENDED ADDRESS DEVICE BY TRANSMITTING THE DEVICE'S EXTENDED ADDRESS
207
TRANSMIT HARD-WIRED FIXED ADDRESS TO HAVE THE EXTENDED ADDRESS DEVICE RESPOND

METHOD FOR SELECTIVELY ACTIVATING AND DEACTIVATING DEVICES HAVING SAME FIRST ADDRESS AND DIFFERENT EXTENDED ADDRESSES

This is a divisional of application Ser. No. 765,396 filed Aug. 14, 1985 now U.S. Pat. No. 4,910,655.

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention relates the field of communications media for transferring data between a source and a plurality of peripheral devices coupled to the source. More particularly, the present invention relates to data transfer along a peripheral device bus between a plurality of peripheral devices and a host computer.

2. Art Background

In the computing industry, it is quite common to transfer data and commands between a plurality of data processing devices, such as for example, computers, printers, memories and the like. The interconnection of computers and other peripheral devices principally developed in the early 1970's with the advent of computer networking systems, which permitted the distribution of access to computing resources beyond the immediate proximity of a main frame computer.

Networks, such as the ARPA network, were developed to provide access by various users to large timesharing systems and the transfer of data between such systems. In the case of geographically local networks, so-called "local area networks" (LANs) were developed to connect together a collection of computers, terminals and peripherals located, typically in the same building or adjacent buildings, and permitted each of these devices to communicate among themselves or with devices attached to other networks. Local area networks permit the implementation of distributed computing. In other words, some of the devices coupled to the local area netowrk may be dedicated to perform specific functions, such as file storage, data base management, terminal handling, and so on. By having different machines perform different tasks, distributed computing can make the implementation of the system simplier and more efficient.

Presently, networking has only been applied to provide communications between data processing devices, which are machine input devices. However, it would also be useful to provide a networking means to provide communication between a single computer and a plurality of peripheral devices such as human input devices, listen only devices, appliances, etc. Human input devices include keyboards, cursor control devices (such as a "mouse"), and sketch pads, etc. Listen only devices include transaction logs, etc. In the prior art, such devices are attached to a host computer through a port dedicated to each device. Often, additional "cards" are required to allow a peripheral input device to be added. Further, the addition of cards requires that the host computer be powered down, with no mechanism for adding peripheral devices to a live system. Such prior art systems are inefficient since peripheral devices are not generally operated simultaneously. (for example, someone using a mouse is generally not using the keyboard or sketchpad at the same tine). Thus, the devices could share a common line to the host computer without creating data traffic problems, eliminating the needs for cards.

Prior art networking schemes also include elaborate methods for establishing control of the network to allow a device to transmit. Such systems are not needed for networking of peripheral devices, since only one is generally used at a time. In addition, prior art networking schemes provide for means for attached devices to identify themselves to each other through elaborate "handshaking" schemes. Again, such complexity is not required to connect peripheral devices since there is no need for these devices to identify themselves to other devices, only to the host computer.

Therefore, it is an object of the present invention to provide a communications medium for a plurality of peripheral devices, which provides a simple and efficient means for coupling those devices to a host computer.

It is a further object of the present invention to provide a communications medium by which all such peripheral devices can be coupled to a host computer at a single input.

It is still another object of the present invention to provide a communications medium which provides a means for peripheral devices to indicate a need for servicing to the host computer.

It is yet another object of the present invention to provide a communications medium which provides a means for determining if the communications medium is in use.

It is another object of the present invention to provide a communications medium which allows peripheral devices to be added during operation of the system.

SUMMARY OF THE INVENTION

A communications medium is disclosed including apparatus and methods for transferring data between a plurality of peripheral devices and a host computer. In the preferred embodiment, a plurality of peripheral devices such as human input devices (including mice, keyboards, sketchpads, etc.), appliances, listen only devices, etc., are coupled to a common cable for data transmission and reception of commands. A peripheral device coupled to the cable may signal the host computer when it requires servicing. This peripheral device will continue to request service until the host computer commands it to transmit its data. All peripheral devices of the same generic type (e.g., all keyboards), may have an identical hard wired address used as an identification number. In this manner, the host computer can identify the generic type of device communicating on the cable. If more than one of the same type of device is coupled to the cable (e.g., 2 mice), the host computer will assign new addresses in the status registers of the mice so they can be differentiated.

In the preferred embodiment, a return to zero modulation scheme is used to transmit data and commands over the cable. As a result, a peripheral device will assume a collision if it attempts to transmit a high signal on the cable and the cable is pulled low by another device. In order to simplify the protocol of the system, only the computer can initiate communication.

The present invention permits the addition of peripheral devices to a computer while the computer is in use, without the need to power down the computer system. The present invention can be embodied in a narrow band medium, as well as broad band, fiber optic, infrared and other media.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is block diagram illustrating the networking system of the present invention.

FIG. **2** is a timing diagram illustrating the present invention's use of return to zero encoding.

FIG. **3** illustrates a register of a peripheral device of the present invention.

FIG. **4** is a flow chart illustrating the sequence of operations utilized by a peripheral device to request service by the host computer.

FIG. **5** is a flow chart illustrating the sequence the operations ulitized to provide new addresses to devices sharing the same hard-wired address.

FIG. **6** is a timing diagram illustrating a command transaction of the present invention.

FIG. **7** is a flow chart illustrating the sequence of operations utilized to activate a peripheral device.

DETAILED DESCRIPTION OF THE INVENTION

A peripheral device bus including apparatus and methods for transferring data between a plurality of peripheral devices coupled to a host computer is disclosed. In the following description numerous specific details are set forth, such as specific numbers, registers, addresses, times, signals, and formats, etc., in order to provide a thorough understanding of the present invention. However, it will be apparent to one skilled in the art that the present invention may be practiced without these specific details. In other instances, well-known circuits and devices are shown in block diagram form in order not to unnecessarily obscure the present invention.

Referring to FIG. **1**, the preferred embodiment of the present invention may be seen. A plurality of peripheral devices, generally identified by numbers **11** through **16** are coupled through a single cable **17** to a host computer **10**. In the preferred embodiment, all devices communicate with the host computer by a mini-phono jack with the following connector assignments; tip-power, ring-data, sleeve-power return. A "high" signal (**1**) is 2.4 volts minimum. A "low" signal (**0**) is 0.8 volts maximum. Although a single cable is contemplated in the preferred embodiment of the present invention, other communications media, such as broad band methods, fiber optic systems, and infrared signals, are contemplated.

The bus of the present invention supports coded devices (for which a keystroke represents a symbol or a function, such as a keyboard **14**), relative devices (in which movement of a display cursor in response to a control device, such as a mouse **11** or **12**, may be from any starting point), and absolute devices (for which there is a constant and direct relationship between display position and device position, such as sketch pad **13**).

The system also permits the networking of extended address devices. Extended address devices share a common hard wired address **35**, but further include an address unique to the individual device which the host computer must recognize before the device can be accessed. As shown in box **201** of the flow chart of FIG. 7, each extended address device is provided with a unique extended address. For example, is it contemplated that appliances may be coupled to the host computer and controlled by the host computer. In such a situation, all appliances would have an identical hardwired fixed address. The host computer, on a first level, would simply address the hard wired address for appliances. As shown in box **203** of the flow chart of FIG. 7, a command is transmitted from the host computer to the hardwired fixed address for the extended address devices. At this time, all appliances coupled to that address are inactive. An individual appliance may be activated by the host computer if the host computer sends a signal to that appliance which matches the extended address of the appliance. An extended address is an individual identification number, which, in the preferred embodiment, may be up to 64 bytes long. As shown in box **205** of the flow chart of FIG. 7, an extended address device is activated by transmitting the device's extended address. Once the host computer has provided the extended address, the device having that address is active. Subsequent commands to the appliance address location will be executed by that device without the need for providing the extended address each time. As shown in box **207** of the flow chart of FIG. 7, the hardwired fixed address is transmitted in order to have the extended address device respond. An activated appliance will respond to all commands to the appliance address, while unactivated devices remain passive. To deactivate an active extended address device, the host computer provides the extended address of another extended address device, activating it and deactivating the previously active device. It is contemplated that any device which could be controlled by the host computer is suitable for the present networking scheme, such as lights, ovens, sprinkler systems, phone answering machines, etc. It is contemplated that at least one other hardwired address for extended address devices be provided in the present system. Such an address would be used for system protection schemes or user identification schemes. For example, a device at this location could contain an extended address which must be provided by the system user before the system could be enabled. In other instances, individual operations could require that the extended address of other security devices be provided by the host computer prior to performance. Such security devices could function as "keys" to lock the entire system or certain operations performed on the system.

Also reserved for use on the network of the present invention are soft address locations **16**. Soft address locations are reserved for duplicates of peripheral devices coupled to the bus. When more than one mouse is coupled to the bus, for example, the host computer assigns new addresses to each mouse, those addresses being at the soft address locations.

Although specific examples have been given for each type of device coupled to the bus, there may be more than one kind of each type of device with that address. For example, a sketch pad has been given as an absolute device but a touch screen would also be considered an absolute device and be assigned the same fixed command address as the sketch pad. In those situations, the host computer will assign new addresses from the soft address locations to each device.

In the preferred embodiment of the present invention, the various peripheral devices have been assigned addresses as shown below:

Address	Device Types	Example
0000 (zero)	extended address device	security systems, user ID

-continued

Address	Device Types	Example
0001 (one)	extended address device	appliances
0010 (two)	coded devices	keyboard
0011 (three)	relative devices	mouse, track ball
0100 (four)	absolute devices	sketchpad, touch screen
0101 (five)	reserved	none
0110 (six)	reserved	none
0111 (seven)	reserved	none
1000 (eight)	soft addressed	duplicate peripheral devices
—	—	—
—	—	—
1111 (15)	soft addressed	duplicate peripheral devices

It will appreciated by one skilled in the art that other addresses may be assigned to these devices containing more or less bits than in the preferred embodiment. Fixed hard-wired addresses **31, 32, 33** and **34** are shown in FIG. **1** for mouse **11**, mouse **12**, sketch pad **13**, and keyboard **14**, respectively.

All peripheral devices have four registers in the preferred embodiment to receive data and send data. For each device, register **3** talk and register **3** listen have status information such as device address and handler information. The remaining registers are data registers which are device specific except register **2** listen which contains the extended addresses for extended address devices or device specific contents for soft addressed devices.

In the preferred embodiment of the present invention, there are three types of communication on the peripheral bus: commands, data and global signals. Commands are sent from the host computer to the peripheral devices, data is sent from the host computer to the devices or from the devices to the host computer, and global signals are special messages sent to the entire system.

In the preferred embodiment data is encoded as the ratio of low time to high time of each bit cell. A bit cell boundary is defined by a falling edge on the bus. A "zero" is encoded as a bit cell in which the low time is greater than the high time. This is shown in FIG. **2** by bit cell **20**. Therefore, a "1" is defined as a bit cell in which the low time is less than the high time as shown by cell **21** of FIG. **2**. In the present preferred embodiment, a start bit is defined as a "1". A stop bit is a "0" which does not have an additional falling edge to define the bit cell time. The stop bit is used to synchronize the stopping of transactions on the bus.

The period for each bit cell of command signals and low speed data transmission is approximately 100 microseconds plus or minus 30%. For high speed data transmission, the bit cell is 50 microseconds plus or minus 1%. The format of a data transaction is a start bit (**1**), followed by up to 256 bits of data and ending with a stop bit. It will be appreciated that when other communications media are utilized, other signaling methods may be utilized.

Commands are sent only by the host. In the preferred embodiment of the present invention, there are three commands; talk, listen, and flush. As shown in FIG. **6**, to signal the start of a command, an attention pulse is sent out. An attention pulse is generated by the host computer by transmitting a bus low for a period of "T-attn". In the preferred embodiment, T-attn is approximately 560–1040 microseconds. The attention pulse is followed by a synch pulse to give the initial bus

timing. The following edge of the synch pulse is used as a timing reference for the first bit of the command. The command is followed by a stop bit, (in the preferred embodiment a "0"). After the stop bit, the bus returns to its normally high state unless a device requests service.

The command is an 8 bit value in the preferred embodiment. The command includes a 4 bit device address field which specifies the fixed hardwired address of the desired peripheral device (e.g., 0011 for a mouse). The next 2 bits form the command and the final 2 bits form a register address field which allows a specific register, **R0–R3** within an addressed peripheral device to be specified. In the preferred embodiment, the commands have the following bit code:

Command	Code
Flush	01
Listen	10
Talk	11

The talk command orders the addressed device to provide its data to the host computer. The listen command orders the addressed device to accept data from the host computer and place it in one of its registers. The flush command has an effect on each device which is defined by the individual device. It can be used for such functions as clearing a register or resetting all keys on a keyboard so that they will be sent again.

When a peripheral devices is addressed to talk, it must respond within a certain period, called the "time out" period. The time out, "Tlt", is approximately 140 to 260 microseconds (2 bit cells). The selected device, if it does not time out, becomes active on the bus and performs its data transaction, and then "untalks" itself and goes inactive on the bus.

Global signals are used for transactions which are neither commands nor data transactions. Global signals include: attention and synch, which is used to signal the start of a command and to give initial bus timing; service request, a transaction that devices use to signal the host that they require service; and reset, used to issue a break on the bus by holding the bus low for a minimum of "Tres", which is approximately 2.8 to 5.2 milliseconds, (40 bit cells). Global signals will be described in more detail in conjunction with other transactions.

Since a peripheral device can only send data when it has been commanded to talk by the host computer, the present system provides a means for a device to notify the host computer that it needs servicing. This is accomplished by having the device send a service request signal to the host computer. In the present invention, a service request is sent by holding the bus low after the stop bit of any command transaction. Each of the peripheral devices coupled to the bus include a number of registers (in the preferred embodiment four registers). FIG. **3** shows one of the registers for a peripheral device. Bit **A13** has been identified as the service request enable bit. When this bit is set high by the host computer, the device is enabled to hold the bus low after the stop bit of a command transaction, as shown in FIG. **6**, if the device needs service. A device will keep requesting service until it receives a talk command from the host. The flow chart in FIG. **4** shows the steps followed by a device requiring service.

Initially the device determines if it requires servicing, Block **41**, that is, if it has data to send to the host. If it does, it sets an internal flag bit, Block 42. When the next command is sent out from the host, Block 43, the device checks to see if the command is addressed to the device, Block 44. If the command was not addressed to the device, Branch 45, the device checks to see if its service request enable bit, (bit A13 or register), is set high, Block 47. If so, Branch 48, it holds the bus low after the command stop bit, Block 50. (See FIG. 6) The device then waits until the next command is received from the host to see if it will be addressed to talk, Block 43. If the command is addressed to the device, Branch 46, the device determines if it is a command to talk, Block 51. If it is not a command to talk, Branch 52 the device sends a service request, Block 57, performs whatever command is instructed, Block 58, and awaits the next command, Block 43. If the command is to talk, Branch 53, the device sends its data, Block 59 and considers its service request to be satisfied, Block 60. The device continues to monitor itself to determine when it needs service, Block 41. By allowing the host computer to control the service request enable bit, more efficient operation of the bus is realized. When a service request is received, the host computer need only ask those devices whose service request bit was enabled whether they need servicing. Additionally, the host computer can disable certain devices that are not required for particular applications.

When sending data, the device is able to detect collisions. If a peripheral device tries to output a 1 and the data line is or goes to a 0, the device assumes it has lost a collision to another device. This means that another device is also sending on the bus. When this happens the losing device untalks itself from the bus and preserves the data which was being sent for retransmission. The device sets an internal flag bit if it loses a collision. Prior art peripheral devices were unable to detect collisions. This novel feature of the present invention permits more efficient operation of the communications medium. By having the device sense a collision, it can preserve the data that is transmitted and indicate to the host computer that it requires serving. Additionally, the collision detection scheme of the present invention does not require a waiting period before a collision is assumed. A device will end its transmission if the line is modulated by another device or simply not begin its transmission if the line is already in use. Further, this collision detection scheme is useful in locating multiple devices at a single hardwired address location, such as mouse 11 and mouse 12 of FIG. 1.

In such a situation, the host will change the address of the devices by forcing a collision of devices sharing the same address. The host achieves this by issuing a talk R3 command addressed to those devices. As shown in FIG. 3, Register 322 (one of the registers of the device) contains the following information. Bits A0 through A7 31 contain a device handler which tells the host computer the function of a device and the use of data provided by the device. Bits A8 through A11 32 are an address field which can be changed when more than one device, having the same command address, is coupled to the bus. In that situation, one of the soft address locations are assigned to bits A8 through A11 32 which then serve as the command address for that device. Until that time, those bit locations contain a random number which aids in the detection of collisions. For example, if two mice received a talk R3 command and both began talking at the same, neither would detect a collision. However, by having random numbers in the address

field 32 of register 3 22, the output of the two devices will eventually differ. When that occurs, one of the devices will detect a collision and stop talking. Bit A12 34 is a high speed enable bit which if set, provides for data transmission at the higher modulation rate (50 microseconds per bit frame). The high speed enable bit is set by the host computer. If the host computer is unable to receive data at the higher modulation rate, it sets the high speed enable bit low in each of the devices. If the host computer is able to accept data at the higher modulation rate, and the device is able to transmit at the higher rate, (that information being contained in the handler bits 31 of register 3), the host computer sets the high speed enable bit 34 high for the device. As previously mentioned, bit A13 35 is service request enable which is set by the host to enable the device to perform a service request transaction. Bits A14 36 and A15 37 are reserved for future use and are set to 0.

When a device receives a talk R3 command the device provides its status (handler and address) to the host computer. If there are two devices of the same type coupled to the bus, only one can respond since the other will detect a collision. FIG. 5 shows the method of assigning new addresses on the bus.

After receiving a talk R3 signal, Block 101, the device sends its status from Register 3. If the line goes low, the device determines that there has been a collision, Branch 104, it stops sending (untalks itself) and sets an internal flag bit to indicate a collision, Block 106. The host sends a listen R3 to the mouse address, Block 107. Each commend resets the internal collision flag of the device. The device checks to see if its collision bit is set, Block 108. If the collision bit is not set, Branch 109, the device changes A8 through A11 to the soft address provided by the listen R3 command, Block 111. In this manner the address of the winning device is changed with the host computer keeping track of the new address of the device. If a collision bit is detected by the device after a listen R3 command, Branch 110, the device does not change the soft address bits, but may change other fields in R3. The host computer sends out another talk R3 command, Branch 101 to see if any devices remain at the mouse address. In this situation the remaining mouse will send its start bit, Block 102, not detect a collision, Branch 105, and send its status from register 3, Block 112. The host computer will send back a listen R3 command to the mouse address, Block 107. The remaining mouse will not detect a collision bit being set in this instance, Branch 109 so it will change bits A8 through A11 of register 3 to the soft address received from the host computer, Block 111. The host computer then sends out another talk R3 command to the mouse address, Block 101. This time, since no mouse remains at that address, the bus is timed out and the host computer knows that it has assigned new addresses to each of the mice sharing the mouse address.

In one embodiment of the present invention, peripheral devices have a device on them to indicate activity called the activator. The activator can be a special key on a keyboard or a button on a mouse. When more than one of a device is coupled to the bus, the host computer can display a message requesting one of the devices to use the activator. The host can then issue a listen R3 command which will change the address of the device which is activated. In this manner individual devices can be located and assigned new addresses in multiuser applications.

Thus, a peripheral device bus has been described which allows a plurality of peripheral devices to be coupled to a host computer through a single port.

We claim:

1. A method for transferring signals and data, wherein the signals and data are transferred under the control of a host computer between the host computer and first and second peripheral devices, wherein the signals and data are transferred over a bus coupling the first and second peripheral devices to the host computer, wherein the first and second peripheral devices are initially inactive, and wherein the first and second peripheral devices have a same first address, comprising the steps of:

the host computer initially activating the first peripheral device by transmitting over the bus (1) the first address of the first peripheral device and (2) a signal that matches a first number stored as an extended address of the first peripheral device, wherein the second peripheral device remains inactive;

after the first peripheral device is activated, the host computer sending a command over the bus to the first address of the first peripheral device, wherein the first peripheral device executes the command without the host computer sending the extended address of the first peripheral device, and wherein the second peripheral device remains inactive;

the host computer both activating the second peripheral device and deactivating the first peripheral device by transmitting over the bus a second number as an extended address, wherein the second number is the extended address of the second peripheral device, and wherein the second number is different from the first number;

after the second peripheral device is activated, the host computer sending a command over the bus to the first address of the second peripheral device, wherein the second peripheral device executes the command without the host computer sending the extended address of the second peripheral device, and wherein the first peripheral device remains inactive.

2. The method of claim 1 for transferring signals and data, wherein at least one of the first and second peripheral devices comprises an appliance.

3. The method of claim 1 for transferring signals and data, wherein at least one of the first and second peripheral devices comprises a system protection device.

* * * * *

United States Patent [19]

Baker et al.

[11] **Patent Number:** 4,926,316

[45] **Date of Patent:** May 15, 1990

[54] **MEMORY MANAGEMENT UNIT WITH OVERLAPPING CONTROL FOR ACCESSING MAIN MEMORY OF A DIGITAL COMPUTER**

[75] Inventors: **Paul A. Baker**, Los Altos; **Gary L. Marten**, Cupertino, both of Calif.

[73] Assignee: **Apple Computer, Inc.**, Cupertino, Calif.

[21] Appl. No.: **933,071**

[22] Filed: **Dec. 17, 1986**

Related U.S. Application Data

[63] Continuation of Ser. No. 426,869, Sep. 29, 1982, abandoned.

[51] **Int. Cl.^{5}** **G06F 9/00**

[52] **U.S. Cl.** **364/200**; 364/238.4; 364/246; 364/246.3; 364/246.4; 364/246.5; 364/245.4

[58] **Field of Search** 364/200, 900

[56] **References Cited**

U.S. PATENT DOCUMENTS

3,828,327	8/1974	Berglund et al.	340/172.5
3,902,163	8/1975	Amdahl et al.	340/172.5
4,004,278	1/1977	Nagashima	340/172.5
4,037,215	7/1977	Birney et al.	364/200
4,084,224	4/1978	Appell et al.	364/200
4,084,227	4/1978	Bennett et al.	364/200
4,084,228	4/1978	Dufond et al.	364/200
4,096,568	6/1978	Bennett et al.	364/200
4,104,718	8/1978	Poublan et al.	364/200
4,130,867	12/1978	Bachman et al.	364/200
4,297,743	10/1981	Appell et al.	364/200
4,316,245	2/1982	Luu et al.	364/200
4,354,225	10/1982	Frieder et al.	364/200
4,376,297	3/1983	Anderson et al.	364/200
4,378,591	3/1983	Lemay	364/200
4,410,941	10/1983	Barrow et al.	364/200
4,424,561	1/1984	Stanley et al.	364/200

FOREIGN PATENT DOCUMENTS

0040702	2/1981	European Pat. Off. .
190324	12/1982	New Zealand .
1413739	11/1975	United Kingdom .
1477977	6/1977	United Kingdom .
1487078	9/1977	United Kingdom .
1498116	1/1978	United Kingdom .
1547382	6/1979	United Kingdom .
1557121	12/1979	United Kingdom .
1577592	10/1980	United Kingdom .
1585960	3/1981	United Kingdom .
2073458	2/1984	United Kingdom .

OTHER PUBLICATIONS

Wescon Conference Record, vol. 25, Sep. 1981, pp. 1–9, El Segundo, U.S.; S. Walters: "Memory Management Made Easy with the Z8000".

Electronic Design, vol. 29, No. 17, Aug. 1981, pp. 115–121, Waseca, MN, U.S.; D. L. Collins et al.: "Memory Management Chip Masters Large Data Bases".

Electronics International, vol. 54, No. 11, Jun. 1981, pp. 134–138, New York, U.S.; J. Beekmans et al.: "Chip Set Bestows Virtual Memory on 16-bit Minis".

Primary Examiner—Gareth D. Shaw

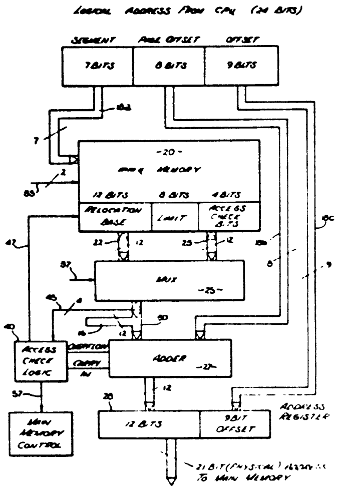
Assistant Examiner—John G. Mills

Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman

[57] **ABSTRACT**

An improved memory management unit (MMU) for interfacing between a CPU and a main computer memory. The MMU receives logical addresses from the CPU and converts a portion of the logical address to be used for generating a physical address to address to address the main memory. The MMU memory contains relocation data which is stored in a plurality of segments known as contexts. For a given logical address provided by the CPU, the CPU also selects an appropriate context so that the mapping of the main memory is determined by the selected relocation base. This permits relocation data to be stored for a plurality of processes and thus, allows several programs to be run without reprogramming the MMU. Special "limit" bits and "access" bits are also stored in the MMU's memory for each of the relocation base data. The limit bits are used to check the range of the memory area requested for a given context to determine if it is in the allowable range. Access bits are used to determine if the type of access being requested is a legal access for the given context. Because the MMU stores a number of relocation bases which are programmable by the CPU, areas of main memory can be accessed by more than one context, thereby providing an overlapped mapping of the main memory. For example, in a supervisory mode the supervisory context is able to access all of the main memory.

5 Claims, 3 Drawing Sheets



Lisa
MMU

"APPLE_PAT_4_926_316_01" 802 KB 2000-02-23 dpi: 600h x 600v pix: 3858h x 6039v

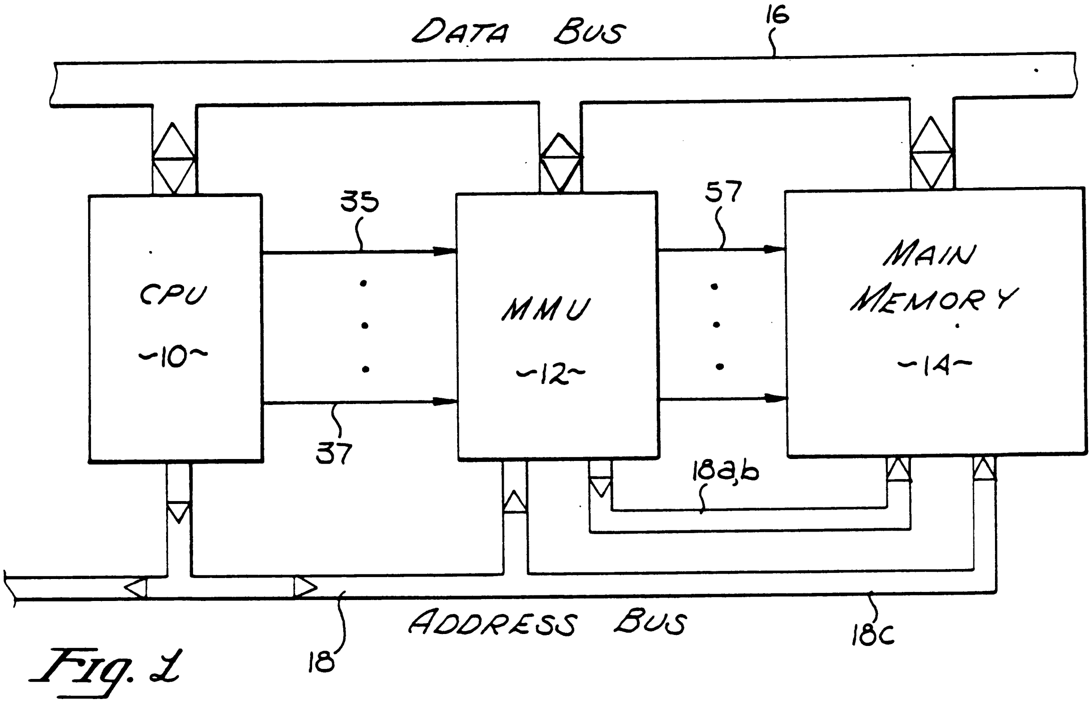

DATA BUS
16
CPU
~10~
35
37
MMU
~12~
57
MAIN MEMORY
~14~
18a,b
ADDRESS BUS
18
18c

Fig. 1

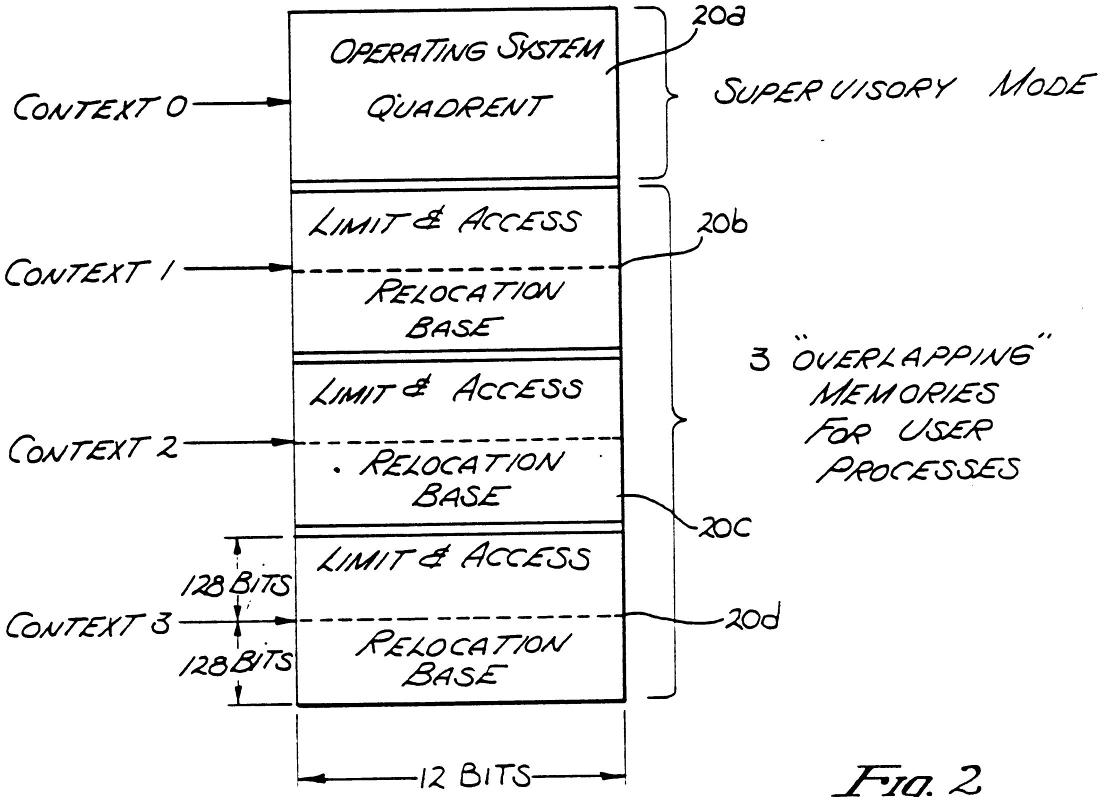
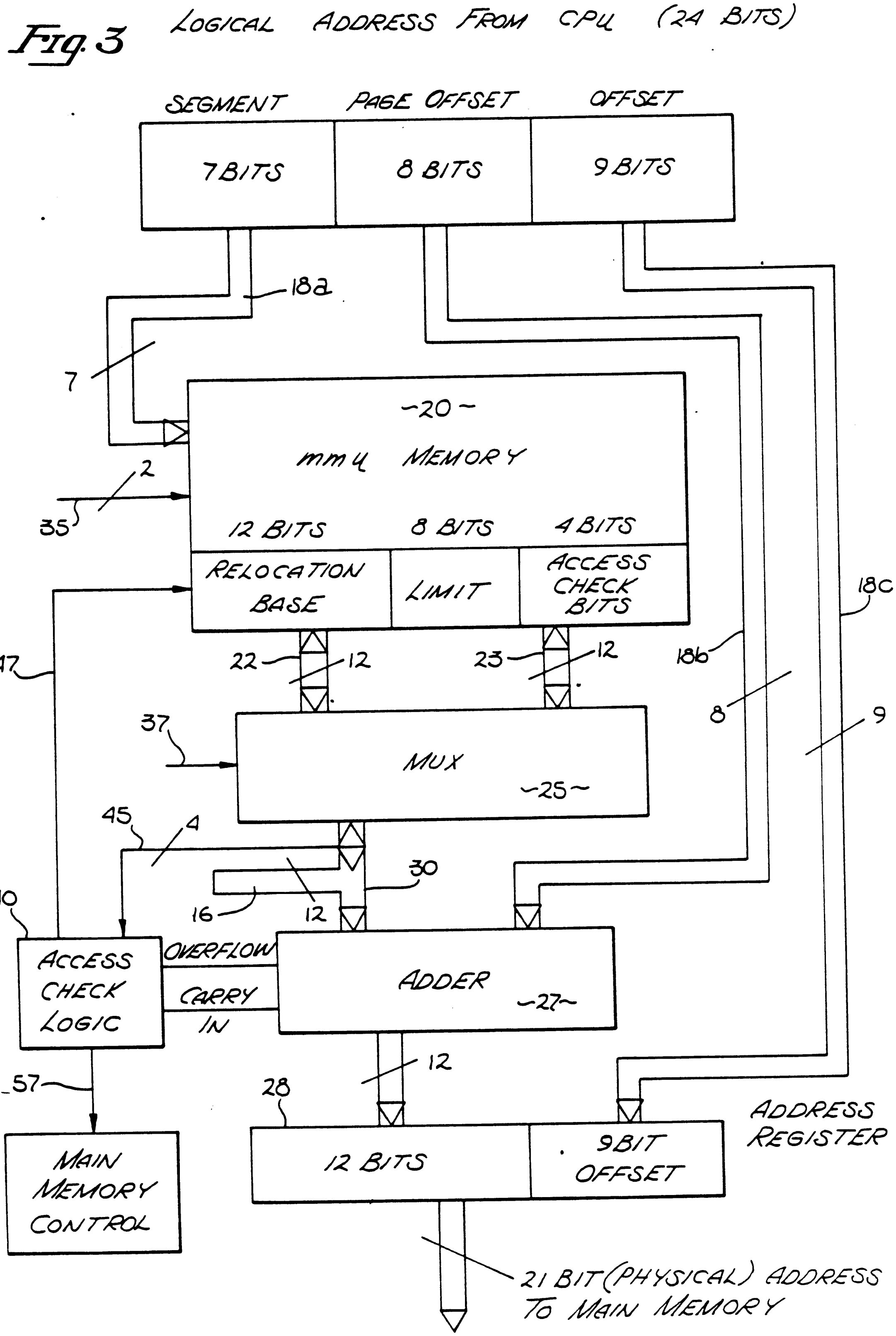

20a
OPERATING SYSTEM QUADRENT
CONTEXT 0
SUPERVISORY MODE
LIMIT & ACCESS
20b
CONTEXT 1
RELOCATION BASE
3 "OVERLAPPING" MEMORIES FOR USER PROCESSES
LIMIT & ACCESS
CONTEXT 2
RELOCATION BASE
20c
LIMIT & ACCESS
128 BITS
CONTEXT 3
20d
RELOCATION BASE
128 BITS
12 BITS

Fig. 2

Fig. 3



"APPLE_PAT_4_926_316_03" 338 KB 2000-02-23 dpi: 600h x 600v pix: 3760h x 5658v

U.S. Patent **May 15, 1990** **Sheet 3 of 3** **4,926,316**

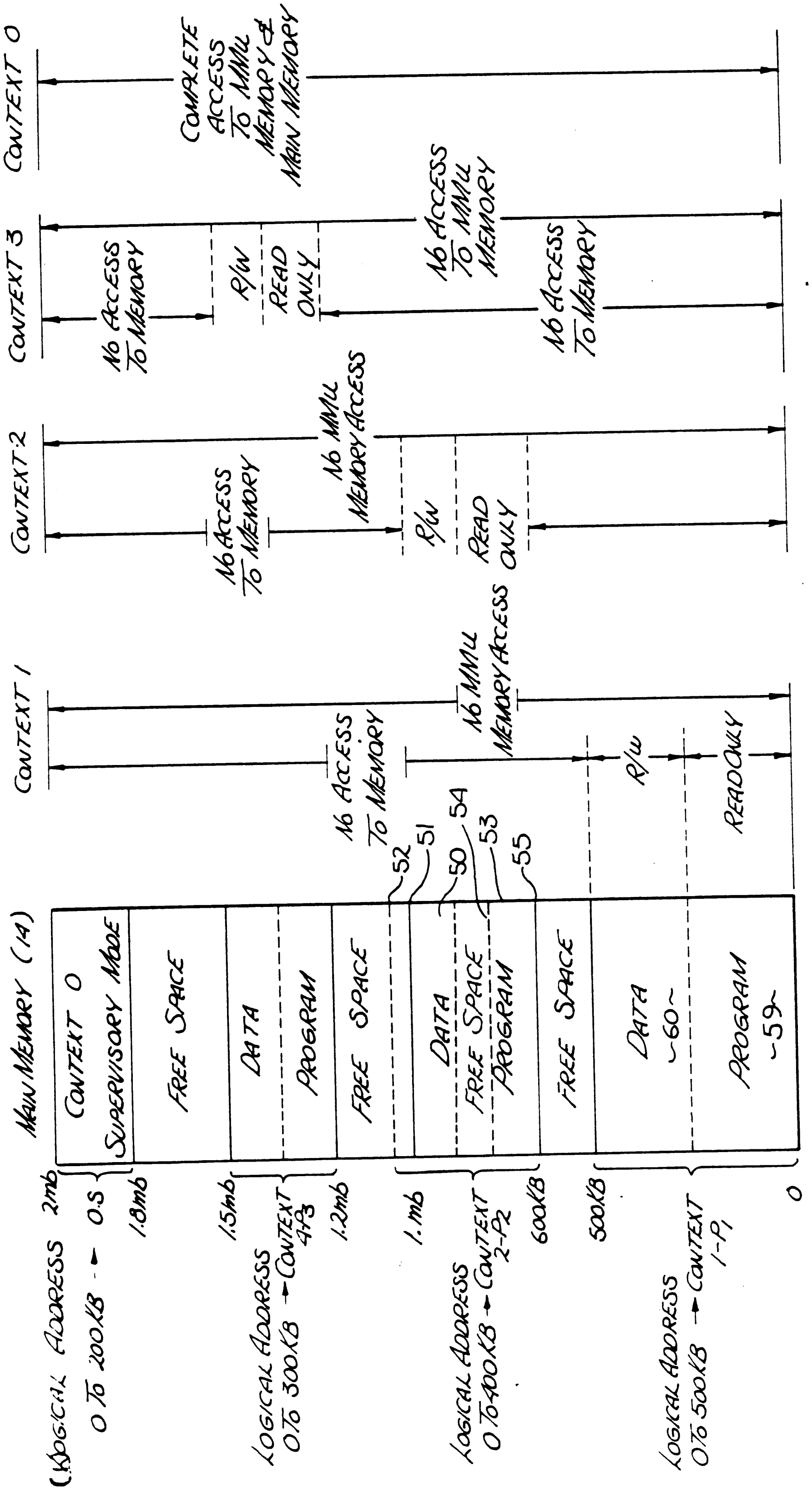


Fig. 4

MEMORY MANAGEMENT UNIT WITH OVERLAPPING CONTROL FOR ACCESSING MAIN MEMORY OF A DIGITAL COMPUTER

This is a continuation of application Ser. No. 426,869 filed Sept. 29, 1982.

BACKGROUND OF THE INVENTION

1. Field of the Invention.

The invention relates to the field of computer memories and units for managing the contents of such memories.

2. Prior Art

In most computers, a central processing unit (CPU) communicates directly with both an address bus and a data bus. These buses are coupled to a main memory (or main memory systems) in addition to numerous other items such as input/output ports, specialized processors, DMA units, etc. The main computer memory is often the most expensive component of the computer, particularly when compared to the price of currently available microcomputer CPUs such as the 8080, 8086, 6800 and 68000. Thus, it is important to efficiently utilize the computer's main memory.

Memory management units (MMUs) are used in the prior art to provide efficient utilization of the computer's main memory. These units perform housekeeping functions such as remapping, etc. Often, an MMU includes a memory which stores a data relocation base The higher order bits of the logical address from the CPU are used to address the MMU's memory. These bits from the CPU's standpoint, for instance, select a segment of the main memory. The selected CPU segment number is replaced by a new number from the MMU's memory and effectively, a relocation occurs between the logical address from the CPU and the physical address used to access the main memory.

Another function performed by prior art MMUs is to check addresses from the CPU to verify that they fall within certain ranges. A limit number stored in the MMU's memory is compared with lower order bits of the logical address (for example, the page offset) to assure that the page offset falls within a predetermined address range of the selected segment number. This prevents, by way of example, the accidental reading of "data" from memory locations where data has not been placed.

The present invention builds upon those prior art MMUs which provide a relocation base and address range verification. As will be seen, the MMU's memory is expanded in one direction to store signals representing the nature of information stored in the main memory. This is used to control access of the main memory and, by way of illustration, prevents accidental writing into programs and user access to operating systems. The MMU's memory is expanded in another direction so that overlapping memory management is provided. This allows several different processes (program and data) to be run by the computer without reprogramming the MMU memory.

SUMMARY OF THE INVENTION

An improved memory management unit (MMU) is described for use with a computer which includes a central processing unit (CPU) and a main memory. The MMU includes a relocation base and when receiving first address signals from the CPU, provides second address signals for accessing the memory. The MMU also includes storage means for receiving and storing signals representative of the types of information stored in locations in the main memory. Accessing means are provided for accessing these stored signals when the corresponding locations are accessed in the main memory. The stored signals from the storage means are coupled to the main memory to, for example, limit access of certain types of data in the memory such as operating systems. The signals may be also used to permit reading-only of programs, and reading and writing of data.

In the presently preferred embodiment, the storage means is an integral part of the MMU's memory. The MMU's memory has four times the capacity than is needed to provide relocation base numbers and limit numbers for the entire main memory. As will be described, this additional capacity permits a form of "bank switching" and allows different processes to be run on the computer without reprogramming of the MMU memory.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a general block diagram illustrating a central processing unit, memory management unit (MMU) and main memory and their interconnections in a computer.

FIG. 2 is a diagram illustrating the organization of data stored in the memory of the invented MMU.

FIG. 3 is a block diagram of the invented MMU.

FIG. 4 is a diagram used to describe the different contexts used in the operation of the MMU and the resultant organization of information stored in the computer's main memory

DETAILED DESCRIPTION OF THE INVENTION

A memory management unit (MMU) is described for use in a digital computer which includes a central processing unit (CPU) and a main memory. In the following description, numerous specific details are set forth such as specific memory sizes, part numbers, etc., in order to provide a thorough understanding of the present invention. However, it will be obvious to one skilled in the art that these specific details are not required to practice the present invention. In other instances, well-known structures and circuits are not described in detail in order not to obscure the present invention in unnecessary detail.

Referring first to FIG. **1**, the coupling between an MMU, CPU and main memory is illustrated This coupling is somewhat the same for the present invention as it is for the prior art. The computer of FIG. **1** includes a bidirectional data bus **16** which communicates with the CPU **10**, main memory **14** and the MMU **12**. The address bus **18** receives address signals from the CPU **10** and communicates part of these addresses to the MMU **12** and part to the main memory **14**. Other control signals are coupled between the CPU **10** and MMU **12** as illustrated by lines **35** and **37** and between the MMU **12** and the main memory **14** as shown by line **57**.

The MMU **12** is programmed from the CPU **10** through the data bus **16**. Addresses are communicated over the bus **18** to the MMU from the CPU **10** to allow the loading of the MMU **12**.

In the presently preferred embodiment, the CPU **10** comprises a 68000 processor. For this processor, the CPU **10** provides 24 bit addresses (Actually, the lowest order bit is not physically present as such but encoded into other signals, however, for purposes of discussion, it will be assumed to be an ordinary address bit.) Also, for purposes of discussion, it will be assumed that the 7 highest order bits of each logical address from the CPU selects a segment in memory, the next 8 lesser significant bits comprise a page offset, and the least significant 9 bits, an offset.

In the presently preferred embodiment, the segment and page offset of each address are coupled to the MMU 12. The MMU provides a relocation base by exchanging the segment number from the CPU 10 with a segment number stored in the MMU 12. Specifically, the segment number from the CPU 10 addresses a memory within the MMU 12 and this memory provides a segment base used to address the main memory 14. The page offset portion of the address from the CPU 10 is checked to determine if the page offset falls within a predetermined range of the segment. This, for instance, would prevent the reading and interpreting as data, all zeros from an unused space in main memory. The segment base from the MMU along with the page offset are added and then coupled to the main memory 14 on the bus 18*a* and 18*b* of FIG. 1.

The 9 least significant bits are passed directly from the CPU to the main memory via bus 18c.

Referring now to FIG. 3, the presently preferred embodiment of the MMU includes an MMU memory 20. This memory is a random-access memory fabricated from commercially available MOS static RAMs. As currently implemented, three Part No. 2148 RAMs are used for memory 20, thus providing a total capacity of 12k bits. The organization of the MMU memory is discussed in greater detail, particularly in conjunction with FIG. 2.

The address from the CPU is shown as the 24 bit address (logical address) in the uppermost part of FIG. 3. The 7 most significant bits of this address are coupled to the MMU's memory via bus 18*a* and are used to address the MMU's memory. The next most significant bits (bus 18*b*) are coupled to an adder 27, and the least significant 9 bits (offset) are coupled via bus 18*c* to register 28. The output of the MMU's memory 20 consists of two 12-bit words (buses 22 and 23). These words are coupled through the multiplexer 25 to the 12-bit bus 30. One of the 12 bit words from the memory 20 provides the segment base from the stored relocation base. The second 12 bits consist of 8 bits for limit checking of the page offset and 4 additional bits which perform functions which are part of the present invention.

(In the presently preferred embodiment, multiplexer 25 does not physically exist, rather the output of memory 20 is time division multiplexed. However, for purposes of explanation it is easier to include the multiplexer 25.)

The multiplexer 25 is also used to load information from the bus 16 into the memory 20. The signal on line 47 from the access check logic 40 provides access to the memory 20 as do the signals on line 35. The signal on line 37 controls the multiplexing of data between either the bus 22 or the bus 23.

The 12 bit bus 30 from the multiplexer 25 is coupled to the adder 27. This adder also receives the 8 bits on bus 18*b*. As will be described, the adder 27 is used to determine if the page offset falls within a predetermined range of the selected segment. The adder 27 also combines the relocation (segment base) from the MMU's memory with the page offset to provide the 12 most

significant bits of the physical address These 12 bits along with the 9 bits from bus 18*c* are coupled to the register 28 to provide a 21 bit address which is communicated to the main memory 14. (The register 28 does not exist in the presently preferred embodiment, it is shown for purposes of explanation).

The 4 access check bits are coupled from the multiplexer 25 via line 45 to the access logic 40. Here the signals are decoded to provide main memory control and other control as follows: One bit controls the type of main memory access (1=read only, 0=read/write}. The second bit controls I/o access (1=4I/o, 0 =no I/o access}. The third bit controls main memory access (1=memory access, 0=no main memory access). The fourth bit controls stacking (1=stack segment - check for no overflow, 0=normal segment - check for overflow). The access check logic 40 is shown in FIG. 3 coupled to the main memory control via line 57 to control memory access and the type of accesses permitted (i.e., read or read/write). Logic 40 is coupled to adder 27 via the overflow/carry in lines and to memory 20 via line 47 to enable memory 20 access.

The specific access control bit pattern used in the presently preferred embodiment is shown below.

ACCESS CONTROL BITS

MEM/				ADDRESS
BITS	IO/	RO/	STK/	SPACE AND ACCESS
0	1	0	0	Main Memory - Read Only Stack
0	1	0	1	Main Memory - Read Only
0	1	1	0	Main Memory - Read/Write Stack
0	1	1	1	Main Memory - Read/Write
1	0	0	1	I/O Space
1	1	0	0	Page Invalid (segment not present)
1	1	1	1	Special I/O Space
Any other				Not allowed (unpredictable result)

Assume first that the memory 20 has been programmed from the CPU. For purposes of a first level explanation of the MMU's operation, the function of the 2 bits on lines 35 shall be ignored When the CPU addresses the main memory, the most significant 7 bits address the MMU's memory 20. The 12 bits from the relocation data segment are coupled via bus 22 and bus 30 to the adder 27. There they are combined with the page offset (bus 18*b*) and the resultant address is combined with the 9 bits of the offset in the register 28 to provide the final physical address. This portion of the MMU operates in a manner quite similar to prior art MMUs. Thus, the relocation segment base data can be programmed into the memory (ignoring line 35) in a manner well-known in the prior art.

The 12 bits forming the limit and access data are coupled via bus 23 through the multiplexer 25. The 8 bits of the limit data are coupled to the adder 27. The 4 bits of the access data are coupled to logic 40 via line 45 as discussed. The limit data in the presently preferred embodiment is stored in ones complement form in the memory 20 for a non-stacked segment. For stacked segments the limit stored is "length minus one" (e.g. a two page segment would be stored as 0000 0001 in memory 20.) When this limit data is added to the page offset in adder 27, the result of this addition determines whether or not the page offset falls within the predetermined range of the segment. This is an improvement over prior art limit checking where additional logic steps are required

NON-STACK EXAMPLE

Referring briefly to FIG. **4**, a representation of the computer's main memory **14** is illustrated. Assume that data is stored at locations **50**. Further assume that the highest page offset (1111 1111) for data **50** extends to location **52**, and that within this segment data extends to a page offset of 1110 0000 (line **51**). For this page offset, the ones complement of 1110 0000 (0001 1111) is stored in the memory **20** of FIG. **3**. If this segment is addressed, and assuming the page offset address is 1111 1111 (that is, into the free space of the memory), adder **27** adds 1111 1111 to the stored number 0001 1111. An overflow occurs from the adder **27** and this overflow condition is sensed by the logic **40** of FIG. **3**. For this example, an overflow indicates that the page offset is not within range and a signal is provided on line **57** to show that the address is in error. Logic **40** via line **57** prevents access to main memory and/or an error signal is generated.

Again referring to FIG. **4**, assume that a program is stored at locations **53** and that the highest page offset (1111 1111) for program **53** extends to location **50** which is outside of the actual program which ends at location **54**. If the page offset for location **54** is 0011 0000 then 1100 1111 is stored within the memory **20** of FIG. **3** for the segment which begins at location **55**. If this segment is addressed and the page offset is 0000 0001, (addressing the program) the adder **27** adds 1100 1111 and 0000 0001. This time no overflow occurs and no signal is communicated to the logic **40**, that is, access is permitted. Note that if the page offset is 0100 0000 (not within range) when this number is added to the stored number of 1100 1111 an overflow occurs. This overflow indicates to the logic **40** that the page offset is not in range and memory access is disabled.

STACK EXAMPLE

For some programming languages (e.g. Pascal) stacks (in memory) are very desirable. Stacks can be formed by moving data up in memory, albeit time consuming. Stacks with the presently described system are permitted to grow down in memory with a different limit checking procedure.

Assume a one page stack segment. The limit number stored in memory **20** as the one's compliment of the page offset (1111 1111→0000 0000) which is the same as the size minus one (0000 0000→0000 0000). The access check bits causes the logic **40** to provide a carry-in of one. If the page offset is 1111 1111, an overflow occurs. This overflow is sensed by logic **40**, and interpreted as a valid (within range) condition. If the page offset were 1111 1110 (stack grown too much), no overflow occurs and this is interpreted as an out of range address.

Similarly, if the stack is a two page segment, 0000 0001 is stored in memory **20**. Again the carry in is set to a one. A page offset of 1111 1110 would result in an overflow indicating an in range address, whereas with a page offset of 1111 1100 no overflow would occur, indicating an out of range address.

FIG. 4 EXAMPLE

Referring again to FIG. **4**, assume that a process (program and data) is stored in the main memory **14** between the locations 0 and 500 KB. The 3 remaining access bits in the memory **20** corresponding to the segment addresses for locations 0–500 KB are used to provide special control, as mentioned. For instance, for those segments containing only program, only reading of the memory is allowed. This, of course, prevents the inadvertent writing into program. Both reading and writing into the segments which contain data may be permitted. This is indicated to the right of program **59** and data **60** in FIG. **4**.

The memory **20** is programmed (i.e., access check bits) to prevent reading of some segments of the main memory except in certain modes (e.g., supervisory mode). This is done, for instance, to prevent a user from reading and then copying an operating system. Referring briefly to FIG. **4**, when the program **59** is being run, no access to memory **20** is permitted since such access could cause the relocation base, limit data or access data to be inadvertently altered. Thus, the four access bits provide protection for the program stored within the main memory and also limit access to certain information stored in the memory. In a typical application, an operating system is loaded from a disk into the main memory. Once in the main memory, the CPU can access the operating system in supervisory modes, however, the user is prevented from accessing and hence copying the operating system.

With the present invention, the memory **20** has four times the capacity than is actually needed to provide a relocation base, and limit and access data for the main memory. The signals from the CPU on lines **35** allow the selection of each quadrant of the memory **20**. Each of these quadrants are referred to as a context (context 0–3) in the following description.

Referring to FIG. **2**, the organization of the MMU memory **20** is illustrated as four separate quadrants: **20***a* (context **0**), **20***b* (context **1**), **20***c* (context **2**) and **20***d* (context **3**). Context **1**,**2** and **3** are each organized in a 256×12 bit arrangement (128×12 bits for the relocation base and 128×12 bits for the limit and access data). Context **0** is selected by the CPU during the supervisory mode and this context stores management data relating to the operating system. It should be noted that each context is capable of storing information covering the entire main memory, thus there are three overlapping MMU memories for user processes.

The value of having these overlapping memories is best illustrated in FIG. **4**. The main memory **14** is shown programmed with three processes, P**1**, P**2** and P**3**. Process **1** is stored between 0 and 500 KB, process **2** between 600 KB and 1 mB and process **3** between 1.2 mB and 1.5 mB. Data relating to the operating system is stored between 1.8 mB and 2 mB. Assume first that the operating system is loaded into memory and is stored between 1.8 mB and 2 mB. An appropriate relocation base is stored within the memory **20** such that during supervisory modes, the addresses 0–200 KB automatically select 1.8 mB through 2 mB in the main memory. Also, the appropriate limits are loaded to assure that during the supervisory mode, the free space in the memory is not accessed. During the supervisory mode (context **0**) as indicated in FIG. **4** under the heading context **0**, complete access to the MMU memory and main memory is possible (except for access bits which prevent the writing into the operating system stored in main memory thereby protecting the program from damage due to a program error). Since the MMU memory is accessible at this time, it can be programmed through the bus **16** as indicated in FIG. **3**, and as previously discussed.

Assume that context **1** is to be used for program **59** and data **60**, one quadrant of the MMU's memory **20** corresponding to context 1 is programmed to indicate the location of program 59 and data 60. The limit and access bits are set as indicated under context 1. Thus, when context 1 is selected, program 59 can be read (only) and, reading and writing of data 60 is permitted. No other access to other memory locations is possible nor can the MMU memory be written into.

A second process can be stored in memory. The operating system knows the location of the first process and can program another quadrant of memory 20 for process 2. The relocation base is programmed such that when the CPU addresses locations corresponding to 0–400 KB, locations 600 KB to 1 mB, are provided to the main memory. As indicated under the heading context 2 in FIG. 4, the access bits are programmed to allow reading and writing into the data 50 and reading-only of the program 53. Also, no access (for writing) to the MMU memory is permitted, nor is access permitted to other locations in the main memory. Similarly, a third process can be stored in the main memory for context 4 as indicated in FIG. 4.

The advantage to the arrangement of FIG. 4 is that three separate processes are stored within the main memory and that each process may be easily selected through the MMU's memory, that is, by selecting context 1, 2 or 3. A separate context (context 0) is reserved as a starting point for the operating system, in the presently preferred embodiment, as discussed. This allows running of three separate programs without any reprogramming of the MMU's memory. This versatility is achieved because of the overlapping memory management capacity of the MMU's memory.

Thus, an improved memory management unit has been described which allows a plurality of programs to be run without reprogramming of the computer's MMU memory. The improved unit also limits access to certain types of data and prevents inadvertent writing into programs.

We claim:

1. In a computer system which includes a central processing unit (CPU and a computer main memory, a memory management unit (MMU) coupled to said CPU and said main memory for translating a logical address from said CFU to provide a physical address for accessing said main memory, comprising:

a MMU memory for storing a plurality of relocation base addresses, wherein said relocation base addresses are segmented into sections of memory (contests) such that each said context has at least one relocation base address associated therewith;

each said relocation base address having corresponding limit bits and access bits associated therewith, said limit bits and access bits also store said MMU memory;

said MMU receiving a control signal from said CPU for selecting a predetermined one of said contexts when said logical address is provided by said CPU;

said MMU memory for receiving a first portion of said logical address from said CPU and said first portion of said logical address accessing a stored relocation base address of a selected context and corresponding to said limit and access bits;

an adder coupled to said MMU memory for receiving said accessed relocation base address of said selected context and combining it with a second portion of said logical address to output said physical address for accessing said main memory;

said adder also coupled to receive said limit bits corresponding to said accessed relocation base address and adding it to said second portion of said logical address and generating an indication signal if said second portion of said logical address exceeds a value set by said limit bits;

access check logic means coupled to said MMU memory and said adder for receiving said access bits corresponding to said accessed relocation base address and determining if said access bits permit access of said main memory for a type of access requested by said CPU and generating a fault signal to prevent access of said main memory if an illegal access of said main memory is attempted;

said access check logic means also generating said fault signal if said indication signal is received from said adder;

each said relocation base address for pointing to a corresponding mapped base address in said main memory, such that a given logical address is mapped into a plurality of physical addresses, wherein at least one physical address is provided for each context; and

wherein selected physical addresses of said main memory can be accessed by more than one context.

2. The MMU defined by claim 1 wherein one of said MMU memory contexts is selected as a supervisory context when said CPU is in a supervisory mode, such that said supervisory context accesses all of said main memory.

3. The MMU defined by claim 2 wherein said adder receives said limit number which is a binary complement of an offset from its relocation base address, such that when said binary complement is added to said second portion of said logical address said indication signal is generated when an overflow occurs from said adder.

4. The MMU defined by claim 3 wherein said MMU memory stores said relocation base addresses, said limit bits, and said access bits from said CPU during a MMU program cycle.

5. In a computer system which includes a central processing unit (CPU) and a computer main memory, a memory management unit (MMU) coupled to said CPU and said main memory for translating a logical address from said CFU to provide a physical address for accessing said main memory, an improvement comprising:

a MMU memory for storing a plurality of relocation base addresses, wherein said relocation base addresses are segmented into sections of memory (contexts) such that each said context has at least one relocation base address associated therewith;

each said relocation base address having corresponding limit bits and access bits associated therewith, said limit bits and access bits also stored in said MMU memory;

said MMU receiving a control signal from said CPU for selecting a predetermined one of said contexts when said logical address is provided by said CPU;

said MMU memory for receiving a first portion of said logical address from said CPU and said first portion of said logical address accessing a stored relocation base address of a selected context and corresponding of said limit and access bits;

an adder coupled to said MMU memory for receiving said accessed relocation base address of said selected context and combining it with a second portion of said logical address to output said physical address for accessing said main memory;

said adder also coupled to receive said limit bits corresponding to said accessed relocation base address and adding it to said second portion of said logical address and generating an indication signal if said second portion of said logical address exceeds a value set by said limit bits;

access check logic means coupled to said MMU memory and said adder for receiving said access bits corresponding to said accessed relocation base address and determining if said access bits permit access of said main memory for a type of access requested by said CPU and generating a fault signal

to prevent access of said main memory if an illegal access of said main memory is attempted;

said access check logic means also generating said fault signal if said indication signal is received from said adder;

each said relocation base address for pointing to a corresponding mapped base address in said main memory, such that a given logical address is mapped into a plurality of physical addresses, wherein at least one physical address is provided for each context; and

wherein selected physical address of said main memory can be accessed by more than one context.

* * * * *

United States Patent [19]

Fitch et al.

[11] **Patent Number: 4,931,923**

[45] **Date of Patent: * Jun. 5, 1990**

[54] **COMPUTER SYSTEM FOR AUTOMATICALLY RECONFIGURATING MEMORY SPACE TO AVOID OVERLAPS OF MEMORY RESERVED FOR EXPANSION SLOTS**

[75] Inventors: **Jonathan Fitch**, Cupertino; **Ronald Hochsprung**, Saragota, both of Calif.

[73] Assignee: **Apple Computer, Inc.**, Cupertino, Calif.

[*] Notice: The portion of the term of this patent subsequent to Feb. 27, 2007 has been disclaimed.

[21] Appl. No.: **25,499**

[22] Filed: **Mar. 13, 1987**

[51] **Int. Cl.**^{5} **G06F 9/02**; G06F 9/06; G06F 13/00; G06F 13/10

[52] **U.S. Cl.** **364/200**; 364/232.7; 364/238.3; 364/238.4; 364/232.8; 364/240; 364/245; 364/245.2; 364/245.31

[58] **Field of Search** ... 364/200 MS File, 900 MS File

[56] **References Cited**

U.S. PATENT DOCUMENTS

3,675,083	7/1972	White	361/413
3,710,324	1/1973	Cohen et al.	340/172.5
3,993,981	11/1976	Cassarino, Jr. et al.	340/172.5
4,000,485	12/1976	Barlow et al.	340/172.5
4,250,563	2/1981	Struger	364/900
4,368,514	1/1983	Persaud	364/200
4,467,436	8/1984	Chance et al.	364/200
4,633,402	12/1986	Flinchbaugh .	

FOREIGN PATENT DOCUMENTS

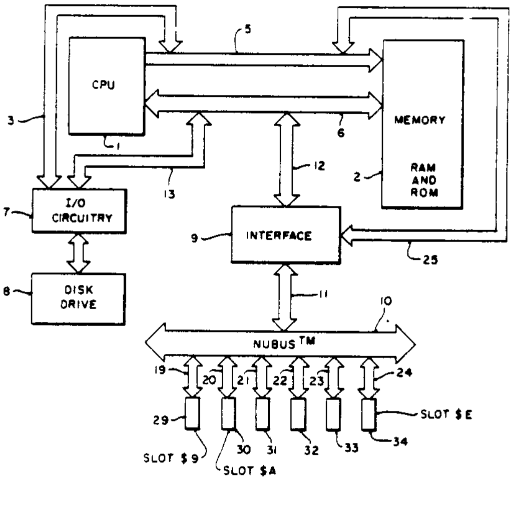
1380776	1/1975	Fed. Rep. of Germany .
2060961	5/1981	United Kingdom .
2101370	1/1983	United Kingdom .
2103397	2/1983	United Kingdom .

Primary Examiner—Archie E. Williams, Jr.
Assistant Examiner—Emily Y. Chan
Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman

[57] **ABSTRACT**

A personal computer system includes a main circuit board having a central processing unit and expansion slots each of which is adapted to receive a printed circuit board card. The main circuit board further includes memory, a 32-bit address bus with control signals associated therewith, and input/output circuitry. The slot is coupled to the 32-bit address bus, which is substantially a NUBUS bus, and the slot includes distinct identification line means which provide the slot with an identification number (distinct number) in the computer system. The computer system reserves 256-megabytes of memory space ranging from location $X000 0000 to location $XFFF FFFF for memory on a card in a slot having a distinct number equal to $X.

5 Claims, 7 Drawing Sheets



NuBus

(Macintosh 21

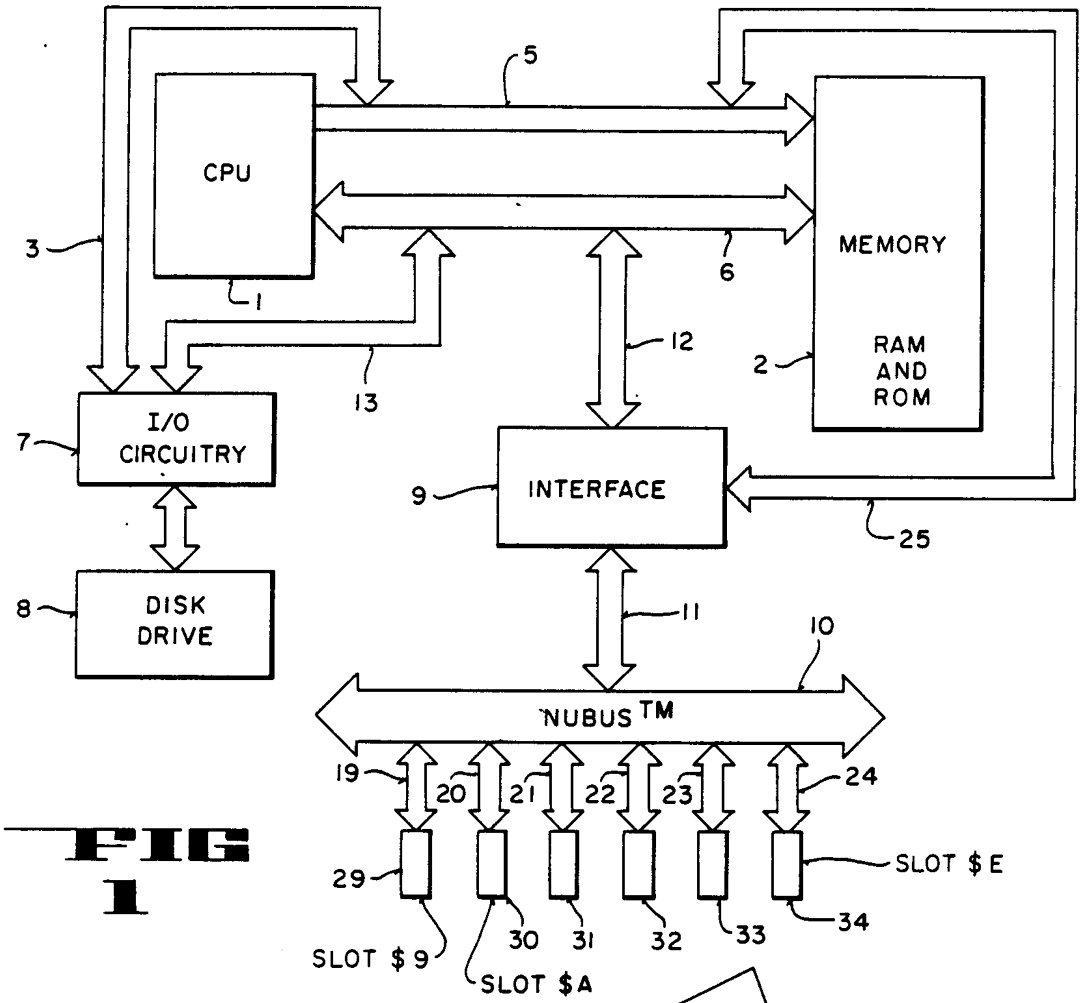

5
CPU
1
3
6
MEMORY
RAM
AND
ROM
2
12
13
7
I/O
CIRCUITRY
9
INTERFACE
25
8
DISK
DRIVE
11
10
NUBUS TM
19
20
21
22
23
24
29
SLOT $ E
34
33
32
31
30
SLOT $ 9
SLOT $A

FIG 1

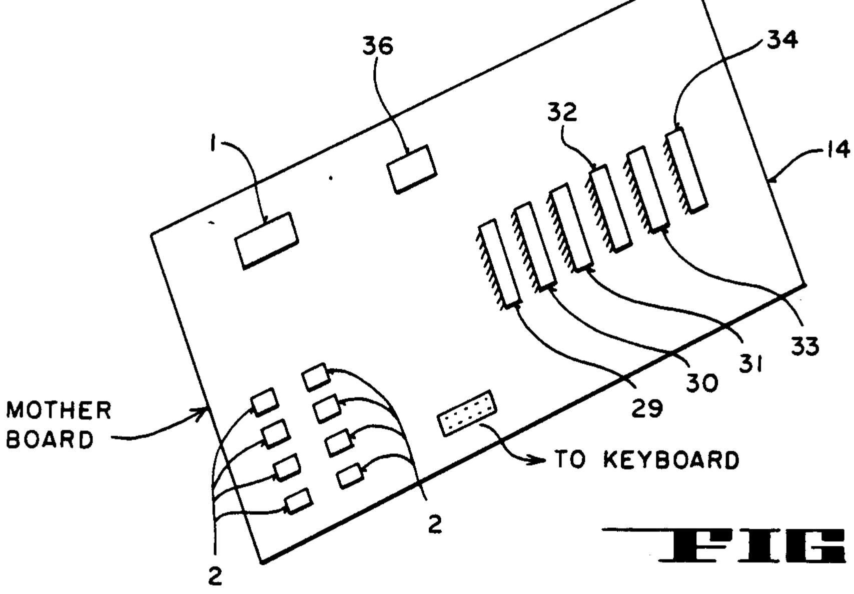
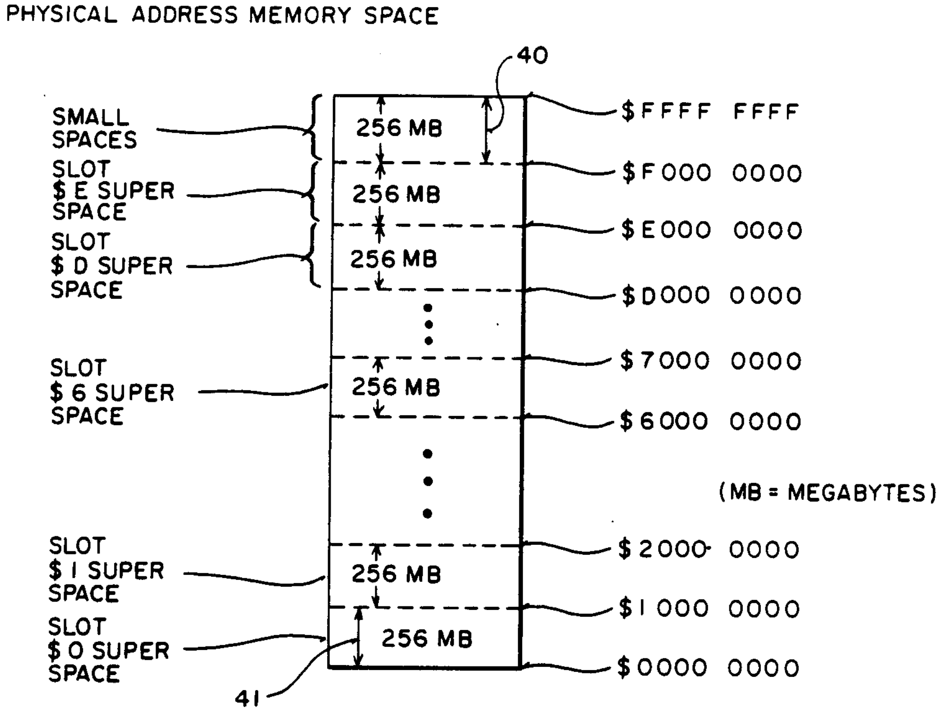

36
34
32
1
14
33
31
30
29
MOTHER
BOARD
TO KEYBOARD
2
2

FIG _ 11

FIG 2



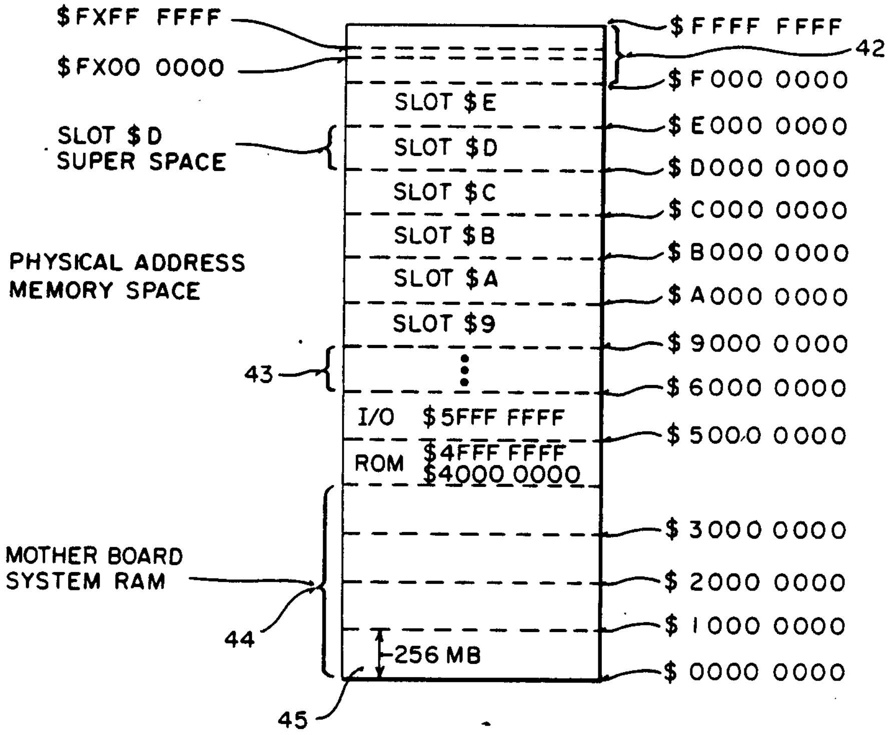
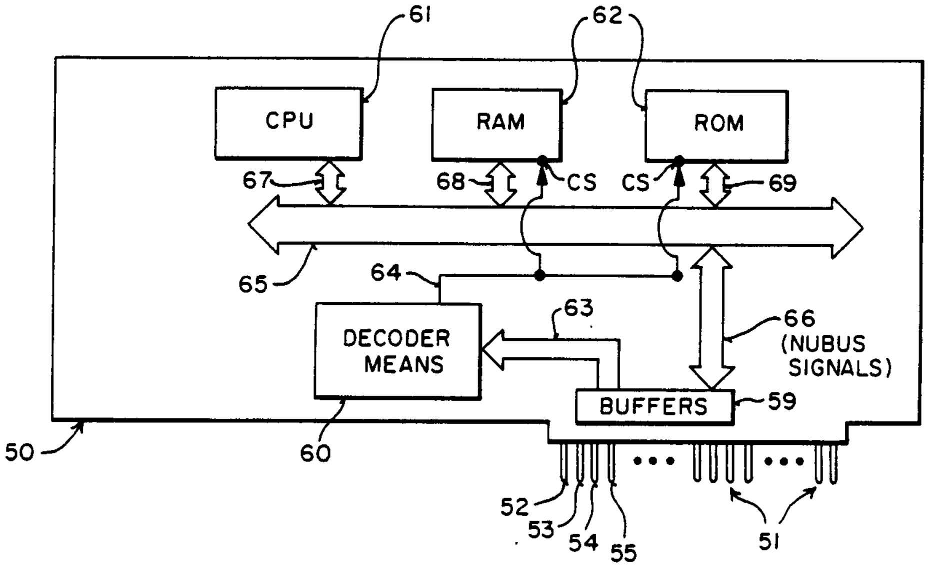


FIG 3

“APPLE_PAT_4_931_923_03” 115 KB 2000-02-23 dpi: 300h x 300v pix: 1762h x 2823v

FIG_4 CARD



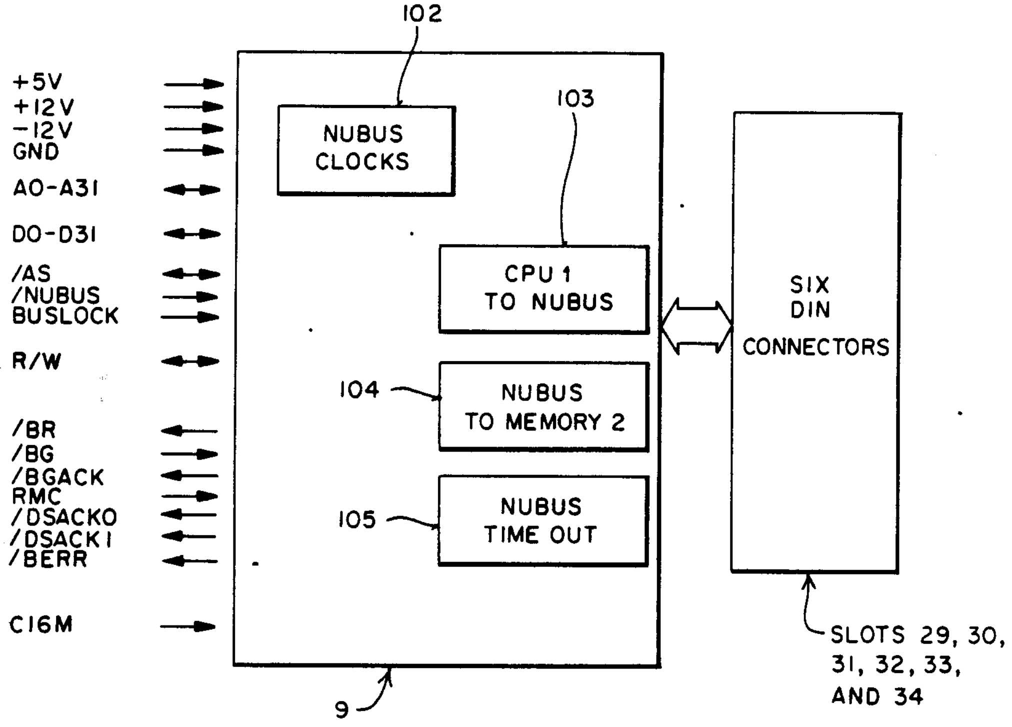


FIG_5 NUBUS INTERFACE BLOCK DIAGRAM

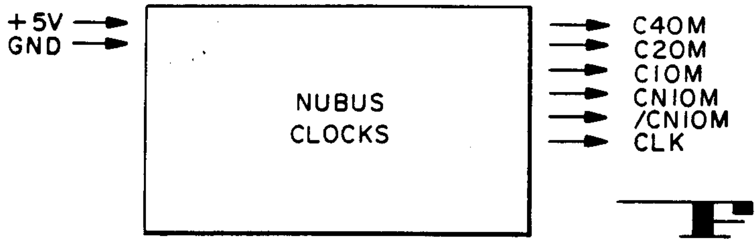


FIG _ 6

NUBUS CLOCKS BLOCK DIAGRAM

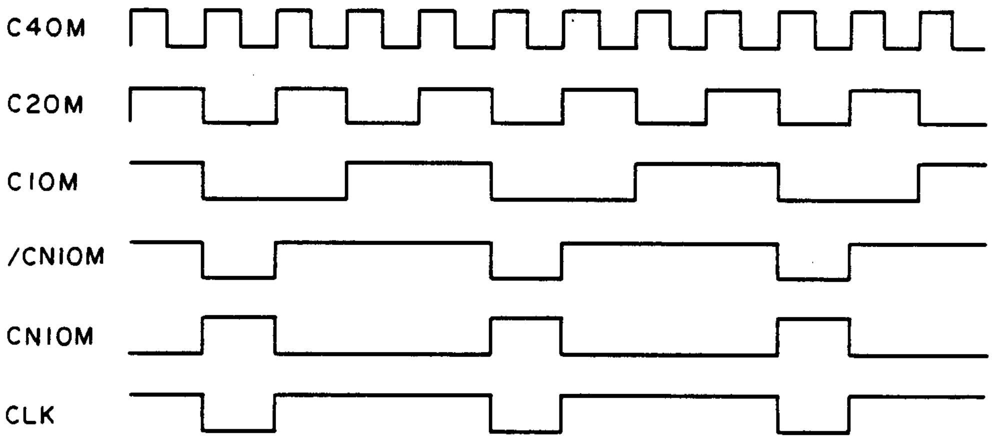


FIG _ 7

PHASE RELATIONSHIP OF NUBUS CLOCKS

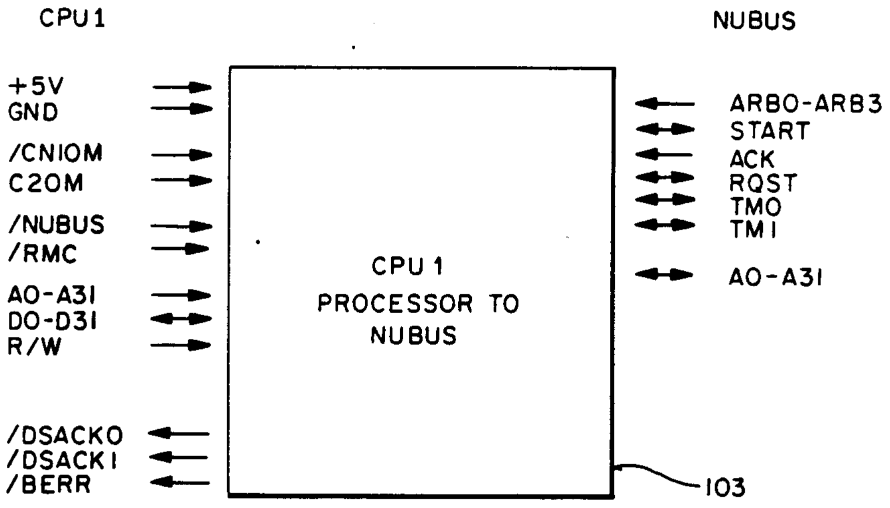


FIG _ 8

PROCESSOR TO NUBUS BLOCK DIAGRAM

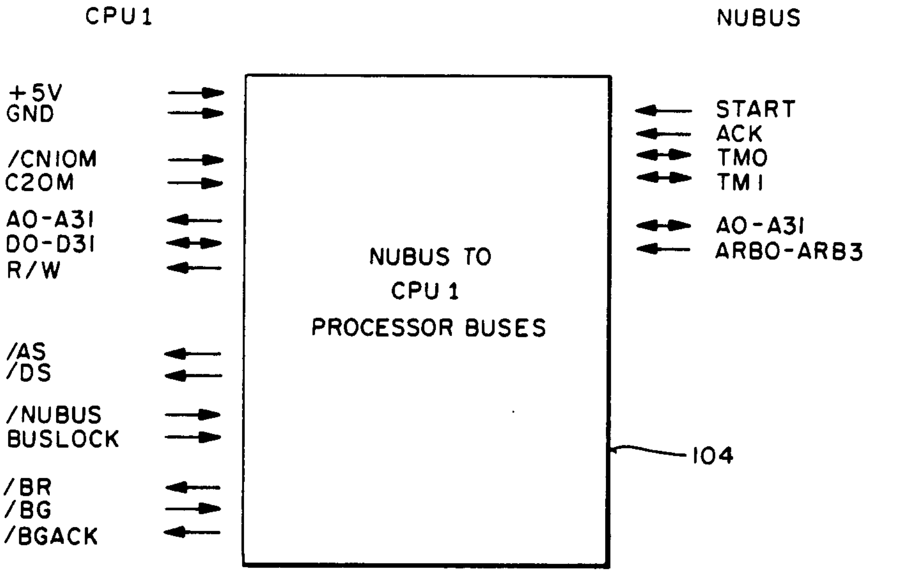


FIG_9 NUBUS TO PROCESSOR BUS BLOCK DIAGRAM

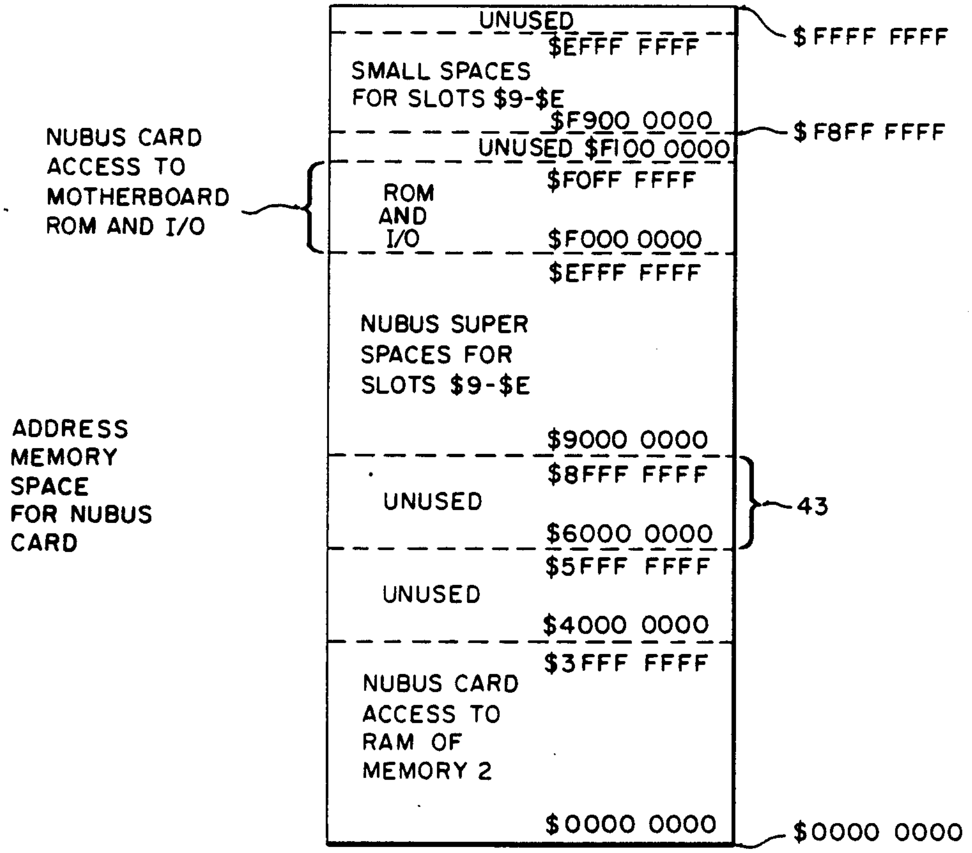


FIG_10

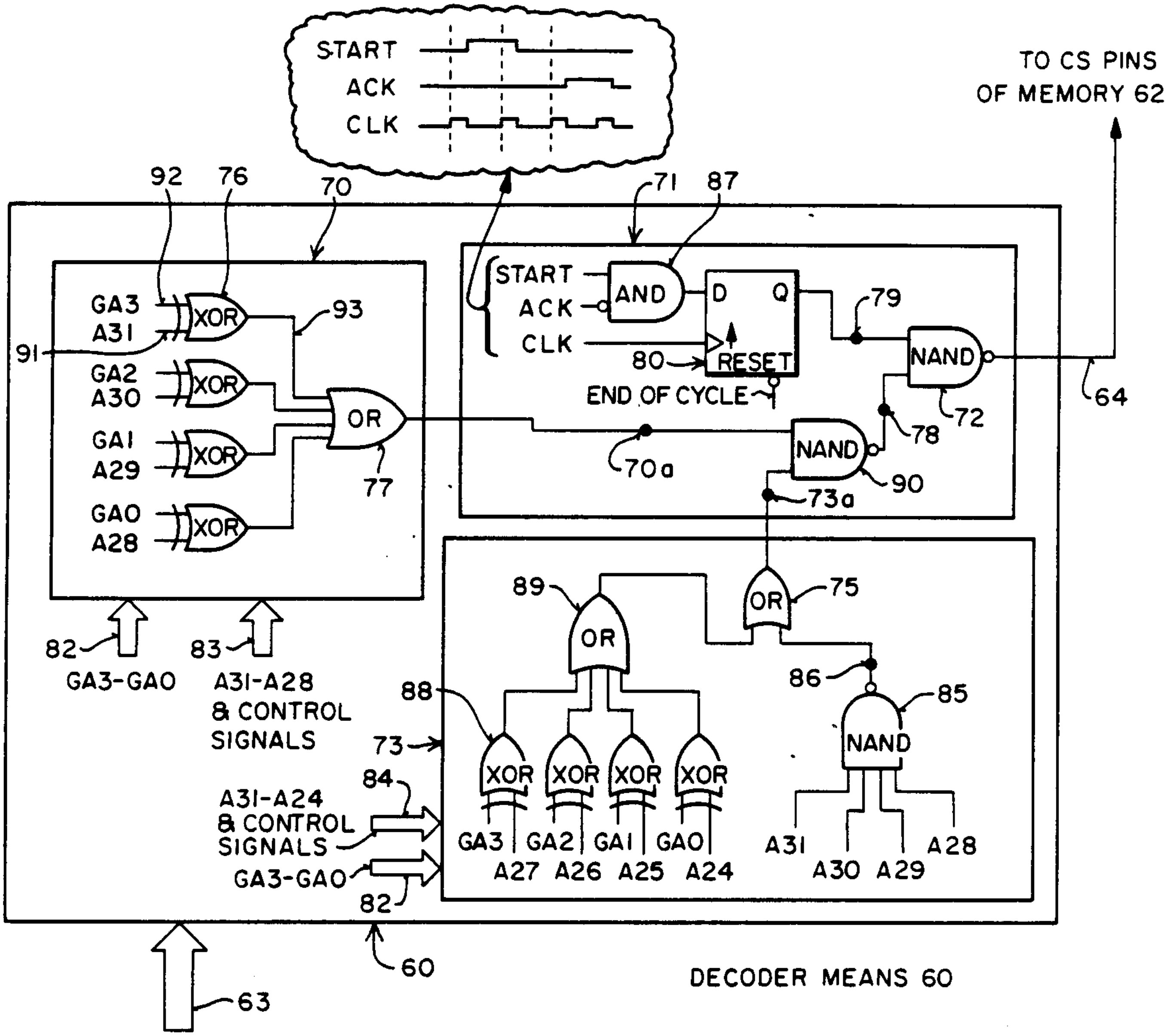

START
ACK
CLK
TO CS PINS OF MEMORY 62
92
76
70
71
87
GA3
A31
XOR
93
91
GA2
A30
XOR
GA1
A29
XOR
GA0
A28
XOR
OR
77
START
ACK
CLK
AND
D
Q
80
RESET
END OF CYCLE
79
NAND
72
64
78
70a
NAND
90
73a
82
GA3-GA0
83
A31-A28 & CONTROL SIGNALS
89
OR
OR
75
86
85
88
NAND
73
XOR
XOR
XOR
XOR
84
A31-A24 & CONTROL SIGNALS
GA3-GA0
82
GA3
A27
GA2
A26
GA1
A25
GA0
A24
A31
A30
A29
A28
60
63
DECODER MEANS 60

FIG_12

FIG_13

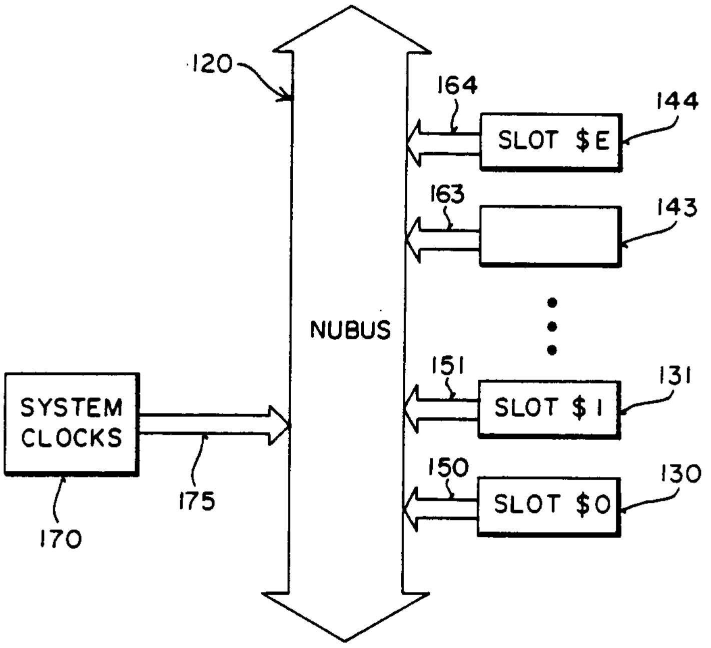
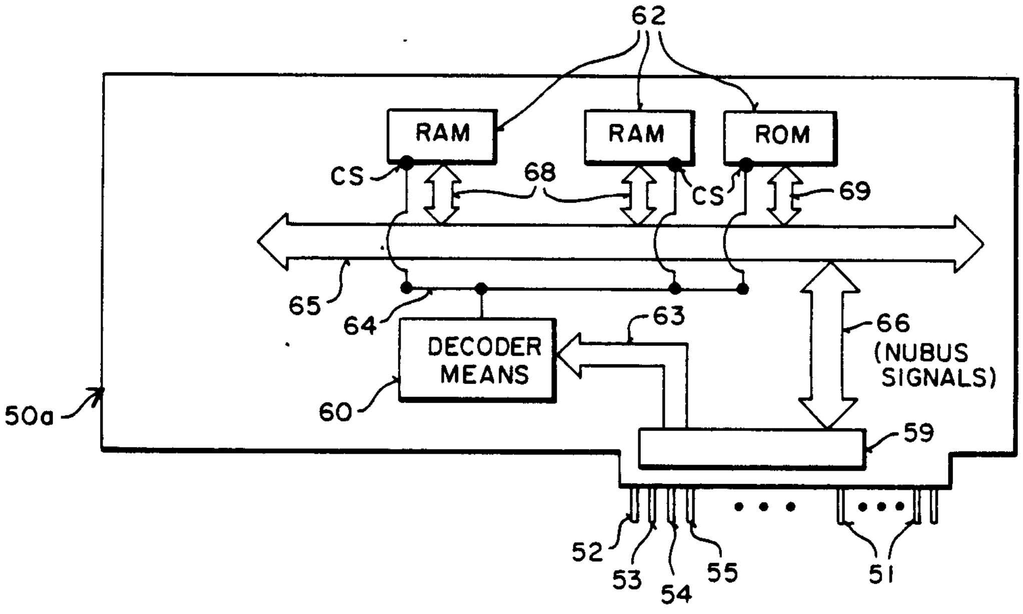


FIG_14 CARD



“APPLE_PAT_4_931_923_08” 80 KB 2000-02-23 dpi: 300h x 300v pix: 1762h x 2595v

COMPUTER SYSTEM FOR AUTOMATICALLY RECONFIGURATING MEMORY SPACE TO AVOID OVERLAPS OF MEMORY RESERVED FOR EXPANSION SLOTS

BACKGROUND OF THE INVENTION

1. Field of Invention

This invention relates generally to computer system having expansion slots on a mother board (main circuit board) and more specifically, to personal computers including such slots and printed circuit board cards which are adapted to fit in such slots which are connected to a bus, where a portion of the address memory space in the computer is reserved for the slots.

2. Prior Art

Computer systems having expansion slots are well known in the prior art. For example, the Apple IIe is a well known personal computer having expansion slots; memory is reserved for the slots in that computer. However, the memory of a card in that computer is accessed not by first presenting the address but rather by selecting a particular pin in the slot (along with the address) which tells the card in the slot that the address which the microprocessor is calling for is somewhere in that peripheral card's reserved memory. Moreover, the reservation of memory space for cards in these systems is relatively small (e.g. 16-bytes or 256-bytes). That is, the address itself is usually not used alone to indicate when a card's address space is being addressed. Various references are available to one with ordinary skill in the art concerning the general nature of these computer systems. For example: *The Apple II Reference Manual*, Apple Computer (1981); *From Chips to Systems: An Introduction to Microprocessors*, Rodnay Zaks, Sybex, Inc., 1981; *An Introduction to Microcomputers*, by Adam Osborne and Associates, 1975; and *The Apple II Circuit Description*, Winston Gayler, published by Howard W. Sams & Co., Inc. (1983).

This invention relates more specifically to computer systems using systems buses which follow substantially NUBUS Tm (a trademark of Texas Instruments) bus specifications, which specifications describe the protocols (e.g. logical, electrical and physical standards) and general standards of a sychronous (10 Mhz), multiplexed, multimaster bus which generally provides a fair arbitration mechanism. NUBUS bus originated at the Massachusette Institute of Technology. It has subsequently been revised and exists as published in certain publications of Texas Instruments, Inc. (including Texas Instruments publication number 2242825-0001 and Texas Instrument publication number 2537171-0001). Recently, a committee of the Institute of Electrical and Electronic Engineers (IEEE) has proposed specifications for a system bus, as an IEEE standard, that is substantially a NUBUS bus, although it has been modified from the specifications published by Texas Instruments. The proposed IEEE bus is referred to as the IEEE 1196 Bus. A copy of the proposed specification for the IEEE 1196 Bus (Draft 2.0) is provided with this application for whatever reference may be necessary by one of ordinary skill in the art. The IEEE 1196 Bus is substantially a NUBUS bus as originally specified in Texas Instruments'publications.

In a NUBUS system, there are 4-gigabytes of physical memory address space since there is a 32-bit address bus which may be coupled to a CPU capable of generating 2^{32} different addresses. In its simplest form, a computer

utilizing the NUBUS bus architecture is essentially a main circuit board having slots into which one place cards (sometimes referred to as modules) having microprocessors, memory and other circuitry generally associated with microcomputers. In effect, each card may itself be a microcomputer which communicates through NUBUS bus to other cards in other slots which are also connected to NUBUS bus. Thus, for example, a NUBUS bus system may include a card having a CPU (central processing unit) microprocessor, a memory management unit, some memory in the form of random access memory (RAM) and read only memory (ROM), and a bus on the card which permits the microprocessor on the card to read the ROM on the card and to read from and write to the RAM on the card. In addition input and output (I/O) circuitry may be included on the card, which circuitry permits the card to communicate through terminals on the card with parts of the rest of the system, including peripheral units such as disk drives, printers, video systems and other peripheral units. The card typically has an edge which includes electrical terminals in the form of pins designed to make electrical connections with cooperating terminals in a slot. Such a card, having a microprocessor, would be capable of mastership of the NUBUS bus by executing certain signals to initiate a NUBUS bus transaction and thereby to transfer and receive information over the NUBUS bus on the main circuit board. Thus, that card could write information to memory located on other cards through NUBUS bus (a transaction) and read that information through NUBUS bus (another transaction).

In the NUBUS bus system, memory is reserved for each of the slots. In the NUBUS bus system, there can be up to 16 slots which are allocated memory space in the upper 1/16 of the entire 4-gigabyte NUBUS bus address space. That upper 16th is 256-megabytes of memory space, and it is divided into 16 regions of 16-megabytes which are mapped to the 16 possible NUBUS bus card slots based on a slot identification number which produces a distinct number at each slot, allowing a card in the slot to "read" the distinct identification number to determine the slot number of the slot into which the card is plugged. See, generally, pages 30–32 of the proposed specification of the IEEE 1196 Bus. Thus, each card gets a "slot space" of 16-megabytes. In the conventional NUBUS bus system, a card's "slot space" is reserved by a device on the card which matches the distinct number (expressed in hexadecimal) of the slot (where the card is) to the second most significant hexadecimal digit (2nd MSHD) of an address appearing on the NUBUS bus, when the address's most significant hexadecimal digit (MSHD) is $F. Thus, the device determines when MSHD equals $F and then determines if the slot number (slot identification number) matches the 2nd MSHD; if there is a match, then the device permits the card to be addressed. Of course, the actual comparison by the card is done in binary, but for purposes of explanation, it is easier to consider the comparison as if it were done in hexadecimal.

This NUBUS bus system provides for considerable flexibility because the vast majority of the memory address space is unreserved. Moreover, the seemingly large (16-megabytes) spaces reserved for the slots (the slot spaces) provide considerable data storage ("data" is used herein to include computer programs). However, too much flexibility fosters incongruities between cards which may be used on the same mother board. That is, this flexibility permits one to design a card which reserves most of the remaining address space in the NUBUS bus system which card would compete with another card developed to use a portion of the same memory space. Of course, switches and jumper cables may be utilized to configure the system to prevent over laps of memory space; however, such solutions are cumbersome in many ways, including their tendancy to frighten novices who would prefer a computer system that permits the user to simply plug the card into a slot and not worry any further.

The present invention solves these problems by allocating automatically 1/16th of the entire memory address space to each slot in the NUBUS bus system. Thus, it is an object of the invention to provide a system which configures itself and which is still flexible but which does not penalize the user because of its flexibility. It is a further object of the invention to provide a main circuit board (mother board) having slots which allow greater automatic computer power due to increased memory space for each card. It is a further object of the invention to provide printed circuit board cards (modules) which automatically configure to their memory space and have increased memory space reserved for each of the cards.

This invention invloves a computer system which has expansion slots coupled to a NUBUS bus, which slots have increased memory space available for and reserved for memory on cards (modules) in the expansion slots and where the reservation of the increased memory occurs by use of distinct identification line means which provides, via a distinct signal, a distinct number identifying the slot number to any card located in the slot. Moreover, the invention provides a card having a decoder means which is coupled to receive the distinct signal provided by the distinct identification line means. A decoder means compares the distinct number provided by the distinct signal to an address appearing on NUBUS bus. The comparison results in 256-megabytes of memory space being reserved for the card in a slot where the memory space ranges from $X000 0000 to $XFFF FFFF, where the slot number is X.

The decoder means compares the distinct number to the most significant hexadecimal digit of the address appearing on the NUBUS bus to determine whether the distinct number, in hexadecimal, is equal to the most significant hexadecimal digit in the address. When the decoder means determines they are equal, it enables any memory on the card to be addressed based on the address appearing on the NUBUS bus. The comparison, of course, is done in binary, but for purposes of explanation, it is easier to consider the comparison process as if it were done in hexadecimal.

BRIEF DESCRITPION OF THE DRAWINGS

FIG. 1 is a block diagram of a general computer system of a preferred embodiment of the invention where there are 6 slots coupled to the NUBUS bus **10**.

FIG. 2 is a map of the physical address memory space of an embodiment of the invention.

FIG. 3 is a physical address memory space map showing the memory space allocation for a preferred embodiment of the invention.

FIG. 4 shows a printed circuit board card of the invention which is intended for use with the mother board of the invention.

FIG. 5 is a block diagram showing the NUBUS bus interface with a microprocessor on the main circuit board.

FIG. 6 is a block diagram showing the various NUBUS bus clocks designed for use with the NUBUS bus.

FIG. 7 shows the phase relationship of the various NUBUS bus clocks.

FIG. 8 is a block diagram of the interface between the mother board processor (CPU 1) and NUBUS bus cards in NUBUS bus slots.

FIG. 9 is a block diagram showing the NUBUS bus to mother board processor bus interface.

FIG 10 shows an address memory space allocation as seen by a card in a NUBUS bus slots where the card accesses the ROM portion of memory 2 by addressing the upper portion of the small space for slot 0.

FIG. 11 is a perspective view of the main circuit board (mother board) of a computer system according to the invention.

FIG. 12 is a schematic diagram of an exemplary decoder means utilized on a card according to the invention.

FIG. 13 is a block diagram of a computer system according to the invention.

FIG. 14 shows a printed circuit board card of the invention which is intended for use with the main circuit board of the invention.

DETAILED DESCRIPTION OF THE INVENTION

In the following description, numerous specific details are described and shown, such as circuits, block diagrams, memory locations, logic values, etc. in order to provide a thorough understanding of the present invention. However, it will be obvious to one skilled in the art that the present invention may be practiced without these specific details. In other instances, well known components and sub-systems are not described in detail, in order not to unnecessarily obscure the present invention.

FIG. 1 shows the general structure of a computer system according to the present invention. The system includes a central processing unit 1 (CPU 1), which is usually a microprocessor, and which is coupled to memory 2 to permit the CPU 1 to read data from the memory 2 and write data into the memory 2. The CPU 1 is coupled to the memory 2 to provide addresses of memory locations via the processor bus **5**, which acts as an address bus and provides addresses to the memory 2 from the CPU 1. Data (which includes computer program instructions) from the addressed memory locations is provided by the memory 2 into the processor bus **6** which acts as a bidirectional data bus. The CPU 1 may write to the memory 2 by first providing an address over the processor bus **5** which addresses memory locations in the memory 2 according to the address signals over the processor bus **5** and then writing to the memory 2 by providing data over the processor bus **6** to the memory 2. As is well-known, certain signals from the CPU 1, which may be carried over the processor bus **5**, indicate whether the CPU 1 is writing to the memory 2 or reading from the memory 2. The processor bus **5** is a 32-bit address bus and thus includes 32 address lines which provide the address signals. The processor bus **5** further includes control signals (e.g. R/W (read/write) and Chip Select) which indicate whether the CPU 1 is reading (from the memory) or writing (to the memory) and other associated control signals, including control signals for the particular microprocessor being used and timing signals (e.g. column address strobes and row address strobes) as is well-known in the prior art and therefore is not discussed herein in greater detail. The processor bus 6 includes a 32-bit data bus (and thus 32 data lines which provide the data signals) and associated control signals for the particular microprocessor being used which are typically included with data buses, as is well-known in the prior art (e.g. write enable signal, etc.). The CPU 1 according to the invention includes an address generation means for generating 2^{32} different addresses ranging from location $ 0000 0000 to location $FFFF FFFF (the dollar sign indicates hexadecimal notation); that address generation means is typically coupled to the processor bus 5 and is part of the CPU 1, such as the microprocessors 68020 (Motorola) and 80386 (Intel).

The computer system also includes input and output circuitry which, as is well known in the prior art, is used to interface the computer to receive data from and provide data to peripheral units. The details of this circuitry as well known. Input/output (I/O) circuitry 7 is coupled to the CPU 1 and the memory 2 via the interconnect bus 13 and the processor bus 6 and the processor bus 3. The I/O circuitry 7 may be utilized to provide access to peripheral devices, such as disk drives, printers, modems, video displays and other peripheral units for use with the computer system. As shown in FIG. 1, a disk drive 8 is coupled to the I/O circuitry by an interconnect bus shown between the I/O circuitry 7 and the disk drive 8. The I/O circuitry 7 is coupled to the memory 2 through the processor bus 6 to provide data to the memory and to receive data from the memory and from the CPU 1; the bus 3 allows the CPU to address the peripheral units attached to the I/O circuitry 7 and allows the I/O circuitry 7 to address the memory 2. The I/O circuitry 7 is also coupled to the CPU to receive data and control signals from the CPU 1. Thus, the peripheral units, such as the disk drive 8, can exchange data (which includes programs) with the CPU 1 and the memory 2; it can also exchange data with any cards and the slots coupled to the NUBUS bus 10 such as slot 29 which has a distinct number, $9, in the computer system shown in FIG. 1.

In a typical transaction the CPU 1 provides an address over the bus 5. The memory 2, which is coupled to the bus 5. receives the addresses and provides a value over bus 6 based on the location addressed according to the address provided on the bus 5. The data from memory 2 is provided over the processor bus 6 to the CPU 1. Memory 2 typically includes RAM and may further include ROM (read only memory). The processor bus 6 is coupled to the NUBUS bus 10 through the interface 9 and interconnect buses 11 and 12.

The computer system shown in FIG. 1 includes six "expansion" slots which are designed to receive printed circuit board cards and to make electrical connections with circuitry on the cards, such as cards 50 and 50*a* in FIGS. 4 and 14 respectively. That system includes slots 29, 30, 31, 32, 33 and 34 which are each coupled to another system bus, NUBUS bus 10, on the other board. Thus, slot 29 is coupled to NUBUS bus 10 via the interconnect bus 19. Each of the slots includes cooperating terminals, each of which is electrically coupled to a particular signal line of the NUBUS bus 10 through the interconnect buses; thus, each of the slots 29, 30, 31, 32, 33 and 34 includes a set of cooperating terminals which

provide electrical connections to the NUBUS bus 10. A card according to the present invention includes terminals 51 which are designed to make electrical connections with the respective cooperating terminals in the slot, to thereby permit components on the card to receive all of the signals of the NUBUS bus 10.

A card in one of the slots 29, 30, 31, 32, 33 or 34 can communicate with the memory 2 via the NUBUS bus interface 9, and the CPU 1 can communicate with any memory on the card via the NUBUS bus interface 9, which is described below. For example, the NUBUS bus interface 9 receives addresses for memory on a card in a slot from the CPU 1 over the bus 25 and provides those addresses onto the NUBUS bus 10 through interconnect bus 11; the interface 9 serves to allocate the synchronize the processor buses 5 (through 25) and 6 between the CPU 1 and any CPU on a card (which may seek to control the NUBUS bus to read from or write to the memory on a card). Similarly, the interface 9 receives addresses for the memory 2 from a CPU on a card ("NUBUS bus device") through NUBUS bus 10 and the interconnect bus 11; following synchronization to the processor buses and determination that the NUBUS bus device (which generated the address) may take control of the processor buses (by placing address signals onto the processor address bus 5 through bus 25), the interface 9 provides the address signals to the bus 25 which is connected to the memory 2. The memory 2 responds with data from the addressed location, which data is placed onto the bus 6 which is coupled to the interface 9 which provides that data to the NUBUS bus device through the NUBUS bus 10.

The computer system shown in FIG. 1 utilizes the NUBUS bus as an expansion bus for a computer system on a main circuit board where the CPU 1 processor buses on the main circuit board may not be NUBUS buses. Thus, the slots coupled to the NUBUS bus 10 provide the capability to expand the system to include, for example, additional memory or an additional processor card. However, it is possible to utilize the invention with a NUBUS bus architecture where there is no CPU on a main circuit board and no memory on that board. Such a system is shown in FIG. 13 and will be described below.

FIG. 13 shows a general example of the invention for a computer system utilizing a NUBUS bus 120 on a main circuit board which includes slots each of which is coupled to the NUBUS bus 120. The main circuit board of such a system, as illustrated in FIG. 13, may include the NUBUS bus 120 and 15 slots designated as slot 130, slot 131, ...through slot 144. Each of the slots is coupled to the NUBUS bus 120 by in interconnect bus; hence, slot 130 is coupled to the NUBUS bus 120 by interconnect bus 150, which interconnect bus normally includes all lines of the NUBUS bus 120 and, in addition, includes four lines which serve as distinct identification line means. These four lines typically carry binary values which together can specify any number from 0 to 15. Each of the slots receives a distinct identification line means which provides a different (distinct) number to each of the slots. That is, a distinct identification line means incorporated as part of the interconnect bus 150 barries a distinct signal equal to 0. Slot 144 (slot $E) has a distinct identification line means incorporated as part of the interconnect bus 164 which provides a value (a distinct signal) equal to $E. It is noted that there is no 16th slot because the NUBUS bus standard uses the upper most 256-megabytes (shown as region 40 in FIG.

2) for the small slot spaces (16-megabytes each) allocated to slots 0 through 15. This is seen more clearly in FIG. 2 which illustrates the physical address memory space of a system such as that shown in FIG. 13. Each of the slots $0 through $E have a "super space" of 256-megabytes. Thus, for example slot 0 has a super space of 256-megabytes which was reserved for it from memory location $0000 0000 to $0FFF FFFF. This space is shown generally by number 41 on FIG. 2. This sytem shown in FIG. 13 and 2 includes a slot $0 with memory space reserved for that slot; however, because many microprocessors favor memory in region 41 (the slot $0 super space), for the sake of convenience a typical application of the general invention (e.g. FIG. 13) may not include a slot $0 and no reservation of memory space 41 will be made for any particular slot. Thus, any cards in the remaining slots (i.e. slots $1 to $E) may use the memory in region 41. Of course, any number of slots less than 15 may be implemented according to the invention. As required by the NUBUS bus standards, each of the slots $0 through $E have reserved for them 16-megabytes of space located in the 256-megabyte region labelled generally 40; this region spans from location $F000 0000 to location $FFFF FFFF. Identification signals, such as the four distinct identification lines are used to allocate the "small spaces" in the region 40 to each of the cards. Each of the small spaces in the region 40 is also referred to in the NUBUS bus standards as the "slot space". Addresses of the form $FSiXX XXXX reference address space which belongs to the slot space of the card in slot Si. See pages 30–31 of the IEEE 1196 Specification, Draft 2.0 which is submitted with this disclosure.

FIG. 2 illustrates the general physical address memory space of the system illustrated generally in FIG. 13. The main circuit board which includes NUBUS bus 120 does not include a CPU or memory. The system clocks 170 on the main circuit board provide the NUBUS bus clock signals and are coupled to the NUBUS bus 120 via lines 175 as shown in FIG. 13. Not shown, but understood is the power supply circuitry for the NUBUS bus signals. It is also understood that the main circuit board of the system shown in FIG. 13 should include other NUBUS bus services which are not placed on the cards, such as the NUBUS timeout circuitry.

The computer system shown in FIG. 13 would typically include two printed circuit board cards one of which would be inserted into one slot and the other card (a second card) being inserted into another slot. For purposes of illustration, assume the first card is plugged into the slot $0 (i.e. slot 130) and the second card is plugged into slot $1 (i.e. slot 131). The cards are generally illustrated in FIGS. 4 and 14. They include a printed circuit board card 50 or 50*a* and terminals 51, which terminals are coupled to various components and signal lines on the card 50 or the card 50*a*. The terminals 51 are on a portion of a printed circuit board which protrudes into a receptacle in the slots which contain cooperating terminals for making electrical connections with respective terminals on the card. The physical standards of the interconnections are specified by the NUBUS bus standard. The cooperating terminals in the slots are coupled to the various line and components on the main circuit board; for example, many of the cooperating terminals in the slots are electrically coupled to the NUBUS bus signal lines. These cooperating terminals permit the components on the card to receive the various signals present on the NUBUS bus 120 and to permit one card in one slot to communicate with another card in another slot through NUBUS bus 120 via the interconnect buses, such as interconnect 150 and 151.

In the present example involving FIG. 13, the first card 50 (assumed to be in slot $0) includes a CPU, such as CPU 61 shown in FIG. 4, and a memory, such as RAM 62 and ROM 62 which are coupled together through a card bus 65 disposed on the first card 50. The CPU 61 and the memory 62 are coupled to the system bus, which is the NUBUS bus 120, through the terminals 51 on the card 50. The second card 50*a* (see FIG. 14) in slot $1 includes a memory 62 shown in FIG. 14, such as a random access memory, but does not include a CPU. Such a card is referred to as a slave card and cannot take mastership of th bus 120. The second card typically includes a card bus 65 which includes most (if not all) of the same signals found on NUBUS bus 120. Certain of the address (and data) lines of the NUBUS bus 120 (which are referred to as AD (31...0) in the IEEE 1196 specification, Draft 2.0 since the addresses and data are multiplexed over the same lines) are applied to the decoder means 60. The bus 66 shown in FIG. 4 usually carries the complete NUBUS bus address and data signals and control signals and power signals. In this disclosure, the 32 address lines of NUBUS bus (which also serve as the 32 data line on NUBUS bus) are referred to as A31 through A0 even through they are the NUBUS signals AD (31...0). Essentially, the decoder means 60 of card 50*a* permits the meory 62 on the second card 50*a* to be addressed when the addresses on the NUBUS bus 120 are in the reserved address space of the second card, which in this instance is addresses from location $1000 0000 through location $1FFF FFFF. When the addresses are in that reserved memory space, the decoder means 60 activates the Chip Select (CS) lines (which are coupled to the line 64 from the decoder means 60) of the memory 62 on the card 50*a* thereby indicating to the various RAM or ROM chips on that card that they are being addressed, thereby addressing the memory 62 on the card 50*a* in slot $1. Thus, the memory on the second card 50*a* will receive addresses from the system bus when the decoder means enables, by the Chip Select pins, the memory chips.

Thus, the CPU on the first card 50 in slot $0, which CPU has an address generation means for generating 2^{32} different addresses for addressing memory, provides an address through the terminals of the card in slot $0 onto NUBUS bus 120. Portions of that address appear in the decoder means 60 on the second card 50*a*. If that address is in the range $1000 0000 to $1FFF FFFF the memory on the second card will respond providing data onto NUBUS bus 120 during the appropriate timing cycle.

The decoder means 60 on the second card in slot $1 of FIG. 13 compares the distinct number of slot $1, which number is $1, to the most significant hexadecimal digit of the address appearing on the system bus (NUBUS bus 120) to determine when the distinct number, in hexadecimal, is equal to the most significant hexadecimal digit of the address. When that occurs, the decoder means enables the second memory to be addressed to provide data onto the system bus. Thus, the 256-megabytes "super space" is reversed for the second card in slot $1. As explained below, the decoder means also performs the function of reserving the 16-

megabytes of memory space called for in the specifications of NUBUS bus systems.

It will be appreciated that slot $1 of FIG. 13 is coupled to a distinct identification line means which provides a distinct signal to that slot, which signal identifies a distinct number of that slot. This is true for each of the other slots in FIG. 13 (e.g. slot 144 has a distinct signal of $E which is the distinct number of that slot). Typically, a distinct identification line means comprises four conductors carrying binary values. For slot $1, only one of the four lines will carry the binary value 1 while all others will carry the binary value 0, where the 1 is in the least significant binary digit. Thus, the distinct identification line means will provide the distinct signal 1 to the slot $1 which will identify that slot as having a distinct number $1. It is understood that other ways of identifying a distinct number may be accomplished, such as providing an identification number which through arithmetic conversions produces the distinct number of the slot. Alternatively, one conductor having multilevel logic may be provided as the distinct identification line means.

A preferred embodiment of the invention utilizing six (6) slots will now be described with reference generally to FIGS. 1, 3, 11 and 12. FIG. 11 shows a perspective view of a main circuit board 14 (also referred to as a mother board) which includes a CPU 1, memory 2 which includes read only memory (ROM), I/O circuitry 36, and six slots numbered 29 through 34. The mother board 14 also includes a connector means for providing a connection to a key board as shown in FIG. 11. As with any other personal computer system, the mother board 14 also includes various other circuitry, such as power supplies, latches and buffers, drivers and may include video circuitry, clock circuitry and other components typically associated with personal computer systems as is well known in the prior art. Each of the slots 29, 30, 31, 32, 33, and 34 include cooperating terminals which make electrical connections with terminals 51 on a card which is inserted into the slot. Each of the slots 29-34 receive, according to NUBUS bus standards, substantially all the NUBUS bus signals in NUBUS bus 10 as shown in FIG. 1. The slots receive the NUBUS bus signals through interconnecting buses 19, 20, 21, 22, 23, and 24 as shown in FIG. 1. These connections are common (identical) to each of the slots except for the distinct identification line means which identifies to each of the slots a distinct number that each slot has.

In this particular embodiment, slot 29 is assigned a distinct number $9 by four conductors (lines) carrying binary values as illustrated in the table below. These four conductors are part of the interconnecting bus 19 although they need not be physically present throughtout the entire length of the lines in the NUBUS bus 10 because they can be locally provided in the immediate proximity of slot $9. This is similarly true for slots 30, 31, 32, 33 and 34. The Geographic Address shown in Table 1 is, of course, the distinct number of each of the slots.

TABLE 1

NUBUS bus Slot Numbers For FIG. 1 System

Slot Number in FIG. 1	Geographic Address	GA3	GA2	GA1	GA0	Binary Value
29	$9	GND	open	open	GND	1001
30	$A	GND	open	GND	open	1010
31	$B	GND	open	GND	GND	1011
32	$C	GND	GND	open	open	1100
33	$D	GND	GND	open	GND	1101
34	$E	GND	GND	GND	open	1110

(Binary Values shown after logical inversion by an inverter of the NUBUS bus signals)

Each of the lines in the distinct identification line means for each of the slots is coupled to circuitry which attempts to pull up the lines to the power supply signal +5V. This circuitry will usually invoice a pull up resistor, according to NUBUS bus standards, on each of the distinct identification lines, which resistor will pull up the open signals to substantially +5V and the ground signals will remain substantially at ground. The circuitry shown in FIG. 12, which will be discussed below, assumes that the open signals have already been pulled up (prior to applying them to the decoder means 60) substantially to the power supply voltage level of +5V and that the NUBUS bus signals (including the GA3...GA0 signals and address (A31...A0) signals) have been logically inverted by an inverter. Furthermore, each of the NUBUS bus signals on NUBUS bus 10 must be inverted logically (through an inverter on the cards) before application to the circuitry on the NUBUS bus cards (e.g. card 50 and card 50*a*); similarly, signals from the cards onto NUBUS bus 10 must be inverted logically (through an inverter). Typically, these inverters would be included on the input and output buffers used on the cards. At the interface 9, which interfaces between the NUBUS bus 10 and the motherboard circuitry (i.e. CPU 1, Memory 2, I/O Circuitry 7, the various buses 5, 6, 25, etc.), signals going onto the NUBUS bus 10 are inverted and signals coming from bus NUBUS bus 10 are inverted. Thus, for example, the GA3 NUBUS signal (GND) which is applied to the slots is inverted to logical one ("1") on the card and is then applied to the circuitry in the decoder means 60 shown in FIG. 12. These inversions are well known in the art. Of course, if the CPU 1 and its associated circuitry and buses (e.g. buses 5, 6, 25.) utilize the NUBUS bus system, standards and signals, then no inversion at the interface 9 is necessary.

It can be seen that in this embodiment (shown in FIGS. 1, 11 and 3), slot 30 will have the distinct number $A; slot 31 will have the distinct number $B; slot 32 will have the distinct number $C in the computer system; the distinct number for slot 33 will be $D, and slot 34 will have the distinct number $E. In the IEEE's proposed specification for the NUBUS, referred to as the IEEE 1196 bus specification the distinct identification line means are referred to as the card slot identification and are represented by the symbol "ID (3 . . . 0)" which represent the geographical addresses GA3, GA2, GA1, and GA0. As noted in that specification of the IEEE, at page 6, these four lines are not bussed but are binary encoded at each position to specify the card's position in the computer.

According to the present invention, a computer system as generally shown in FIG. 1 results in a physical address memory "super space" containing 256-

megabytes of reserved memory space. Thus, for example, slot $9 has a reserved super space beginning at location $9000 0000 and ending at location $FFF FFFF. In addition, slot $9 may also have a small space ("slot space") reserved according to the NUBUS bus specification; in accordance with those specifications, slot $9 will have a small space reserved for it beginning at location $F900 0000 to location $F9FF FFFF. As shown in FIG. 3, the 256-megabyte region 42 contains the small spaces for the various slots. There is an unreserved NUBUS memory address space 43 which may be used by additional expansion slots which may be added to a system designed according to the present invention. The lowest 256-megabyte memory space, designated 45, is the local address space for the CPU 1 which is assigned the distinct number $0 as if it were on a card in slot $0. The CPU 1 may be designed to "occupy" additional slots—that is it may be assigned distinct numbers $1, 2 and 3 and therefore have the entire region 44 reserved as in the particular embodiment shown in FIG. 3; in effect, the motherboard becomes a card in 4 slots ($0, 1, 2, and 3). If the designer seeks to isolate super space slot $0 completely for CPU 1's use (i.e. prevent a NUBUS bus access to that super space $0), the NUBUS bus interfere 9 will be designed to prevent such access but permit access to the data in super space $0 by aliases replicated in super space $1 or $2 or $3. Thus, NUBUS bus addresses on NUBUS bus 10 in super space $0 may decoded to the same respective location (i.e. $0XXX XXXX to $1XXX XXXX) in super space $1. In such a situation the NUBUS bus cards (in the actual physical slot $9 through $E) may access the slot $0 super space by addressing super spaces $1, 2 or 3 which can be designed to include aliases of the data stored in super space $0. The address space ($0000 0000 to $1000 0000) is also the local address memory space for cards operating entirely on the card without a NUBUS bus transaction; that is, a card, such as the one shown in FIG. 4 having a CPU may locally address its local RAM on the card in this same address space 45 provided the CPU does not initiate a NUBUS bus transaction. Such an arrangement for purely local transactions on the card is implemented by address decoders on the card as is well known in the art.

This particular embodiment shown generally in FIG. 1 also reserves additional memory space for the I/O circuitry and read only memory (ROM) which is part of the Memory 2 as shown in FIG. 3. In particular, address memory space is reserved from $4000 0000 to location $4FFF FFFF. Moreover, memory address space for I/O operations and circuitry is reserved from location $5000 0000 to location $5FFF FFFF. FIG. 3 shows an embodiment of the present invention where the I/O and ROM memory space is located at $4000 0000 to $5FFF FFFF. Thus, access to ROM OR I/O information can be obtained by the CPU 1 or by a second CPU 61 by addressing those locations from $4000 0000 to $5FFF FFFF. Another embodiment of the present invention is shown in FIG. 10 where the motherboard I/O and ROM memory space with respect to NUBUS bus cards is located at $F000 0000 to $FOFF FFFF. In this embodiment, the memory space of motherboard I/O information and system ROM (on the motherboard) which is accessible by the NUBUS bus cards (in NUBUS bus slots) is limited to 16 MB (megabytes) while CPU 1 may still access region $4000 0000 to $5FFF FFFF; however, many possible systems can be constructed in which this limited space of 16 MB is sufficient for ROM and I/O use. Thus, for NUBUS bus card, it may access the ROM which is part of memory 2 on the motherboard by presenting addresses in the range $F000 0000 to $FOFF FFFF on the NUBUS bus which cause an access to that ROM. This is implemented in well-known fashion by the interface 9 which decodes addresses from NUBUS bus in the $F000 to 0000 to $FOFF FFFF region into the ROM and I/O region of the motherboard ($4000 0000 to $5FFF FFFF). The CPU 1 need not be similarly constrained, and accordingly, it may seek motherboard ROM or I/O memory by addressing the region defined by $4000 0000 to $5FFF FFFF; that is, CPU 1*a* may have additional ROM or I/O memory (as part of memory 2) which is not available to the NUBUS bus cards (which are limited in access to essential system ROM and I/O on the motherboard). This embodiment of the invention, as shown in FIG. 10 is consistent with the NUBUS bus standards which require a configuration ROM be located at the top of the 16 MB small (slot) space; thus, slot $0's ROM space is located at the top of the space $F000 0000 to $FOFF FFFF.

The card according to the present invention will be described with reference generally to FIGS. 4, 12 and 14. FIG. 4 shows a card of the present invention which may be incorporated into the computer system of the present invention by plugging it into one of the slots of the system, such as slot 29. The card includes a printed circuit board 50 on which is disposed conducting means forming various lines such as the card bus 65 and the interconnect buses 67, 68, and 69. Similarly, FIG. 14 shows a card 50*a* of the present invention which is substantially identical to the card shown in FIG. 4 except it does not include a CPU 61 which generally permits the card 50 to act as a master with respect to the NUBUS bus 10 while the card 50*a* shown in FIG. 14 can usually only be a slave and cannot take control of the NUBUS bus 10 and cannot initiate a NUBUS bus transaction. The cards 50 and 50*a* include terminals 51 which make electrical connections with cooperating termimals in the slots to thereby couple the various components on the cards to the various signals appearing on the main circuit board 14. All NUBUS signals (to and from NUBUS bus) are buffered and inverted by the buffers 59 on the cards. Thus, for example interconnect bus 63 connects the address lines A31 through A24 of the NUBUS bus 10 to the decoder means 60. The bus 63 also includes power and the distinct identification line means, which in this embodiment has four signal lines GA3, GA2, GA1, and GA0, that are coupled to terminals 52, 53, 54, and 55 respectively. That is, the signal GA3 is applied to terminal 52 through a cooperating terminal located in the slot which receives the card 50. Similarly, the signal GA2 is applied to terminal 53; signal GA1 is applied to terminal 54; and GA0 is applied to terminal 55. These terminals 52, 53, 54, and 55 are coupled to conductor means which present these four signals (as inverted) to the decoder means 60 at the input 82 of the decoder means 60, as shown in FIG. 12.

The signals present in the slots of this particular embodiment are presented below in Table 2 and are NUBUS bus signals. Of course, NUBUS bus 10 includes a 32-bit address bus which, during a first read cycle presents the address of the memory location sought to be accesses and during a second cycle acts as a data bus and receives data stored in that memory location. During a writing to memory, NUBUS bus 10 carries, on its 32-bit address bus during a first cycle, the address of the location to be written to and during a second cycle NUBUS bus 10 provides the data to be written into the location addressed in the first cycle. The NUBUS bus 10 is substantially an IEEE 1196 bus. The cards generally accept and use most of these signals although their use will depend on the particular needs of the card and the designer's goals.

TABLE 2

	NUBUS bus Slot Signals Description
Signal	Description
+5 V	Power to slot. 5 Volts.
+12 V	Power to slot. 12 Volts.
−12 V	Power to slot. −12 Volts.
−5.2 V	Unused in this embodiment. All −5.2 V signals are connected together on the slots.
GND	Power return for +5 V, +12 V, and −12 V.
RESET	Open collector signal. Asserted at power up, by the CPU 1, or by a push button reset switch which may be included. Pulled up to +5 V by a 1K ohm resistor. Slot card should use this signal to reset circuitry on card.
SPV	Slot Parity Valid. If a card is providing parity on /SP this signal is asserted. The slash ("/") indicates the signal is active low-that is, it activates its target when it goes low.
SP	Slot Parity. Odd parity of /AD0–/AD31 if /SPV asserted.
TM0–TM1	Transaction modifiers. Used during START cycle to indicate the size of the transaction. Used during ACK cycle to indicate completion status.
A0–A31	NUBUS bus Address/Data bits 0 through 31. Used during START cycle to indicate address. Used during ACK cycle to indicate data. NUBUS bus specifications refer to these sugnals as AD0–AD31 or AD (31 . . . 0) because the same 32 lines carry address during a first cycle and then carry data during a second cycle.
PFW	Power Fail Warning. An open collector signal pulled up by a 220 w resistor to +5 V. When the signal is pulled up the power supply is activated. When this signal is pulled low the power supply is disabled. The power supply itself will pull this signal low as a power fail warning 2 ms before the AC power is lost. This is an option under IEEE 1196 standards.
ARB0–ARB3	Arbitration bits 0 through 3. Open collector signals which are terminated in the slots in accordance with IEEE 1196 specifications (see, e.g., Table 6 of the specifications). Used to arbitrate bus mastership between the slots according to NUBUS bus Specifications.
GA0–GA3	Geographical Address bits 0 through 3. Hard coded binary address of slot. Pins tied to GND or open (or +5 V instead of open).
START	Asserted to indicate the presentation of an address on A0–A31. Also used to start arbitration for the bus mastership.
ACK	Acknowledge. Used to indicate acknowledgement of START cycle.
RQST	Request. Asserted to request bus mastership.
NMRQ	Non-master request. An open collector signal which are terminated in the slots in accordance with IEEE 1196 specifications (see, e.g., Table 6 of the specifications). Used by card to signal a interrupt to interrupt receiver.
CLK	NUBUS bus Clock. Asymmetrical 10 MHz clock which sychronizes transactions on NUBUS bus.

The construction and use of the decoder means 60 is known by those with ordinary skill in the art. It essentially involves the use of a compartor means with an enabling means where the comparator compares the NUBUS bus address to the signal appearing on the distinct line identification means and determines when the address is within the reserved memory space for the memory 62 of the card. However, the use of the decoder means in this context to reserve 256-megabytes of memory space is novel and accordingly, a description of a simple decoder means including a comparator means and an enabling means will be described. It is within the ordinary skill of the art to develop other decoder means which perform the functions of the present invention.

In a typical transaction between the card 50*a* and the CPU 1, the memory 62 is selectively coupled to the CPU 1 through NUBUS bus 10 and its associated interface 9, described below, to receive addresses and to provide data (or receive data when written to) over NUBUS bus 10. The CPU 1 includes an address generation means for generating 2^{32} different addresses from location $0000 0000 to location $FFFF FFFF. Addresses from the CPU 1, which are 32-bits wide, exit the CPU 1 through the processor bus 5. The 32-bit address then enters the interconnect bus 25 and appears at the interface 9 which determines that the address is within the NUBUS bus address space, which begins at $6000 00000. Below that address, memory 2 and I/O circuitry 7 will be addressed by the CPU 1. At and above that address, memory in the slot's super spaces or small spaces will be addressed. Interface 9 determines that a NUBUS bus address is being selected and permits, after synchronizing the address signals of the CPU 1 to the NUBUS bus and determining ownersip of the NUBUS bus 10 in favor of the CPU 1, the address to appear on NUBUS bus 10 through the interconnect bus 11. For purposes of illustration, we shall assume that a card 50*a*, shown in FIG. 14 is in slot $9 which has a distinct number in the system of $9. The decoder means 60 receives the address signals through NUBUS bus 10 and determine whether the addresses are for that card's memory space.

The decoder means 60 includes a comparator means 70 which compares the most significant hexadecimal digit of the address (for reading or writing) to the distinct number, in hexadecimal, of the slot into which the card having the decoder means 60 is plugged. The decoder means also includes a control and clock signal means 71 which includes NUBUS bus clock and START and ACK signals. The decoder means may also further include a driver, a well-known component in the prior art and hence not shown, which provides enough current to drive the output from the decoder means 60 to sufficient levels to affect the target of those outputs, which is the Chip Select (CS) lines and pins of the memory 62. The comparator means 73, which is also part of the decoder means 60, compares the address to determine whether the slot's small space is being addressed. When one of comparator means (either 70 or 73) determines that the address appearing on NUBUS bus 10 is within the super space or small space of the card, that particular comparator means along with the control means 71 activates the Chip Select (CS) lines connected to the memory 62. The Chip Select (sometimes referred to as the Chip Enable Signal) line is used, as is well known, to indicate to memory, such as memory 62, that it is being addressed (either for reading or writing). The Chip Select lines are coupled to line 64 as shown in FIGS. 4 and 14.

The comparator means 70 of the decoder means 60 includes four exclusibve OR gates ("XOR"), such as the exclusive OR gate 76 which compares the GA3 signal (appearing at input 92) to the most significant binary bit of the 32-bit address line, A31, which is input at input 91 of the exclusive OR gate 76. It is understood, as noted before, that the NUBUS bus signals in the decoder means **60** are inverted (on the card in buffers **59**); thus, GA3 . . . GA**0**, the address signals A**31** . . . A**24** and START, ACK and CLK as used in the decoder means **60** are inverted. For example, the START signal shown in FIG. **12** is the inverted NUBUS bus START signal. If the most significant binary bit of the address is equal to the signal GA3 then a logical **0** will appear at the output of the exclusive OR gate **76**, which output is passed via line **93** to a four input OR gate **77**. The address signals A**31** through A**28** and certain signals, such as power and ground, are applied to the comparator means **70** at the input **83**. These signals are then provided to the various exclusive OR gates of the comparator means **70** as shown in FIG. **12**. The output from each of the exclusive OR gates in comparator means **70** will be logical 0 only if the two inputs to a particular XOR gate are identical. Thus, each exclusive OR gate does a bit for bit comparison between one of the bit carrying lines which acts as a part of the distinct identification line means and one of the four most significant address lines. It can be seen that when a distinct number, in hexadecimal, is equal to the most significant hexadecimal digit of the address, each of the exclusive OR gates will produce a logical 0 at its output causing the output of the OR gate **77** to also be logical 0 causing node **70***a* to be logical 0. Node **70***a* is coupled to the output of OR gate **77** and is also coupled to one of the inputs to NAND gate **90** which is part of the control means **71**. The output from the comparator means **73** is coupled to node **73***a* in the control means **71** and is also coupled to the other input of NAND gate **90**. When an address is in the card's slot space, the output of the comparator means **73** will be logical 0 and node **78** (the output of NAND gate **90**) will be logical 1. When an address is in the super space of the slot, the output of comparator means **70** will be logical 0 and node **78** (the output of NAND gate **90**) will be logical 1. When the address is not in the slot's small space and not in the card's super small, node **78** will be logical 0 (since node **70***a* and node **73***a* will each be logical 1). When the address is valid (during a START), the signal at the output of AND gate **87** will be logical 1 and will be clocked (at the next NUBUS bus clock pulse) to the output Q of the flip-flop **80** so that a logical 1 appears at node **79**. Thus, when an address is valid and is in the card's reserved space (small or super), nodes **78** and **79** will be logical 1 causing line **64** to be logical 0, thereby activating the memory **62** for addressing. At the end of the time when the address is valid, the output of AND gate **87** will be logical 0 and will be clocked to node **79** (through the JK flip-flop **80**) and the memory **62** will be deactivated. When an address is valid, START (as shown in FIG. **12**) is logical one and ACK is logical 0 (see insert to FIG. **12** showing a timing diagram of the signals START, ACK and CLK which are inputted to the means **71**). The ACK signal is inverted at the input to AND gate **87**. Thus, when an address is valid, the output of AND gate **87** is logical 1; when an address is invalid, START is logical 0 causing the output of AND gate **87** to be logical 0, which value is clocked to the output Q of flip-flop **80** at the next NUBUS bus clock pulse as shown in FIG. **12**. A logical 0 at output Q will deactivate the CS lines of memory **62**. The flip-flop **80** is a clocked JK flip-flop with the K input tied to the J ("D") input through an inverter; such a flip-flop is sometimes referred to as a D-type flip-flop where K is the complement of J. An end of Cycle signal may optionally be applied to the Reset input of the flip-flop **80**.

The signal is obtained from the control circuitry on the card (e.g. CPU **61**) and it indicates the end of a transaction. The End of Cycle signal is active low and therefore it is inverted at the input to Reset.

The particular output on line **64** from the control means **71** will depend on whether memory **62** specifies (according to the manufacturer) that CS is active low (i.e. at a low voltage like ground) or high (+5 volts). In this example, the memory **62** is assumed to have CS active low ("/CS") and therefore the memory **62** is selected for addressing when the output of means **71** is logical 0. Thus, the activation of line **64** occurs when the output of NAND gate **72** is logical 0 (low), causing CS to be pulled to substantially ground and thereby indicating to the memory chips (memory **62**) that they are being addressed.

If there is no match between the distinct number and the most significant hexadecimal digit of the address, at least one logical 1 will appear on one of the four outputs of the exclusive OR gates in the comparator means **70** which causes a 1 to appear at the output of the OR gate **77**, which logical value 1 appears at node **70***a*. This means the address is not in the card's super space. In this case, the memory **62** can only be addressed from NUBUS bus **10** only if the address is in region **42** (small spaces).

The decoder means **60** also includes a comparator means **73** which is responsible for reserving for the particular card a "slot space" which is in the upper 1/16th physical address space of the system (i.e. region **42** shown in FIG. **3**). More specifically, a comparator means **73** allocates 16-megabytes of memory for the card is plugged. The comparator means **73** includes a NAND gate **85** which determines when addresses presented to the card are in the region **42**. The exclusive OR ("XOR") gates of the comparator means **73**, such as exclusive OR gate **88**, and the OR gate **89** compare the second most significant hexadecimal digit to the distinct number of the slot into which the card is plugged to determine when the distinct number is equal to the second most significant hexadecimal digit of the address appearing on the 32-bit address bus of NUBUS **10**. When this equality condition occurs each of the XOR gates of means **73**, such as gate **88**, will produce a logical 0 at its output causing the output of the OR gate **89** to be logical 0. The output of OR gate **89** is one of the inputs to OR gate **75**. The four most significant binary bits of the address (A**31** . . . A**28**) are applied to the inputs of NAND gate **85**; the output of this gate is logical 0 only when the address is in the small space region **42**. The output of NAND gate **85** is one of the inputs of OR gate **75**. The inputs to OR gate **75** are both logical 0 only when the address is in the card's small space in region **42**. Thus, the output of OR gate **75** is only logical 0 when the address is in the card's small space. The address lines (A**27**, A**26**, A**25**, and A**24**) constitute the second most significant hexadecimal digit of the address appearing on the 32-bit address bus NUBUS bus **10**.

It can be seen that when a card, such as card **50***a*, is plugged into a slot having a distinct number $X, a decoder means **60** will cause that card to have memory space reserved for it from locations $X**000 0000** to $XFFF FFFF and additional memory space from $FX**00 0000** to location $FXFF FFFF.

Transactions between the CPU **1** and NUBUS bus **10** typically require certain actions of the interface **9** which is referred to as the NUBUS bus interface **9**. The exact implementation of the interface will depend on the microprocessor selected for CPU 1 and on its associated buses. In its simplest form, the interface could be another decoder means, having six decoders, each such as decoder means 60; that decoder means receives six different distinct signals having the distinct numbers $0, $1, $2, $3, $4 and $5, each of those signals for one of the six decoders; this arrangement would produce the resulting division of physical address memory space as shown in FIG. 3 for the computer system shown in FIG. 1. The interface 9 would also be required to synchronize any differences in timing between the CPU 1 and the NUBUS bus Clocks and would determine ownership of the buses being requested (whether the NUBUS bus 10 or the processor buses 5, 25 and 6) by the master device, so that only 1 address appears on all buses 10, 5 and 25 at one time. Thus, there would be several decoder means as shown iin FIG. 12 each of which receives a different distinct signal. The output of these decoder means would be coupled to the CS pins of memory 2. At the same time, the CPU 1 could access the slots attached to NUBUS bus 10 by merely placing signals on the address bus 5 which is coupled to the interface 9 which permits the address signal from the CPU 1 to appear on NUBUS bus 10. Similarly, the CPU 1 could provide data to NUBUS bus slots by placing the data on the data bus 6 which causes the data signals to appear at the NUBUS bus interface 9 via the interconnect bus 12 and those data signals would then be conveyed to NUBUS bus 10 and then received by the appropriate slot depending on the immediately preceding address signal which appeared on NUBUS bus 10. In effect, the CPU 1 and its associated circuitry including the memory 2 would appear to NUBUS bus 10 as if it was on a card in slot 0 or slots 1, 2, and 3. In the following discussion of a NUBUS bus interface, the term processsor bus is generally used to refer to the data bus 6 which is coupled to the CPU 1 and to the memory 2 and to the address buses 5 and 25 as shown in FIG. 1.

The NUBUS bus interface 9, as shown in FIG. 5, includes three state machines and the NUBUS bus clocks which interface between the six slots (29, 30, 31, 32, 33, and 34) and the NUBUS bus 10 and CPU 1 and memory 2 and their associated circuitry on the mother board 14. In general, the interface 9 must determine ownership of the requested bus(es) between masters, such as CPU 1 and CPU on a card (e.g. CPU 61), to prevent 2 different addresses from 2 different masters from appearing on a bus, such as bus 5 or NUBUS bus 10, simultaneously; that is, the interface 9 must determine bus ownership, via arbitration between possible masters requesting the same bus, to prevent address collisions on a bus. Similarly, during data cycles the interface 9 must determine bus ownership, via arbitration between possible masters requesting the same bus, to prevent data collisions on a bus (such as bus 6 or NUBUS bus 10). Moreover, the interface 9 must synchronize the signals of the requesting master to the timing of the requested bus which will be driven (for addresses or writing data) or listened to (for reading data) by the master. The interface may be implemented by well-known techniques in a programmable logic array.

The signals present on NUBUS bus are described in the 1196 specification of the IEEE and in the Texas Instruments' publications referred to above. Generally, the NUBUS bus standards specify logical, physical and electrical standards for the four types of signals present in the NUBUS bus 10. These signals include utility signals such as the clock and the distinct identification line means; the address/data signals along with various control signals; the arbitration signals; and the power signals. It can be seen that certain of these NUBUS bus signals appear on the left side of the NUBUS bus interface 9 shown in FIG. 5. Signals provided by the CPU 1 or the memory 2 flow through the interface or permit the interface to allow the CPU 1 to communicate with NUBUS bus 10 and vice versa. The following table describes the signals used in the NUBUS bus state machine involved in the NUBUS bus interface 9. The particular implementation of the interface 9 will depend on the particular CPU 1 selected for use on the mother board on the designer's goals.

TABLE 3

Signals used in NUBUS bus State Machines In NUBUS bus Interface 9

Signal	Description
RQST	A NUBUS bus signal; active low; indicates a request for bus mastership.
NUBUS	Decoded address from processor CPU 1 indicating an address reference to NUBUS bus; active low. The address from CPU 1 is decoded in a decoder means, which can be readily constructed by one of ordinary skill in the art, and which determines when the address on bus 25 in the NUBUS bus address range of $6000 0000 to $FFFF FFFF.
START	NUBUS bus signal; active low; indicates an address is present on NUBUS bus.
ARB0–ARB3	NUBUS bus signals; active low; arbitration address of bus masters competing for NUBUS bus mastership.
ACK	NUBUS bus "acknowledge" signal; active low; slave NUBUS bus device is acknowledging START transaction.
RMC	Processor CPU 1 signal indicating a read/modify/ write is occurring on the processor CPU 1 bus 6 and 25.
AS	Processor CPU 1 address strobe indicating the address lines from the CPU 1 are valid and a cycle is requested. Active low ("/AS").
/BUSLOCK	The processor buses 6, 5 and 25 can not be interrupted by NUBUS bus transactions into memory 2.
DSACKx	The Data Strobe Acknowledge from the memory 2.
BG	Processor CPU 1 bus grant indicating the processor buses 5, 6 and 25 have been granted to the NUBUS bus to communicate with the memory 2 using the NUBUS bus to Memory 2 state machine 104.
C16M	The processor CPU 1 clock which is used to qualify signals from the processor CPU 1 as valid.
R/W	Read/Write signal which is used to indicate when a read or a write is occurring.
/BR	A bus request from NUBUS bus requesting mastership of the processor buses, principally bus 6 (via bus 12) and buses 5 and 25.
/BGACK	NUBUS bus signal from NUBUS bus state machine 104 acknowledging granting of the processor buses by the processor. Typically, NUBUS bus requests control of the processor buses by issuing a /BR signal; request for the processor buses is granted by the signal /BG which is received by the NUBUS bus to memory 2 state machine 104 which acknowledges receiving the granting of the processor buses for mastership.
/BERR	Bus error signal from NUBUS bus indicating there is an error in the system. This signal is usually issued by the NUBUS bus timeout state machine 105 which watches for transactions which exceed approximately 25 microseconds; any such transaction is assumed by the bus timeout state machine to be in error resulting in the signal /BERR to be sent to the processor.
/DS	Datastrobe: A NUBUS bus signal indicating the data lines from the NUBUS bus are valid and a

TABLE 3-continued

Signals used in NUBUS bus State Machines In NUBUS bus Interface 9

Signal	Description
	cycle is requested.

The processor CPU 1 typically accesses and requests the NUBUS bus 10 whenever the processor CPU 1 generates a physical address from $6000 0000 to $FFFF FFFF. The CPU 1 to NUBUS bus state machine 103 determines there is such a request when decoders on the mother board coupled to bus 25 indicate an address on bus 25 has a most significant hexadecimal digit between $6 and $F, including $6 and $F. Under these circumstances, the output of those decoders causes the assertion of the /NUBUS signal. The state machine 103 the synchronizes the request for NUBUS bus control with the NUBUS bus clock and presents the same address over the bus 10 after determining the CPU 1 may take ownership of NUBUS bus 10 to drive the address signals onto the NUBUS bus 10. If a card on NUBUS bus responds, the data is transferred. If no card responds, a NUBUS bus timeout occurs and a bus error (/BERR) is sent to the processor, which usually causes execution of an error handling routine. The NUBUS bus timeout state machine 105 monitors the time between START signals on NUBUS bus and acknowledge (ACK) signals on NUBUS bus. When the time between those signals exceeds 255 NUBUS bus Clocks, according to the NUBUS bus standards, the NUBUS bus timeout state machine generates the bus error as indicated above. FIG. 8 illustrates the signals involved in the processor CPU 1 to NUBUS bus transaction through the NUBUS bus interface 9 and more specifically through the processor to NUBUS bus state machine 103. The signals on the right side of the block 103 shown in FIG. 8 which are directed to the CPU 1 side of machine 103 are NUBUS bus signals. The right side of machine 103 is the NUBUS bus side of the system and includes the 6 slots. On the left side of the interface 9 is the CPU 1 and the memory 2 portion of the system. This is also true for FIG. 9. Signals entering (i.e. the arrow is directed towards the machine 103) the machine 103 from the NUBUS bus side are generally NUBUS bus signals and signals exiting the machine 103 on the NUBUS bus side are generated by the CPU 1 or the result of the interaction CPU 1 and the machine 103. Similarly, signals on the CPU 1 side of the machine 103 which enter the machine 103 are signals generally from the CPU 1 or memory 2 or circuitry associated with that portion of the system. The signals on the CPU 1 side of machines 103 and 104 are carried by the bus 12 of FIG. 1 and the signals on the NUBUS bus side of machines 103 and 104 are carried by bus 11.

The normal CPU 1 to NUBUS bus transaction starts with the state machine 103 waiting for the signal NUBUS bus to be asserted (which is synchronized to the 10-MHz NUBUS bus clock). When this signal is asserted, and no other bus masters are asserting RQST on NUBUS bus 10, state B is entered into form state A, the prior waiting state. State B has asserted the RQST signal of NUBUS bus and establishes a request by CPU 1 for the NUBUS bus 10 among other bus masters which are asserting RQST at the same time. For purposes of arbitration under the NUBUS bus standards, the CPU 1 is assigned to slot $0.

State B is followed by state C during which the arbitration and acknowledge (ACK) signals are sampled to check if any other NUBUS bus transaction is in progress or if some other NUBUS bus master has won NUBUS bus 10. If a transaction is in progress and no other bus master won mastership, state C is retained. If any other bus master requested the bus during state B, state D is entered into. [Note: Since the processor CPU 1 accesses the bus from slot $0, it always loses to the other slots since the arbitration is based on the distinct number under the NUBUS bus standard]. If no other mater has won the bus and no other transaction is occuring, state E is entered into.

State E asserts the START signal of the NUBUS bus 10 and drives the address from CPU 1 onto the NUBUS bus 10. It is understood that latches and buffers are used to temporarily store addresses and data in these state machines 103 and 104 and generally in the system. State F follows State E and waits for the acknowledge signal (ACK) from the card which was addressed. When the acknowledge signal is asserted on NUBUS bus 10, and no other masters are requesting the bus 10, a State G is entered in which the DSACKx signals to the processor CPU 1 are generated to finish the process cycle. If no other master is asserting RQST during State G, State H is entered into which is a State in which the NUBUS bus 10 is "parked" which is to say that a second NUBUS bus transaction from the processor CPU 1 will be able to go directly to state E to start the NUBUS bus access instead of state A. If RQST is asserted during States F, G, or H, the NUBUS bus 10 must be rearbitrated to determine the current bus master and State A becomes the waiting State rather than State H. These sequences of states may be executed by well known state machine techniques. The following table summarizes the states and signals involved in the processor CPU 1 to NUBUS bus interface which is executed by the CPU 1 to NUBUS bus state machine 103.

TABLE 4

Processor CPU 1 to NUBUS bus States

States	Signals Asserted	Description
A		Idle state. Waiting for the processor CPU 1 to generate NUBUS bus address access (addressing a memory location from $6000 0000 to $FFFF FFFF) and for RQST (from cards) to be deasserted by cards in the NUBUS bus slots.
B	RQST	Request NUBUS bus. The processor CPU 1 is requesting NUBUS bus and no other RQST asserted.
C	RQST	Test for arbitration win. The arbitration lines should all be deasserted since processor CPU 1 is arbitration number zero. If last cycle is waiting for ACK, stay put. If an arbitration line is asserted, try again after next START transaction.
D	RQST	Wait for next round of arbitration. START indicates next round of arbitration is available.
E	START, A0–A31 (NUBUS bus)	Start transaction. Assert processor CPU 1 address on 32-bit address line of NUBUS bus 10.

TABLE 4-continued

Processor CPU 1 to NUBUS bus States		
States	Signals Asserted	Description
F	A0-A31	Wait for ACK. Wait for acknowledge from slave device. CPU 1 Asserts A0-A31 (NUBUS bus) if CPU 1 is writing to NUBUS bus device (e.g. a card). Note whether RQST is asserted to determine if bus will remain "parked". If RQST is asserted, the state machine will recycle to state A after state G.
G	DSACK0, DASCK1	Assert DSACKx. NUBUS bus slave completed transaction, and processor CPU 1 cycle. NUBUS bus remains "parked".
H		Wait for next processor CPU 1 to NUBUS bus transaction. NUBUS bus remains "parked" to allow quick start to next cycle.

The state machine shown in FIG. 8 receives the address signals of the CPU 1 (A0-A31) from the CPU 1 on the bus 25. The signals appearing on the right side of the state machine 103 are NUBUS bus signals. Certain signals on the left side of state machine 103 are also NUBUS bus signals such as the clock signals /CN10M and C20M, as well as /NUBUS bus although the latter is caused by CPU 1 by generating a NUBUS bus address.

The NUBUS bus to CPU 1 buses state machine 104, as shown in FIG. 9, is for access for the memory 2 (which may include RAM, ROM and I/O) from NUBUS bus. In one embodiment, if an address from $0000 0000 to $5FFF FFFF is presented on the NUBUS bus, then the NUBUS bus to processor buses state machine 104 requests the processor buses from the CPU 1 and performs an access to the address. An alternative embodiment (FIG. 10) will also be described in which accesses to RAM of memory 2 occur by addressing $0000 0000 to $3FFF FFFF and accesses to ROM or I/O of the motherboard occur by addressing $F000 0000 to $F0FF FFFF. Normally, after the data is sent to or from the NUBUS bus master (i.e. the card in the NUBUS bus slot), control of the processor buses 5 and 6 is returned to the processor CPU 1.

The following Table describes the states and signals involved in the NUBUS bus to CPU 1 buses transaction.

TABLE 5

State	Signals Asserted	Description
A1		Idle state. Waiting for address on NUBUS bus 10 to processor buses locations (e.g. $0000 0000 to $3FFF FFFF and $F000 0000 to $F0FF FFFF). If the processor buses are not locked (e.g. by locking the processor buses through assertion of Buslock signal of CPU 1) and the CPU 1 is not doing a NUBUS bus access, the processor buses will be requested. If Buslock is asserted, then NUBUS bus access to Memory 2 is delayed until Buslock is reasserted and the state remains at A1.
B1	BR	Bus Request asserted. Request by NUBUS bus of processor buses for NUBUS bus to Memory 2 transaction. Wait for CPU 1 to assert Bus Grant and deassert address strobe.
C1	BGACK, A0-A31 (on bus 25) D0-D31 (on bus 6) R/W	Assert mastership of processor buses and set up addresses and/or data.
D1	AS; DS; A0-A31 (on bus 25)	Address strobe asserted. Data strobe asserted.
E1	D0-D31 DSACK	Wait for valid data from Memory 2 (or write to Memory 2 during time when data is valid). Wait for Data Strobe Acknowledge (DSACK) from Memory 2 to indicate end of cycle.
F1	ACK (NUBUS bus)	NUBUS bus to processor buses transaction complete. Wait to determine if next cycle will continue with NUBUS bus controlling the processor buses. NUBUS bus can lock onto the processor buses by asserting a Lock Attention signal which causes CPU 1 to relinquish control of the processor buses for several transactions without CPU 1 contention until Null Attention signal is asserted; assertion of Lock Attention causes looping of the states B1 to F1.

The NUBUS bus to CPU 1 buses transaction begins with state A1 shown in Table 5 above, where the state machine 104 is idling by waiting for an address on NUBUS bus 10 in the Memory 2 memory space (e.g. $0000 0000 to $5FFF FFFF; or, in the alternative embodiment of FIG. 10, $0000 0000 to $3FFF FFFF and $F000 0000 to $F0FF FFFF). NUBUS bus accesses to the processor buses can be prevented by asserting the Buslock signal which causes all NUBUS bus transactions to this address space to be acknowledged with a "try again later" response. If the address is within the Memory 2 space and Buslock is not asserted, then state B1 is entered.

At state B1, the CPU 1 releases the processor buses by issuing a BusGrant which responds to a Bus Request; the Bus Grant is acknowledged by the NUBUS bus device by a BusGrant Acknowledgement in the next state, C1. The addresses are given onto the processor address buses and the data is transferred in states D1 and E1. The transaction is completed in F1 when the NUBUS bus ACK signal is asserted on NUBUS bus 10.

In the alternative embodiment of FIG. 10, the NUBUS bus devices access the RAM of memory 2 by presenting addresses in the range $0000 0000 to $3FFF FFFF. NUBUS bus devices, in this embodiment, access a portion of the motherboard's ROM memory space and a portion of the motherboard's I/O memory space (which is usually physical RAM set aside for I/O use) indirectly by presenting addresses on NUBUS bus 10 in the range of $F000 0000 to $F0FF FFFF (slot space $0). In this embodiment, adresses on NUBUS bus 10 in the range $4000 0000 to $5FFF FFFF do not access ROM or I/O, but addresses on the CPU 1 buses (e.g. bus 5) in that range do access the complete motherboard ROM and I/O memory space. In keeping with NUBUS bus standards, the portion of ROM of the motherboard (which is assigned to at least slot $0) which is accessible to NUBUS bus is placed at the top of slot space $0. The particular allocation of the memory in slot space $0 between motherboard ROM and motherboard I/O depends on the designers needs. In one preferred embodiment, the slot space $0 is divided in half such that an address to $F080 0000 to $F0FF FFFF on NUBUS bus 10 produces an access to an 8 megabyte region of the ROM of the motherboard (i.e. ROM of the memory 2), and an address to $F000 0000 to $F07F FFFF on NUBUS bus 10 produces an access to an 8 MB (megabyte) region of the I/O memory space. The particular 8 MB portions of ROM and I/O memory space will depend on what regions of memory NUBUS bus devices will need or want to use. Often, the entire system (motherboard) ROM and motherboard I/O Wiill fit into the 16 MB region of slot space $0. Well known decoders may be used to cause the decoding from the NUBUS bus address in slot space $0 to the appropriate ROM and I/O location.

What is claimed:

1. A computer system comprising a main circuit board including a central processing unit and slots each with means for receiving a printed circuit board card, memory coupled to said central processing unit (CPU) to receive addresses of memory locations from said CPU and to provide data to said CPU, said memory being disposed on at least one of said main circuit board and said card, said main circuit board including input-output circuitry coupled to said memory to provide data to said memory and coupled to said CPU to receive control signals from said CPU, said main circuit board having less than 16 slots, said main circuit board including a 32 bit address bus being coupled to said CPU and to said memory to address said memory, said CPU including an address generation means for generating 2^{32} different addresses ranging from location $0000 0000 to location $FFFF FFFF, said location being in hexadecimal notation, each of said slots having a distinct number in said system and being coupled to said bus for addressing said memory, each of said slots being coupled to distinct identification line means on said main circuit board, each of said distinct identification line means providing a distinct, unchanging signal to the slot to which said distinct identification line means is coupled, said distinct signal for a particular slot identifying the distinct number of said particular slot, where said distinct number of a particular slot is (ID), said distinct number resrving 256 megabytes of memory space for each of said slots such that said 256 megabyte memory space beings at location $(ID)000 0000 and ends at location $(ID)FFF FFFF, whereby any card in slot X will have memory space reserved beginning at location $X000 0000 and ending at location $XFFF FFFF, said locations being in hexadecimal notation.

2. A personal computer system comprising a main circuit board including a central processing unit (CPU) and slots each with means for receiving a printed circuit board card, memory coupled to said CPU to receive addresses of memory locations from said CPU and to provide data to said CPU, said ,memory being disposed on at least one of said main circuit board and said card, said main circuit board including input/output circuitry coupled to said memory to provide data to said memory and coupled to said CPU to receive control signals from said CPU, said main circuit board having less than 16 slots, said main circuit board including a 32-bit address bus being coupled to said CPU and said memory to address said memory, said CPU having an address generation means for generation 2^{32} different addresses for addressing said memory over said 32-bit address bus, said 2^{32} different addresses defining a memory address space ranging from location $0000 0000 to location $FFFF FFFF, said locations being in hexadecimal notation, each of said slots having a distinct number in said system and being coupled to said 32-bit address bus to receive addresses for memory disposed on said card in said slot, each of said slots being coupled to distinct identification line means on said main circuit board, each of said distinct identification line means providing a distinct, unchanging signal to the slot to which said distinct identification line means is coupled, each of said distinct signals providing the distinct number of the slot which receives said distinct signal, wherein said computer system has 256 megabytes of memory space ranging from location $X000 0000 to location $XFFF FFFF that is reserved for memory on a card in a slot having a distinct number equal to $X, where $X is any integer from $0 to $E.

3. A personal computer system as in claim 2 wherein $X is any integer from $9 to $E and wherein said main circuit board has 6 slots.

4. A personal computer system as in claim 3 wherein said distinct identification line means comprises four lines each carrying binary values and wherein said 32-bit address bus further includes control signals and is substantially a NUBUS bus.

5. A personal computer system as in claim 4 wherein said computer system further has 16 megabytes of memory space ranging from $FX00 0000 to $FXFF FFFF that is reserved for memory on a card in a slot having a distinct number equal to $X.

* * * * *

United States Patent [19]

Bruffey et al.

[11] **Patent Number: 4,945,475**

[45] **Date of Patent: Jul. 31, 1990**

[54] **HIERARCHICAL FILE SYSTEM TO PROVIDE CATALOGING AND RETRIEVAL OF DATA**

[75] Inventors: **Bill M. Bruffey**, Cupertino; **Gursharan S. Sidhu**, Menlo Park; **Patrick W. Dirks**, Cupertino; Christopher R. McFall, Palo Alto, all of Calif.

[73] Assignee: **Apple Computer, Inc.**, Cupertino, Calif.

[21] Appl. No.: **412,408**

[22] Filed: **Nov. 21, 1989**

Related U.S. Application Data

[63] Continuation of Ser. No. 924,802, Oct. 30, 1986, abandoned.

[51] **Int. Cl.**^{5} **G06F 15/40**

[52] **U.S. Cl.** **364/200**; **364/283.2**; **364/282.1**; **364/283.1**

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,468,728	8/1984	Wang	364/200
4,606,002	8/1986	Waisman et al.	364/300
4,611,272	9/1986	Lomet	364/300
4,677,550	6/1987	Ferguson	364/200
4,734,856	3/1988	Dowis	364/300
4,750,106	6/1988	Aiken, Jr.	364/200
4,760,526	7/1988	Takeda et al.	364/300

OTHER PUBLICATIONS

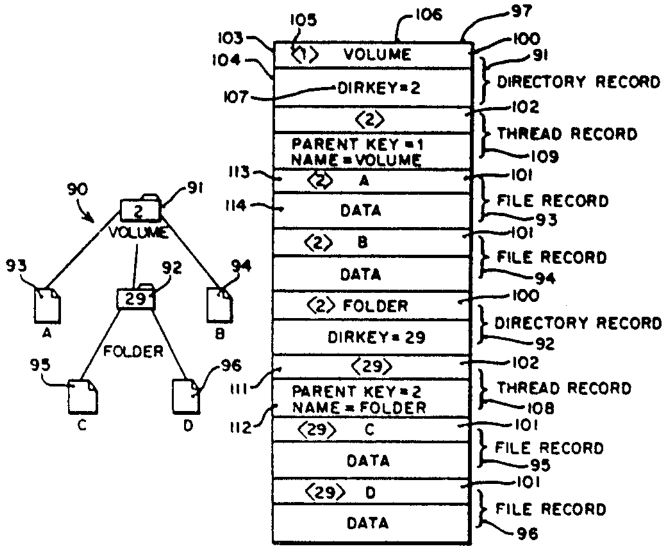
Douglas Comer, "The Ubiquitous B-Tree", Computing Surveys, vol. 11, No. 2, Jun. 1979, pp. 121-136.

Primary Examiner—Gareth D. Shaw
Assistant Examiner—Kevin A. Kriess
Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman

[57] **ABSTRACT**

A hierarchical filing system provides a cataloging of data stored in various locations within a memory device. An upside-down tree type structure provides a downwardly expanding cataloging structure wherein directories provide for further branchings. A branching from a directory is terminated when a file is reached. Each directory is assigned a unique directory identifier value. Then, each file or directory is coupled with the directory identifier value of its parent to provide the interconnection necessary to form the cataloging structure. The complete cataloging structure is organized in the leaf nodes of a B-Tree structure and distributed in an ascending order of the key values to provide a systematic search for a given key. Each file is capable of storing a predetermined number of location description information when data is segmented into non-contiguous segments in memory. A file extents record is used to maintain record of the further segmentation. File location information is kept in the form of file extents descriptors in the leaf nodes of the separate File Extents B-Tree. This extents information is sorted in an ascending order based on a key comprised of a unique file number of a file relative starting block location of the file extent.

6 Claims, 5 Drawing Sheets



Macintosh HFS

U.S. Patent Jul. 31, 1990 Sheet 1 of 5 4,945,475

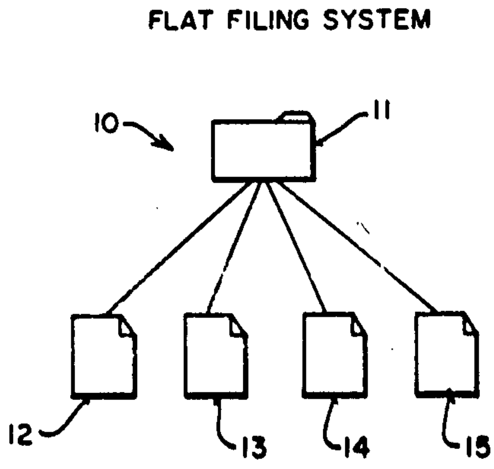


Fig. 1
(PRIOR ART)

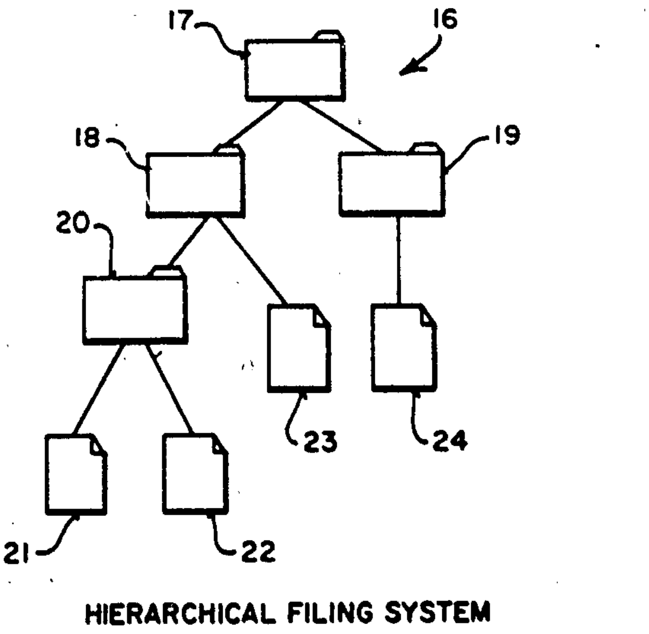


Fig. 2

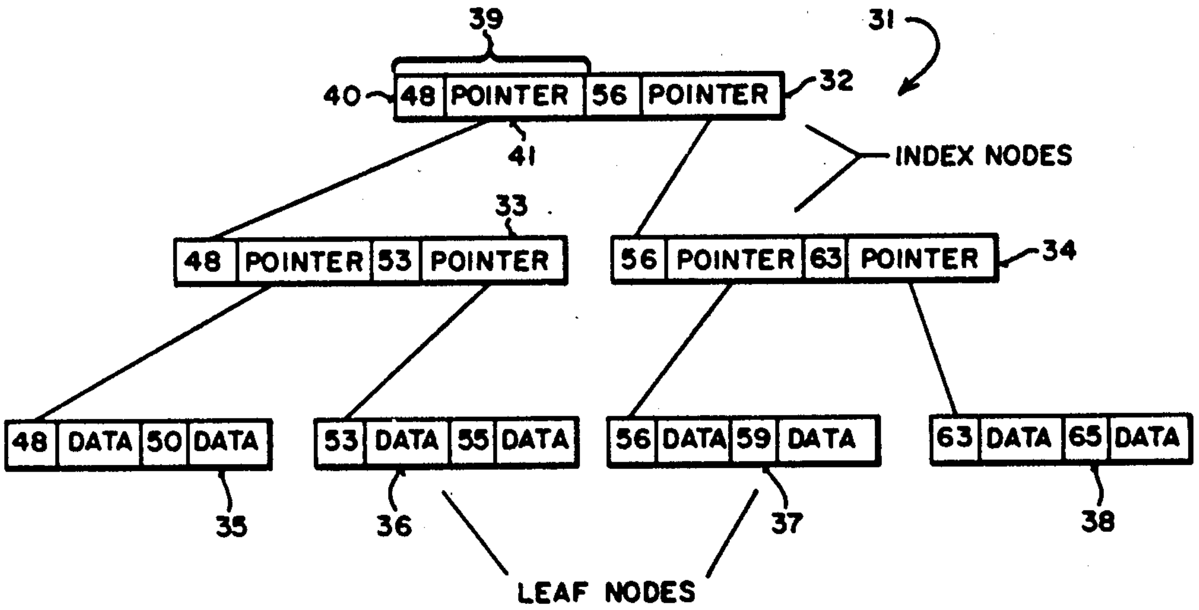


Fig. 3

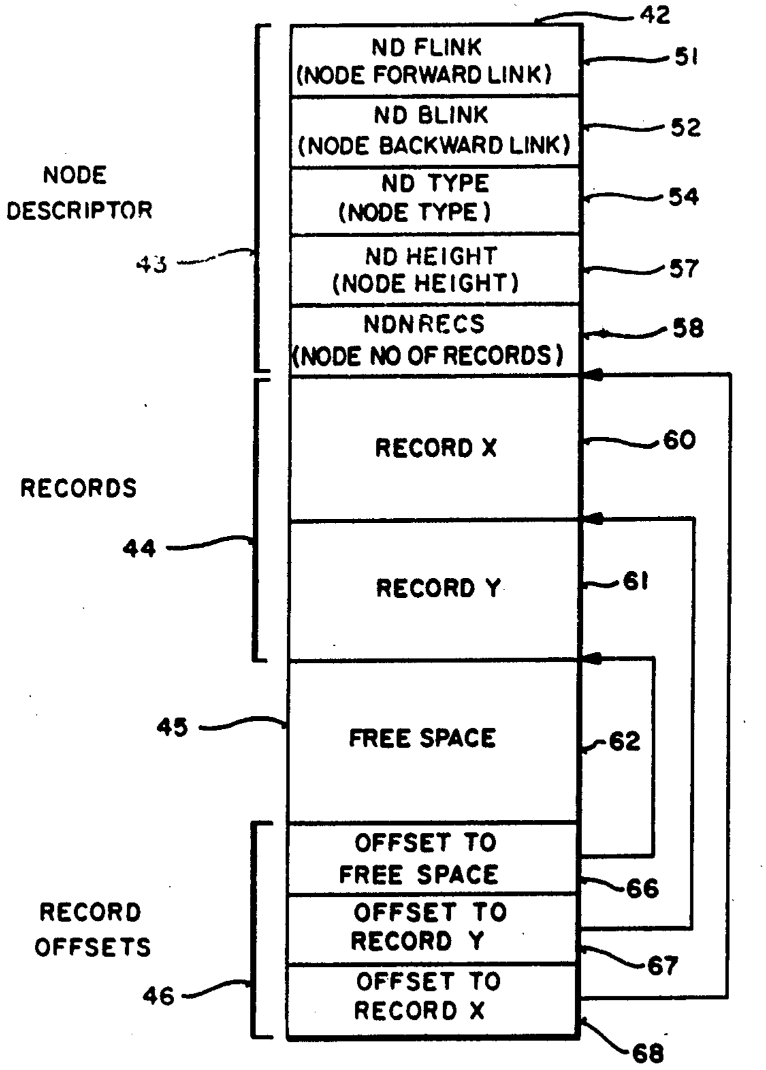


Fig. 4

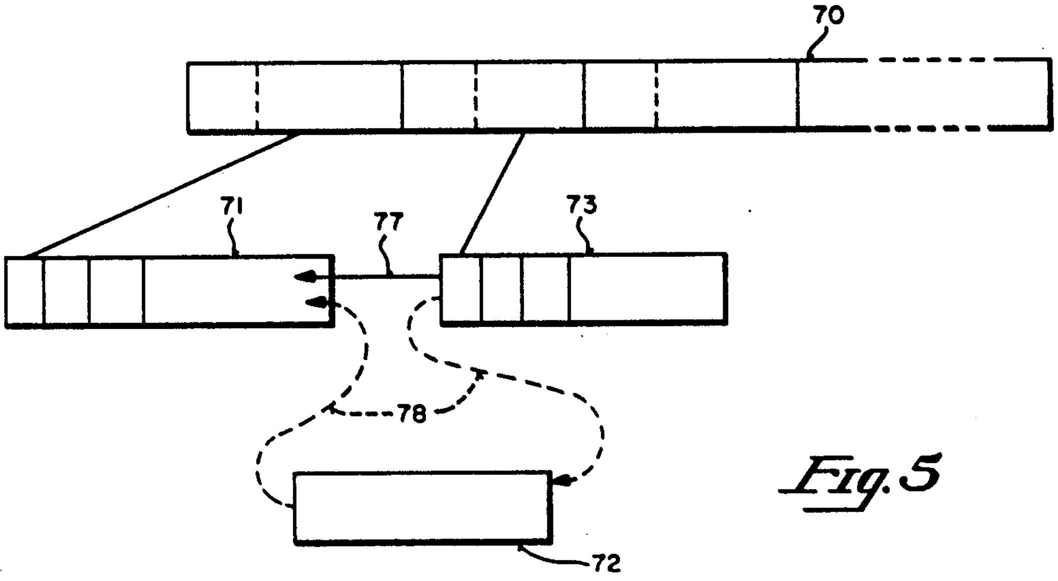


Fig. 5

“APPLE_PAT_4_945_475_03” 88 KB 2000-02-23 dpi: 300h x 300v pix: 1645h x 2570v

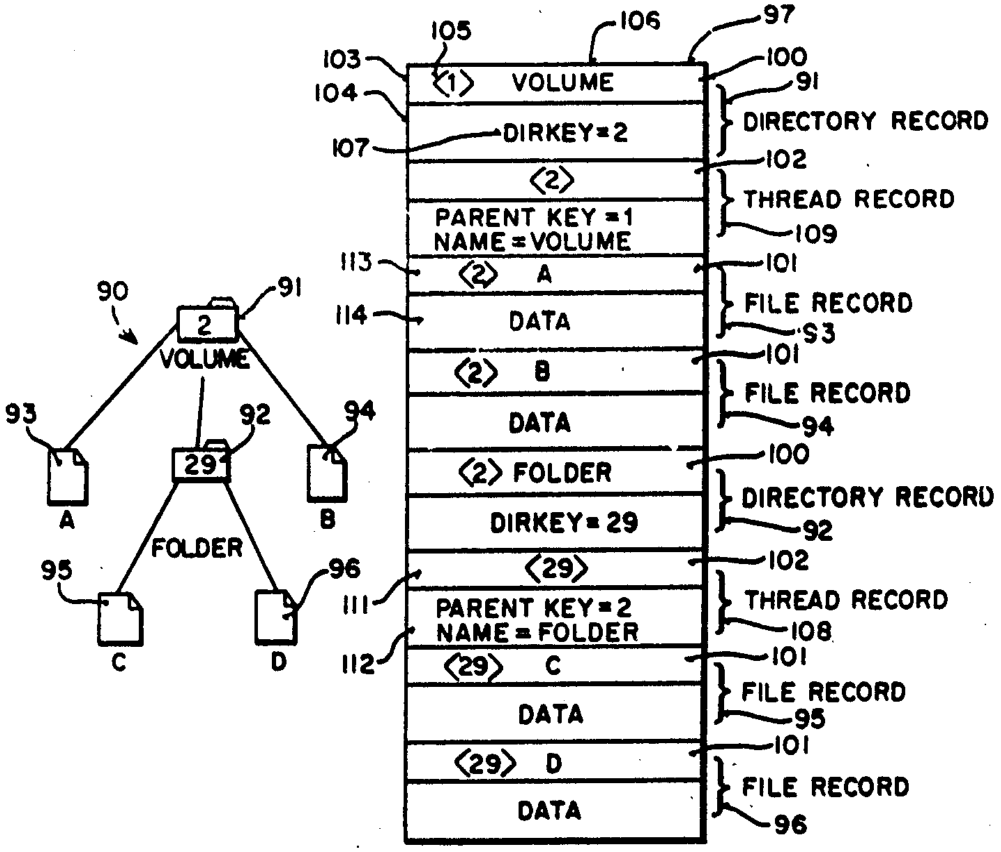

105
106
97
103
104
107
<1> VOLUME
DIRKEY=2
100
91
DIRECTORY RECORD
<2>
PARENT KEY =1
NAME=VOLUME
102
THREAD RECORD
109
113
<2> A
114
DATA
101
FILE RECORD
93
<2> B
DATA
101
FILE RECORD
94
<2> FOLDER
DIRKEY=29
100
DIRECTORY RECORD
92
<29>
111
PARENT KEY=2
NAME=FOLDER
112
102
THREAD RECORD
108
<29> C
DATA
101
FILE RECORD
95
<29> D
DATA
101
FILE RECORD
96
90
91
2
VOLUME
93
92
94
29
A
B
FOLDER
95
96
C
D

Fig. 6

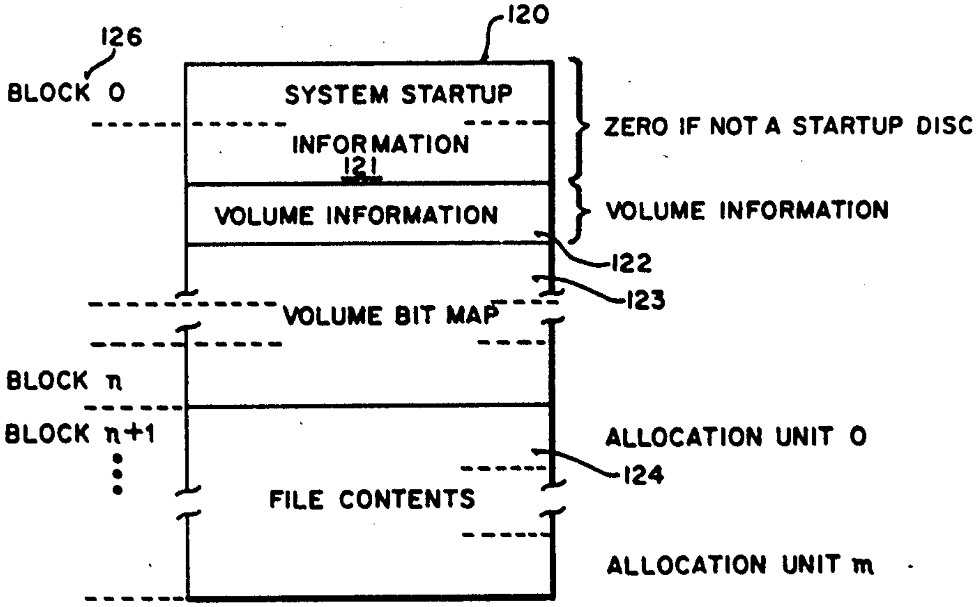
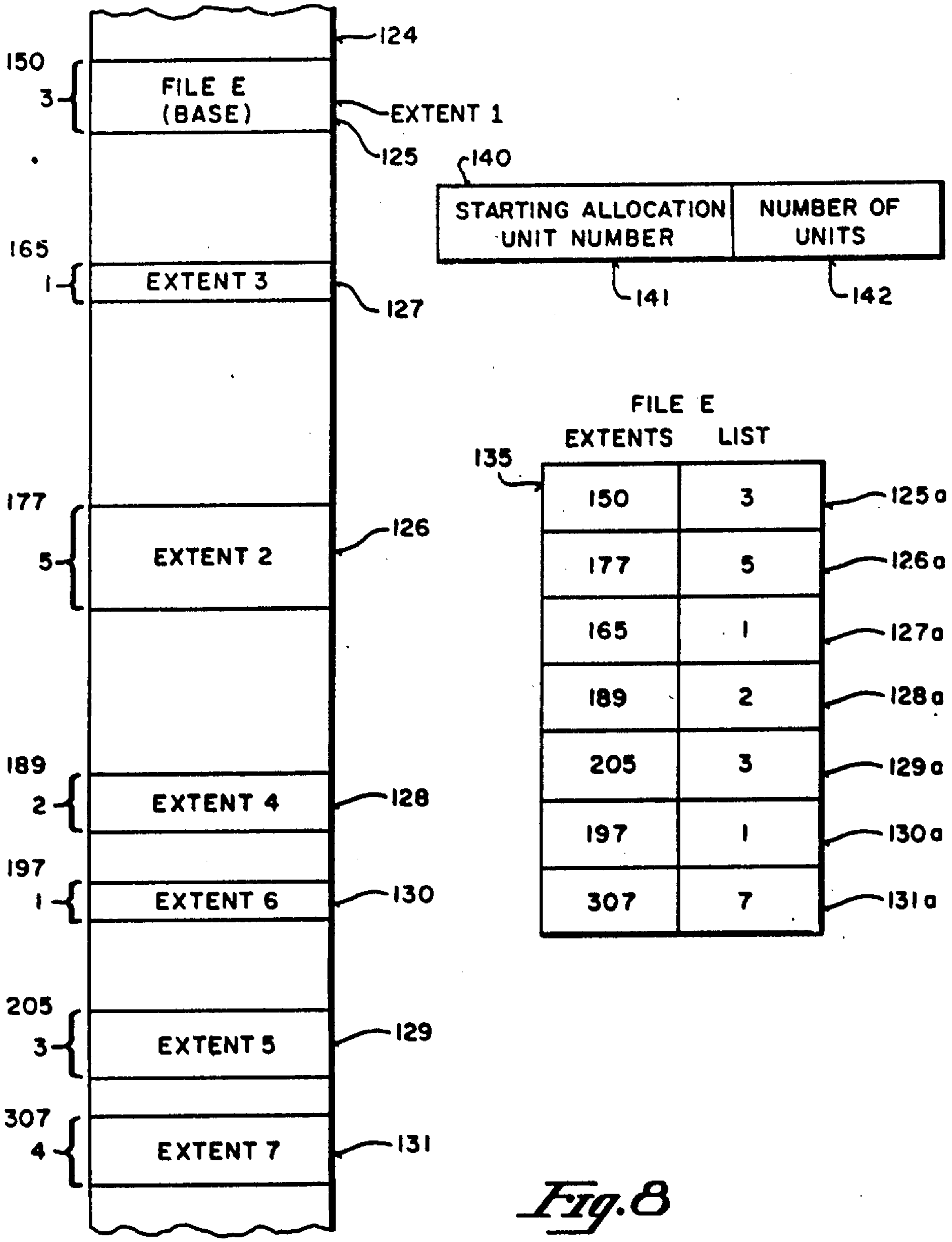

126
120
BLOCK 0
SYSTEM STARTUP
INFORMATION
121
ZERO IF NOT A STARTUP DISC
VOLUME INFORMATION
VOLUME INFORMATION
122
123
VOLUME BIT MAP
BLOCK n
BLOCK n+1
ALLOCATION UNIT 0
124
FILE CONTENTS
ALLOCATION UNIT m

Fig. 7



FILE E EXTENTS LIST

150	3
177	5
165	1
189	2
205	3
197	1
307	7

Fig. 8

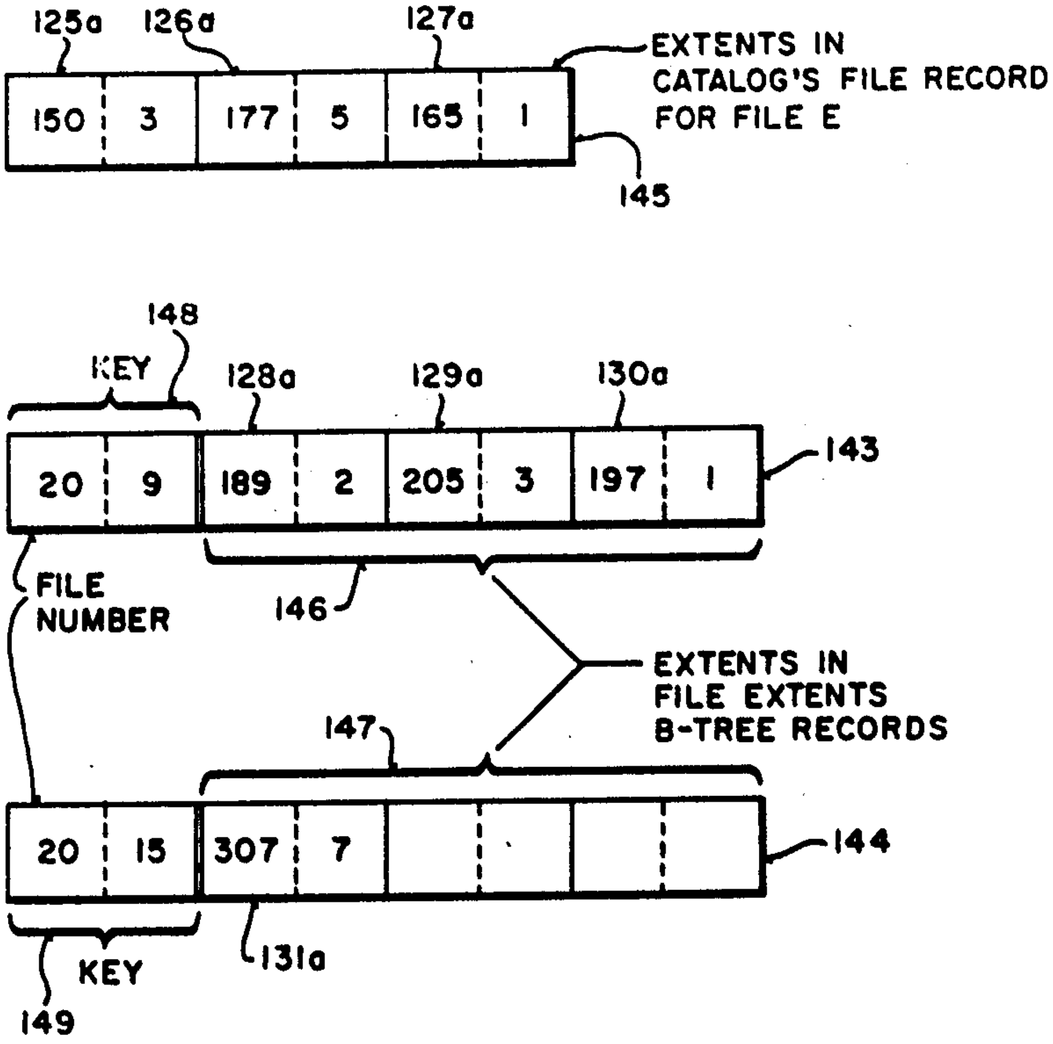


Fig. 9

“APPLE_PAT_4_945_475_06” 50 KB 2000-02-23 dpi: 300h x 300v pix: 1609h x 1621v

HIERARCHICAL FILE SYSTEM TO PROVIDE CATALOGING AND RETRIEVAL OF DATA

This is a continuation of application Ser. No. 924,802 filed Oct. 30, 1986 now abandoned.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to the method of storing and retrieving data using a computer, and more specifically to a hierarchical filing system.

2. Prior Art

In a computer system, information is typically stored as signals on various storage mediums, such as magnetic tapes, disks, semiconductor devices, etc. As storage densities increased with advances in storage device technology, it became possible for a device to store much more information than previously.

When information is stored on a device, it is cataloged so that the same information is later retrieved when desired. Normally, a unique code name is attributed to a particular body of data to differentiate it from others. To retrieve a desired body of data, an appropriate code name associated with that data is used, wherein the device searches for that code name and retrieves the desired data when that code name is found.

It is well-known in the prior art that each separate body of data is termed a file and the cataloging of these files on a device is termed filing. Typically, code names associated with particular data contain pointers which point to areas in memory reserved for mass storage. The various code names and their pointers comprise the cataloging system. When high-density storage devices are used, millions of bits of information are capable of being stored on such a device, which permits hundreds, thousands, and even millions of files to be created. To search through these files in a serial fashion to look for a specific file is time-consuming.

It is appreciated that what is needed is a filing system for a high-density storage medium which rapidly searches and retrieves the desired file stored. Further, with the advent of the personal computer (PC) and the small business computer, where physical size is a concern, it is desirable to have a filing system which may be implemented in a lesser line of program, yet be effectual.

SUMMARY

A method for providing a hierarchical filing system is described. The hierarchical filing system provides a catalog of the data stored in various locations within a memory device. Typically, one cataloging structure is used to organize a volume of memory.

The cataloging structure of the hiearchical filing system is provided by an upside-down tree type structure wherein there is a starting directory which operates as a root directory. Other directories and files emanate as off-spring. A plurality of descendant levels branch downward to provide the hierarchical structure of the catalog. The cataloging structure contains the location information of where the actual data is stored.

The file cataloging system is implemented using a B-Tree. The cataloging information is kept in the leaf nodes of the B-Tree. The non-leaf nodes (index nodes) of the B-Tree contain information that allows searching for particular catalog information by using the code name or key of the corresponding file. Key values,

which are used to identify and catalog various files in the cataloging system, are also used to organize the catalog in the leaf nodes of the B-Tree. The keys are placed in an ascending order for systematic access. Further, the B-Tree grows by using left rotates and left splits with insertion of catalog information about new files from the right to maintain a balanced tree.

When a file's data is stored, additions, deletions and modifications will typically result in non-contiguous physical storage of the data in the memory device. Each of the contiguous segments of the file is known as a file extent. A record of the physical location of the extents for a particular file is maintained in one or more extents records. The hierarchical filing system uses a file extents list to maintain the extents records of the various files on the memory device.

The present invention maintains the first extents record of a file in the cataloging structure, but any further extents records are maintained in a separate file extents list. This file extents list is also implemented in a second B-Tree structure.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a representation of a prior art flat filing system.

FIG. 2 is a representation of a hierarchical filing system of the present invention.

FIG. 3 is a representation of a B-Tree structure of the present invention.

FIG. 4 is a representation of contents of a node for the B-Tree structure of FIG. 3.

FIG. 5 is a representation of a left-split and a left-rotate operation of a B-Tree structure of the preferred embodiment.

FIG. 6 is a representation of a cataloging structure of the preferred embodiment and an organization of the cataloging structure in various nodes of a B-Tree.

FIG. 7 is a representation of a volume allocation mapping in a filing system of the preferred embodiment.

FIG. 8 is a representation of a file extents list of the preferred embodiment and showing various file extents in memory.

FIG. 9 is a representation showing the file extents organization in the Catalog and Extents B-Trees of the preferred embodiment.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The present invention describes a method of storing and retrieving information using a hierarchical filing system. In the following description, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be obvious, however, to one skilled in the art that the present invention may be practiced without these specific details. In other instances, well-known methods have not been described in detail in order not to unnecessarily obscure the present invention.

Referring to FIG. 1, a prior art flat filing system 10 is shown having a directory 11 and files 12–15. For ease of understanding, a directory is shown pictorially as a folder and a file is shown as a sheet of paper with a folded corner. The pictorial representation applies well to an analogy of placing papers into folders (files into directories). In the prior art system 10, there is present a single directory 11, which contains locator information for files 12–15. Each of the files 12–15 contain data which would be associated with a specific body of stored information. In this particular example of a prior art system 10, to access file 15, a serial search is made through directory 11, until the file address of file 15 is located, such sequential search resulting in considerable lapse of time when substantial numbers of files exist in the directory 11. Although in this hypothetical example, directory 11 maintains pointer addresses to four files 12-15, directory 11 will continue to store addresses of subsequent files in a sequential fashion.

FIG. 2 illustrates the architecture of the Hierarchical Filing System (HFS) of the present invention. This particular HFS 16 includes a root directory 17 and files 21-24. The HFS 16 also includes directories 18-20. Each directory is capable of containing files, as well as other directories such as directory 18 containing directory 20. Each directory is a branching node, allowing for none or a plurality of sub-branching nodes. Each directory contains information which permits the branching to occur. The actual data is stored in the files 21-24. Because each file is a termination node, it does not need to maintain further branching information. Instead, each file stores the actual data. Therefore, the directories 17-20 maintain branching information, while files 21-24 contain the stored data.

HFS 16 accesses files 21-24 in a hierarchical fashion so that serial search for the files is not necessary. Assume in the example of FIG. 2 that access to data stored in file 23 is desired. A search of directory 17 reveals that two possible paths exist in seeking the address of file 23. One path from directory 17 leads to directory 18 and the other path leads to directory 19. The desirable path is to directory 18, at which point there are again two paths. The desirable path from directory 18 leads directly to file 23. Although this example is simplistic because of the miniscule number of files shown, one can appreciate the file search time saved when a substantially large number of files are present.

Further, as an example, if file 22 had been chosen, the path from directory 18 would have led to directory 20, at which point two paths exist from directory 20. The desirable path to file 22 from directory 20 then would have been chosen. HFS 16, although shown in a particular form in FIG. 2, may have any number of levels (branchings) down from the root directory 17 as well as any number of branches from a particular directory. However, it is to be noted that all data is stored in the represented files 21-24 which are all located at the termination nodes of HFS 16.

In actuality, the cataloging architecture of the preferred embodiment contains cataloging locator description information in the HFS 16 structure. The catalog entries for files 21-24 contain pointers which provide locator descriptions to locate places in storage area where actual stored data is kept.

B-TREE

The HFS of the present invention is implemented using two B-Tree structures in the preferred embodiment, the Catalog B-Tree and the File Extents B-Tree. A B-Tree structure is well-known in the prior art and is described in *The Art of Computer Programming* Volume 3 (Sorting and Searching); by Donald E. Knuth; at Section 6.4; titled "Multiway Trees"; pp 471-479 (1973). The nodes of a B-Tree contain records, wherein each record is comprised of certain information, either pointers or data, and a key associated with that record.

Referring to FIG. 3, a hypothetical B-Tree is illustrated. A basic feature of the B-Tree 31 is that data is stored only in leaf nodes 35-38. The internal nodes 32-34, also known as index nodes, contain pointers to other nodes such that these index nodes 32-34 provide an index for accessing the data records stored in the leaf nodes 35-38. Each record 39 includes a key 40 and an information segment 41. Within each node, the records are maintained so that their keys are in ascending order. The example B-Tree 31 of FIG. 3 contains hypothetical keys which have been inserted to show the structure of the tree, and the relationship between index nodes 32-34 and leaf nodes 35-38. Leaf node 35 contains key values 48 and 50. The first key of a node is also represented as a key in its ascending node. Therefore key 48, which is the first key of leaf node 35, is also represented as a key within index node 33. Key 53, which is the first key of leaf node 36, is represented as the second key of index node 33. Also, because key 48 is the first key within index node 33, it is again represented as a key within index node 32. This pattern is repeated for each leaf node 35-38 and each ascending index node 32-34 for a B-Tree structure. Although FIG. 3 shows only three levels and two keys per node, any number of keys per node, as well as any number of levels, may be chosen for a particular B-Tree structure. B-Tree 3 of FIG. 3 as drawn is a hypothetical example for illustration purpose only.

When a data record is needed, the key of the desired record is provided. The search begins at the root node, which is also an index node. A search is performed within the node until the record with the highest key that is not higher than the search key is reached. Assume in the hypothetical example of FIG. 3, that data with key 59 is to be selected. The search commences at the root node 32, wherein key 56 is selected because the value 56 is the highest key that is not greater than the search key itself. The pointer of key 56 selects index node 34, wherein the search continues within index node 34. Again, key 56 is chosen because it is the highest key that is not greater than the search key itself (the next key 63 is greater than the search key). The pointer of key 56 in index node 34 selects leaf node 37. Within leaf node 37, another search is made to identify search key 59. When search key 59 is found, its associated information (data) is used.

A particular pointer in an index record leads to another node one level down in the B-Tree 31. For example, node 32 to node 34. The process continues until a leaf node is reached whereupon its records are examined until the desired key is found. If the desired key is not present, then the search stops when a key larger than the search key is reached or when all the records in the leaf node have been examined. The key values may be numeric, alphabetical or alphanumeric.

Referring to FIG. 4, it shows the structure of any of the nodes of a B-Tree of the present invention. Each node 42 includes a node descriptor segment 43, records segment 44, record offset segment 46, and can have a free space segment 45. Each node 42 begins with a node descriptor segment 43. NDNRECS 58 contains the number of records currently in the node. NDTYPE 54 indicates the type of node, either leaf or index node. NDHEIGHT 57 indicates the height of the node in the tree, wherein leaf nodes are chosen as level 1, and the index nodes just above them are at level 2, etc. NDBLINK 52 and NDFLINK 51 are used with B-Tree nodes as a way of quickly moving through the records of the various nodes at a given level. For each node, NDBLINK 52 contains a pointer to the previous node, and NDFLINK 51 contains a pointer to the subsequent node at the same level. In FIG. 3, NDBLINK for node 36 would point to node 35 and NDFLINK for node 36 would point to node 37. Therefore, NDBLINK 52 and NDFLINK 51 are means of locating adjacent nodes without first reversing back up the B-Tree.

The records segment 44 contains the B-Tree's records, each with its key and pointer or data information. In this particular example, there are two records 60 and 61. The records in a node can be of variable length. For this reason, offsets to the beginning of each record are needed. The records segment begins immediately following the node descriptor segment 43. The records are followed by a free space segment 45, which is basically the unused space of the node. Therefore, free space segment may not exist in some instances. The record offset segment 46 at the end of the node contains the offset information for records 60 and 61. Offset 68 contains offset information for record 60 and offset 67 contains offset information for record 61. Offset 66 contains the offset necessary to determine free space 62. Thus the record segment 44 builds downward into the free space segment 45, while the record offset segment 46 builds upward into the free space segment 45 from the opposite end.

If node 42 is an index node, then each record 60 and 61 is comprised of a key and pointer information. Further, NDFLINK 51 and NDBLINK 52 would contain adjacent index node linking pointers. If node 42 is a leaf node, then each record 60 and 61 is comprised of a key and data information. NDFLINK 51 and NDBLINK 52 would also contain leaf node linking pointers. It is also appreciated that although a particular format is illustrated for node 42, the format may be modified readily to include other types of information. Also, in the preferred embodiment data information in the leaf nodes of the HFS catalog B-Tree is used to address locations in memory where the actual data is stored.

Referring to FIG. 5, a specialized B-Tree expansion architecture as implemented in the preferred embodiment is shown. A node 70, which is equivalent to node 42 of FIG. 4, is shown having pointers to two lower-level nodes 71 and 73, which may be index or leaf nodes. Although only two nodes 71 and 73 are shown at the lower level, any number of nodes may reside at this lower level. Also in this particular hypothetical example, nodes 71 and 73 are only partially filled.

For a B-tree to maintain its balance, records must be kept uniformly spaced within the hierarchical structure. An unbalanced tree will result when records are not maintained uniformly in each node or nodes are heavily stacked toward one branch of the B-Tree. The preferred embodiment uses a technique of left rotate and left splits to provide movement of records from one node to another to maintain a balanced Tree. When records are to be transferred to another node, the left rotate operation is used. In this instance, records in node 73 are left rotated to its left adjacent node 71, as shown by arrow 77.

If another node is needed, such as when records in node 73 must be rotated and node 71 cannot accommodate records from node 73, a left split operation is used to insert node 72 to the left of node 73, between nodes 71 and 73. In this instance, node 72 is inserted to link node 71 and node 73, as shown by arrows 78. When node 72 is inserted, appropriate pointer links will be established with its index node 70 as well as adjacent link pointers for nodes 71 and 73. Continually moving data leftward and inserting new data at the right extremities helps keep the B-tree balanced. Because the HFS of the present invention is structured to have the ascending nodes organized in a rightward direction, the balancing is maintained even though the rotates and splits are made toward the left direction. It is appreciated that right splits and rotate operations, or balanced insertions using both right and left operations can be used as well. Although the preferred embodiment uses and attempts to maintain a balanced B-Tree for search efficiency, most any B-Tree structure can be used, including unbalanced B-Tree.

CATALOG TREE

Referring to FIG. 6, a hypothetical catalog 90 is used to illustrate the implementation of cataloging of the preferred embodiment. The structure 90 has a root directory 91 named "Volume". Each directory of the preferred embodiment is assigned a unique numerical identifier known as the directory identifier (DirID). The root directory of catalog 90 has DirID value of 2. Root directory 91 has three branches comprised of directory 92 and files 93 and 94. Directory 92 has a name of "Folder" and a DirID value of 29. In turn, directory 92 has two branches comprised of files 95 and 96. Files 93-96 are named "A", "B", "C" and "D", respectively in this example. The architecture of the directories and files follows the HFS structure as previously explained in FIG. 2. The complete cataloging structure 90 is stored as data records in various leaf nodes of the B-Tree of FIGS. 3 and 4 known as the catalog B-Tree. It is appreciated that the cataloging structure 90, although a tree, is in itself not a B-Tree. The form of structure 90 is actually stored in the various leaf nodes of a B-Tree. It is to be appreciated that the cataloging structure 90 not be confused with the previous description of the B-Tree. Catalog 90 and the B-Tree structure are two separate and distinct structures. The hierarchical structure of the catalog 90 is implemented as a B-Tree structure and stored as data records in leaf nodes of a B-Tree similar to that of FIGS. 3 and 4.

The hierarchical catalog structure 90 is stored in a storage device as shown by a memory map 97 of FIG. 6. Cataloging map 97 is comprised of three possible types of records: directory records 100, file records 101, and thread records 102. Each record 100-102 is comprised of a key 103 and information segment 104, as earlier described in the description of a leaf node of a B-Tree. The key 103 of each record is comprised of a value 105 and a name 106. The key 103 of a directory record, such as that of 91 and 92, is comprised of its directory name 106 and its parent directory's DirID value 105. A information segment 104 of each directory record, such as that of directories 91 and 92 is comprised of the directory's DirID value 107. For directory 92, the directory's DirID has been given the value of 29, and has a name of "Folder". The parent DirID of record 92 has been given the value 2 because directory 92 is an offspring of directory 91 in the structure 90. Directory record 91 has a directory DirID value of 2, with a corresponding name of "Volume". Because directory 91 is a root directory, the parent DirID value has been given the value of 1, wherein the value 1 refers to the foundation of the filing system itself.

A file record, such as file records 93-96, is also comprised of a key 113 and an information segment 114, wherein key 113 is also comprised of a parent DirID value and a name. However, in the information segment 114, the descriptive location information for the actual stored file data is maintained as well as a unique file number. The information segments 114 of file records 93–96 contain the descriptive location of the actual stored data information.

File record 94, having a file name of B, and file record 93, having a file name A, both have a parent DirID value of 2. The parent DirID value of 2 signifies that files A and B are direct offsprings of directory "Volume" having a DirID value of 2. File 95, having a name C, and file 96, having a name D, have parent DirID values of 29, which reflect the origination of files C and D as offsprings of directory 29 labeled "Folder", having a DirID value of 29. Therefore, by looking at any file or a directory record's key 103, the stored information provides the identification of the name of that particular record as well as the DirID value of the parent node.

To provide the interconnection of the different branches, a thread record 102 is provided for each directory. The key of a thread record contains a DirID value and a null-name, which is equivalent to having no name at all. In the example of FIG. 6, thread record 108 provides the connection between the directory "Folder" and files C and D. In the key 111 of thread record 108, only the directory DirID value of "Folder" is given. In the information segment 112 of thread record 108, the DirID of "Folder"'s parent and the directory's name "Folder" are given. Therefore, when file C, having a parent DirID 29 attempts to link to its immediate parent directory 92, which has a DirID of 29, the thread record 108 provides the name (Folder) of the parent directory 92, as well as the parent DirID value of directory 92, which is equal to 2.

Equivalently thread record 109 provides the name (Volume) of directory 91 as well as its parent directory DirID value for the three offsprings 92–94 of directory 91. By having directory records 91–92, file records 93–96, along with thread records 108–109 for each directory, the cataloging structure 90 is interconnected into a HFS, wherein the descriptive location information for the actual stored data is stored in file records 91–92 as shown in the structure 97 of FIG. 6.

By implementing the cataloging structure 90 using a B-Tree structure, the hierarchical configuration of structure 90 is easily stored in the leaf nodes of a B-Tree of the earlier description. For example, when file C is to be accessed by a computer, the system will implement a B-Tree search. Referring to the catalog example 90 of FIG. 6, when file with name C is to be found, the search path must be specified for this search. This can be given in terms of a sequence of the names of all directories on the path from the root to the said file, thus "Volume", followed by "Folder", and finally "C". The search begins by finding the directory record in the Catalog B-Tree that corresponds to "Volume". Its name is "Volume" and since it is the root, its parent DirID value is 1. The catalog B-Tree is searched for a directory record with key <1> Volume; thus, directory record 91 is found. Its information segment then provides the DirID value 2 of this directory. Now a search is made through the B-Tree for the record with key <2> Folder which leads to the directory record 92, whose information segment provides this directory's DirID value of 29. Thus now a search of the B-Tree is made to find the data record with key <29>C. This immediately leads the search to the file record 95, whose information segment contains the information about the physical location of the data contained in the desired file.

It will be appreciated that the specification of the file of the above example could start with the DirID value of any directory on the path from the root to the desired file, and would then consist of this DirID value and the sequence of names of the directories on the balance of the path from that directory to the desired file. The search mechanism followed is an obvious variant of the one indicated above.

Although cataloging structure 90 is a simplified structure and FIG. 6 only shows the presence of a single structure having a single root directory 91, a cataloging structure may be enlarged manyfold. The preferred embodiment uses one HFS cataloging structure per memory device, such as a disk. However, such a disk can be partitioned and an HFS catalog assigned to each such partition.

The catalog records of structure 97 of FIG. 6 are stored as the data records in the leaf nodes 42 of FIG. 4 of a catalog B-Tree. These records are inserted and maintained in the catalog B-Tree in ascending alphanumeric order. Thus, if the leaf nodes of the B-Tree are traversed from left to right, the data records will be encountered in the order shown in structure 97 of FIG. 6. This order maintains the records in ascending order first by the DirID value part of the key. Then, among records with the same DirID value in their keys, the order is alphabetical on the name part of the key.

It is also appreciated that other pertinent information may be stored in the various records besides what has been disclosed in FIG. 6. For example, directory and file records of the present invention maintain flags, date and time of creation of the directory or the file, as well as the date and time of last modification. Also, file records include such items as flags for locking the file, values to set logical and physical end of files, and size of the file.

FILE EXTENTS TREE

As already noted, the catalog B-Tree's file record of a particular file contains information about the locations in the memory device where the file's data is stored. The memory device is considered to be a sequentially numbered collection of blocks. A series of contiguous memory blocks is called an extent. Ideally, a file would be stored in a single extent having a contiguous memory allocation space. However, due to the size of certain files, as well as subsequent additions, deletions and modifications to existing files, files are usually stored in more than one allocated area of the memory. Except in the case of preallocated or small files, the contents of a particular file are usually stored in more than one extent, separated into non-contiguous sections on a volume. Each file extent can be identified by an extent descriptor. Thus, the complete location information of a particular file is a sequential extents list consisting of the extent descriptors of the various extents containing the file's data.

The file extents list of the present invention is organized also as a B-Tree, known as the File Extents B-Tree, and records the volume location and size of the various extents that comprise the files. Although most any memory allocation system can employ the file extents record of the present invention, a specific memory allocation system is described to illustrate the file extents record of the preferred embodiment.

Referring to FIG. 7, a memory volume 120 which is a portion of a memory device, such as a hard disk, is shown. Volume 120 is segmented into a number of logical blocks 126. Typically, each logical block 126 is comprised of a predetermined fixed number of bytes, such as 512 bytes for the preferred embodiment. A fixed number of logical blocks starting at block 0 and ending at block n is reserved for volume information. The balance of the memory device starting at block n+1 is available for data storage and this storage area is separated into allocation units, wherein each allocation unit is comprised of one or more contiguous logical blocks.

Volume 120 includes four areas 121–124. System start-up area 121 contains certain configurable system parameters which are well-known in operating a disk or other memory devices. Volume information area 122 contains information regarding the housekeeping parameters of the volume, such as number and size of each allocation unit. Volume bit map 123 maintains record of each allocation unit on the volume 120 and uses a bit map to designate use or non-use of each allocation unit.

Commencing at block n+1, a file content area 124 extends to the end of the Volume 120. File content area 124 is separated into a number of allocation units, wherein each allocation unit is comprised of a fixed number of logical blocks. While the bit map 123 maintains volume space management, it does not provide file mapping. The file mapping function is provided by the file extents lists.

Referring also to FIG. 8, a portion of file contents area 124 is shown containing information attributed to a file labeled file E. In this hypothetical example the entire contents of file E are separated into seven extents 125–131. The first portion of the file is stored in base extent 125, the subsequent portions of the file are distributed accordingly in extents 2–7 which are labelled 126–131. File E has seven extents 125–131 which are not physically contiguous. To maintain file extents information an extent descriptor 140 is used for the base extent 125 and each of the subsequent extents 126–131 of file E.

Extent descriptor 140 is comprised of a starting allocation unit number 141 and number of allocation units 142. File E extents list 135, which is comprised of seven extent descriptors 125a–131a, provides information as to the address and length of each extent 125–131 of file E. For example, the fourth extent 128, which has a starting allocation address of 189 and is only two allocation blocks long, has a value of 189 in field 141 and a value of 2 in field 142 of descriptor 128a.

Extents descriptors of all files in a volume are maintained in the present invention in the data records contained in the leaf nodes of B-Tree such as of FIGS. 3–5. This tree is known as the File Extents B-Tree and is a separate B-Tree from the earlier described catalog B-Tree. Each data record of this extents B-Tree consists of a key and an information segment as before in the discussion of FIGS. 3–5. The information segment of a File Extents B-Tree data record is comprised of a sequence of extents descriptors of a particular file. The maximum number of extents descriptors in such a record can vary from implementation to implementation, but in the preferred embodiment is set to three. The key of the File Extents B-Tree record consists of two fields: the file number of the particular file and the file relative posistion of the starting block of the first extent descriptor in that record. These extents records are kept in the leaf nodes of the Extents B-Tree sorted in ascending order first on the file number field and then on the file relative position of the starting block. This allows efficient search through the B-Tree for the location information of data at a particular file relative position.

In actuality, the preferred embodiment stores three extents descriptors, base plus two subsequent extents descriptors, the information data segment 114 of the file's catalog B-Tree record such as 94 of FIG. 6. Therefore, in the example of FIG. 8, extent descriptors 125a, 126a and 127a are kept in the information segment of the cataloging structure and extents 128a–131a are kept in the File Extents B-Tree as shown in FIG. 9. Permitting limited extent information to be kept in the data segments of a cataloging structure permits faster access to data. Only when a file contains four extents or more, will it need to consult the File Extents B-Tree. It should be appreciated that the number of extents which are kept in the file's Catalog B-Tree record without using a File Extents B-Tree is arbitrary and can be changed without departing from the spirit and scope of the invention.

Also referring to FIG. 9, it shows a catalog file record 145 and File Extents B-Tree records 143 and 144. As explained in the structure of B-Trees of the present invention, each record 143 and 144 is comprised of a key 148 and 149 and extents list 146 and 147, respectively. To locate a certain portion of the data of a particular file, first the Catalog B-Tree is searched for the corresponding file record. From this file record's information segment, the file number is extracted. Also, the first three extent descriptors in the information segment of the catalog B-Tree file record are examined. If the required file data is contained within the corresponding extents, then the location information is now readily available. If however, the desired file data is located in extents beyond the three in the catalog's file record, then a search is made of the File Extents B-Tree using as a search key the file number and the computed file relative block position of the desired data. This search will lead to the file extent's B-Tree record containing the desired location information.

The example using file E is comprised of 22 blocks and having an arbitrary file number equal to 20. The extent descriptors contained in the catalog file record 145 for file E provide the location information for the first 3 extents which in turn comprises the first 9 blocks (3+5+1) of the file. The location information for the remaining 13 blocks (2+3+1+7) of the file is contained in two data records 143 and 144 within the File Extents B-Tree. Assume that the desired data is at file relative block position 13 within file E. The extent descriptors contained in the file's catalog record are examined first. Since relative block 13 is greater than the number of blocks located by the extent descriptors in the file's catalog record, the File Extent B-Tree is searched. The key used for the B-Tree search for relative block position 13 is <20,13>.

Since the key value of "13" is greater than the value "9" of key 148 for the first Files Extents B-Tree record 143 for file E and is less than the value "15" of key 149 for the second record 144, the search results with a "not found" result but positions to the second B-Tree record 144. By retrieving the previous record 143 of key 148, the extent descriptor for relative block 13 is obtained. The value of "9" for key 148 is derived because extents list 146 starts at the tenth relative block (allocation unit number 9). The value of "15" for key 149 is derived because extents list 147 starts at the sixteenth relative block (allocation unit number 15).

IMPLEMENTATION

The HFS of the present invention is implemented in a computer which is coupled to a memory device, such as a disk, having an ability of storing millions of bits of information, although any storage medium can use the HFS. Typically, the HFS of the present invention provides the cataloging of various groupings of data, such as files, which are stored on the disk.

The preferred embodiment implements data storage by the use of a cataloging structure previously described to catalog data stored on a large capacity memory device. It also maintains a file extents record of up to three extents per file in the catalog. Subsequent extent information is stored in a separate file extents record. Both the catalog record and the extents record are maintained using two B-Trees of the earlier described B-Tree structure.

The HFS as described in the preferred embodiment is controlled by a combination of hardware and software in a computer system. The HFS controlling routines are stored in a separate storage device than the device used for storing the actual data. The preferred embodiment stores the routines in a read only memory (ROM), although most any storage medium may be used.

Thus, a hierarchical filing system for use with a large capacity memory device in described.

We claim:

1. In a computer, a hierarchial filing system to provide cataloging and retrieval of data stored on a storage device, said hierarchial filing system comprising:

a memory for storing a program for said cataloging and retrieval;

a processor coupled to said memory and said storage device for processing an organizing means to catalog and retrieve said data; said processor comprising;

said program for organizing said data on said storage device into a hypothetical catalog which has a root directory, a plurality of branching directories arranged at various subsequent levels from said root directory, wherein some of said branching directories branch from other of said branch directories; said branching directories being interconnected such that for each of said branching directories there is only a singular path from itself to said root directory; and wherein some of said branching directories have at least one file, each file corresponding to a representation of a predetermined portion of said stored data;

an assigning means for assigning a unique identification value to said root directory and each of said branching directories, and assigning an identification name to each of said files, root directory and branching directories, wherein each of said branching directories and files are each provided with a key comprised of its identification name and its next higher level directory identification value;

a list forming means for forming a linear list of files and directory entries such that said file and directory entries are ordered by said keys, such that said root directory being the highest level and files being the lowest level; and said interconnection of each of said singular path is provided by each file and branching directory identification name being associated with directory identification value of its next higher level;

a structure forming means for forming a B-Tree indexing structure having a beginning node, a plurality of indexing nodes and a plurality of terminating nodes, and wherein said linear list is stored in said terminating nodes of said B-Tree indexing structure.

2. The hierarchial filing system defined in claim 1, wherein said memory for storing said program is a read only memory.

3. In a computer system where data is to be catalogued when stored into a memory device, a method performed by the computer system for providing a hierarchial filing system to catalogue said data into a volume of said memory device for subsequent retrieval, comprising the steps of:

creating a root directory, a plurality of subdirectories and a plurality of files;

organizing said root directory, subdirectories and files into a hypothetical catalog wherein said root directory is at a topmost level and said subdirectories are arranged at various subsequent levels from said root directory, some of said subdirectories branch from other of said subdirectories, but said subdirectories being interconnected such that for each of said subdirectories there is only a singular path from itself to said root directory, and wherein each of said files being interconnected to branch from a certain one of said subdirectories only, such that for each file there is only a singular path from itself to said root directory;

assigning a unique numerical directory identification value to said root directory and to each of said subdirectories in said volume;

assigning an identification name to said root directory and to each of said subdirectories and files, such that no two subdirectories branching from a root directory has a same name, no two subdirectories branching from another subdirectory has a same name, and no two files branching from one of said directories has a same name;

wherein each of said subdirectories and files are each provided with a key comprised of its identification name and its next higher level directory identification value;

forming a linear list of files and subdirectory entries such that said file and subdirectory entries are ordered by said keys, such that said root directory being the highest level and files being the lowest level; and said interconnection of each of said singular path is provided by each file and subdirectory identification name being associated with directory identification value of its next higher level;

forming a B-Tree indexing structure having a beginning node, a plurality of indexing nodes, and a plurality of terminal nodes;

storing said linear list in said terminal nodes of said B-Tree structure in alphanumerical order according to said numerical directory value;

assigning said identification name of a given file to a respective portion of said data;

storing said data;

placing memory location information in said files, wherein for each given file its memory location information locates its respective portion of said data stored in said memory device.

4. The method as described in claim 3 wherein said step of forming said B-Tree indexing structure further comprises the step of forming a B-Tree structure wherein said beginning node comprises a root node of said B-Tree, said indexing nodes comprise branch nodes of said Be-Tree, and said terminal nodes comprise leave nodes of said B-Tree.

5. The method as described in claim 4 wherein said step of placing location information in said files comprises the step of providing a plurality of extent pointers, each extent pointer pointing to a location of a portion of said data stored in said memory device corre-

sponding to said given file such that non-contiguous data segments are made to correspond to each said file.

6. The method as described in claim 5 further comprising the step of forming a second B-Tree structure to store a linear list of additional extent pointers for those files which have more extent pointers than that which can be stored in each file, said linear list of additional extent pointers being stored in terminal nodes of said second B-Tree structure by having each additional extent pointer stored in one of said terminal nodes.

* * * * *

United States Patent [19]

Moore

[11] **Patent Number:** **4,958,304**

[45] **Date of Patent:** **Sep. 18, 1990**

[54] **COMPUTER WITH INTERFACE FOR FAST AND SLOW MEMORY CIRCUITS**

[75] Inventor: **Robin B. Moore**, Fremont, Calif.

[73] Assignee: **Apple Computer, Inc.**, Cupertino, Calif.

[21] Appl. No.: **405,397**

[22] Filed: **Sep. 12, 1989**

Related U.S. Application Data

[63] Continuation of Ser. No. 20,599, Mar. 2, 1987, abandoned.

[51] **Int. Cl.**^{5} **G06F 3/14**; G06F 15/20

[52] **U.S. Cl.** **364/521**; 364/900; 364/927.2; 364/964.2; 364/964

[58] **Field of Search** ... 364/200 MS File, 900 MS File, 364/521, 518; 340/799, 801, 802, 798

[56] **References Cited**

U.S. PATENT DOCUMENTS

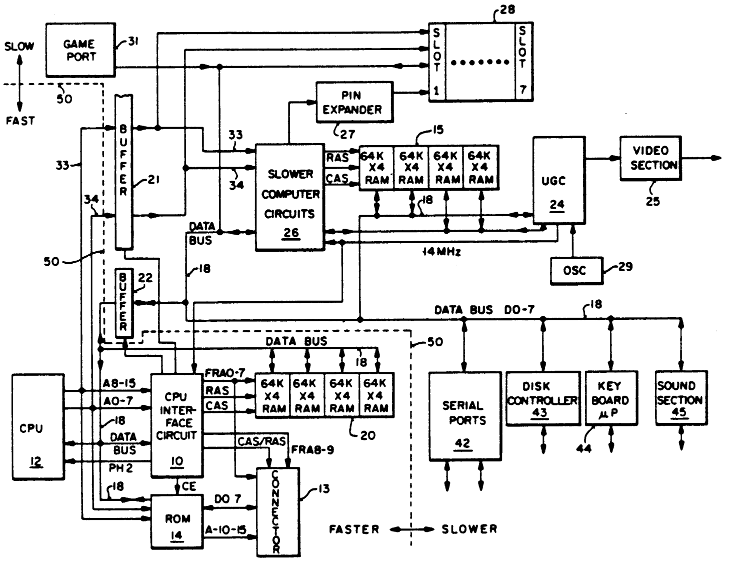
3,231,863	1/1966	Ulfsparre	364/200
4,110,823	8/1978	Cronshaw et al.	364/200
4,150,364	4/1979	Baltzer	340/703
4,298,931	11/1981	Tachiuchi et al.	364/200
4,366,540	12/1982	Berglund et al.	364/200
4,388,621	6/1983	Komatsu et al.	340/802
4,435,775	3/1984	Brantingham et al.	364/900
4,484,261	11/1984	Brantingham	364/200
4,486,856	12/1984	Heckel et al.	364/900
4,511,965	4/1985	Rajaram	364/200
4,609,996	9/1986	Kummer	364/900
4,628,467	12/1986	Nishi et al.	364/521
4,639,721	1/1987	Eto et al.	340/747
4,682,297	7/1987	Iwami	364/521
4,747,081	5/1988	Heilveil et al.	365/219
4,780,712	10/1988	Itaya et al.	340/747
4,791,580	12/1988	Sherrill et al.	364/521

Primary Examiner—Eddie P. Chan
Assistant Examiner—Kevin A. Kriess
Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman

[57] **ABSTRACT**

A CPU with an interface to two different RAMs which operate at different rates. The interface circuit includes a decoder which examines the addresses from the CPU and determines whether a faster cycle or slower cycle is needed. The slow RAM provides video signals to a video display. The fast RAM includes an image of the video signals stored in the first RAM. When the video signals are read by the CPU, they are read only from the fast RAM, however, when it is necessary to update the video signals, they are written into both the slow and fast RAMs.

15 Claims, 9 Drawing Sheets



Apple 2GS

U.S. Patent Sep. 18, 1990 Sheet 1 of 9 4,958,304

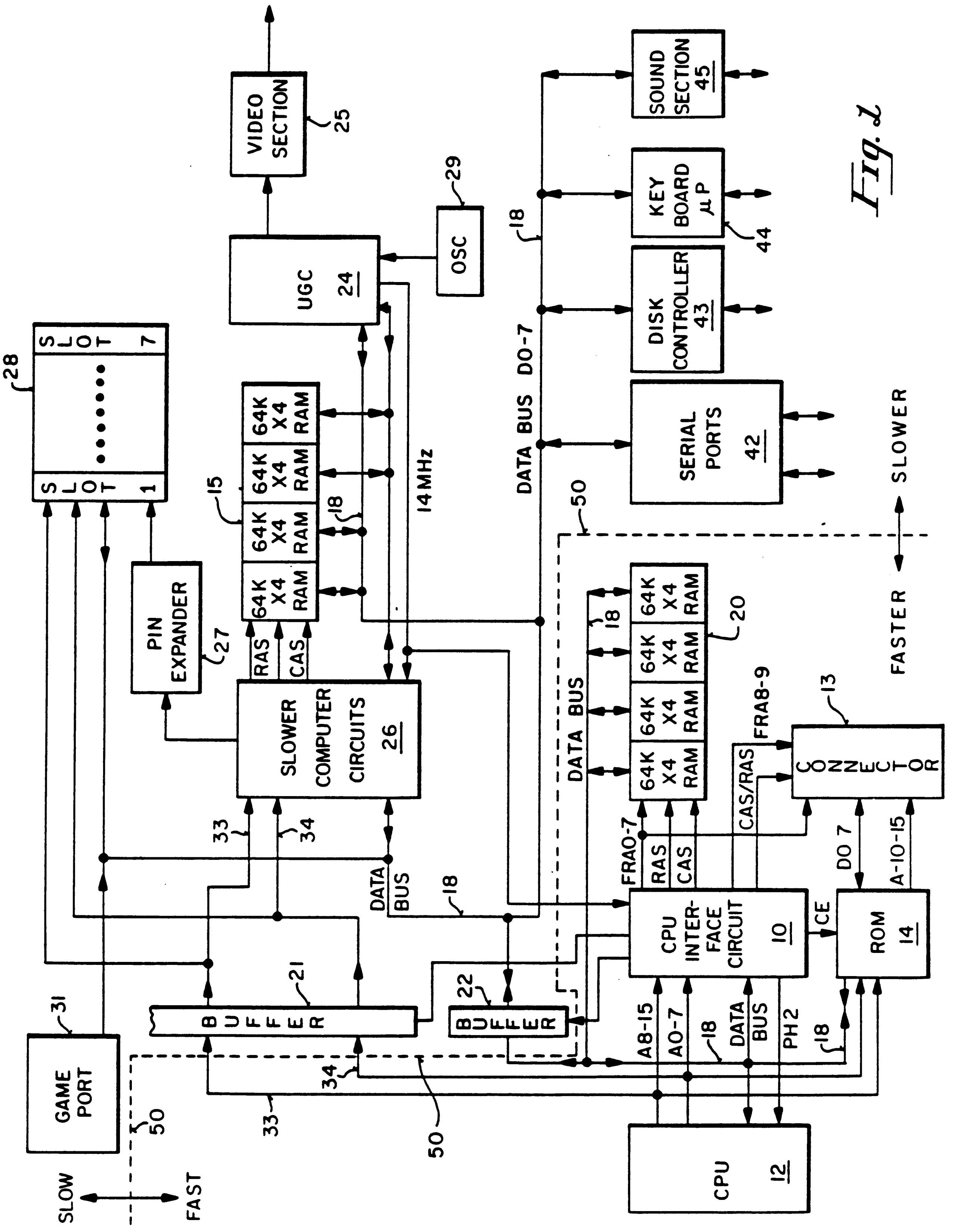


Fig. 1

“APPLE_PAT_4_958_304_02” 455 KB 2000-02-23 dpi: 600h x 600v pix: 3920h x 5300v

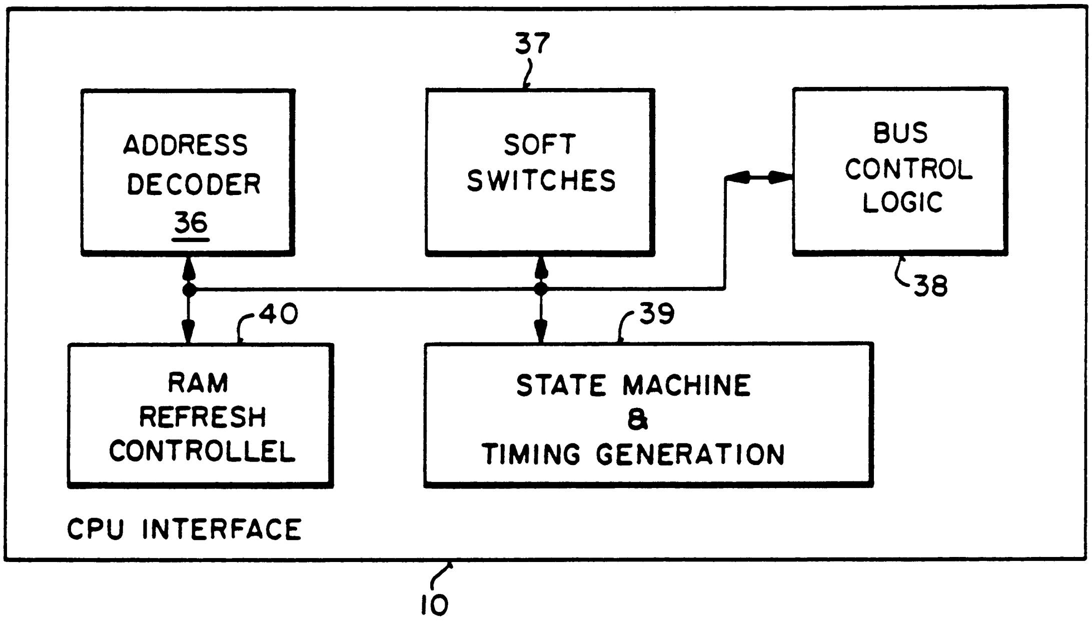

37
ADDRESS DECODER 36
SOFT SWITCHES
BUS CONTROL LOGIC
38
40
39
RAM REFRESH CONTROLLEL
STATE MACHINE & TIMING GENERATION
CPU INTERFACE
10

Fig. 2

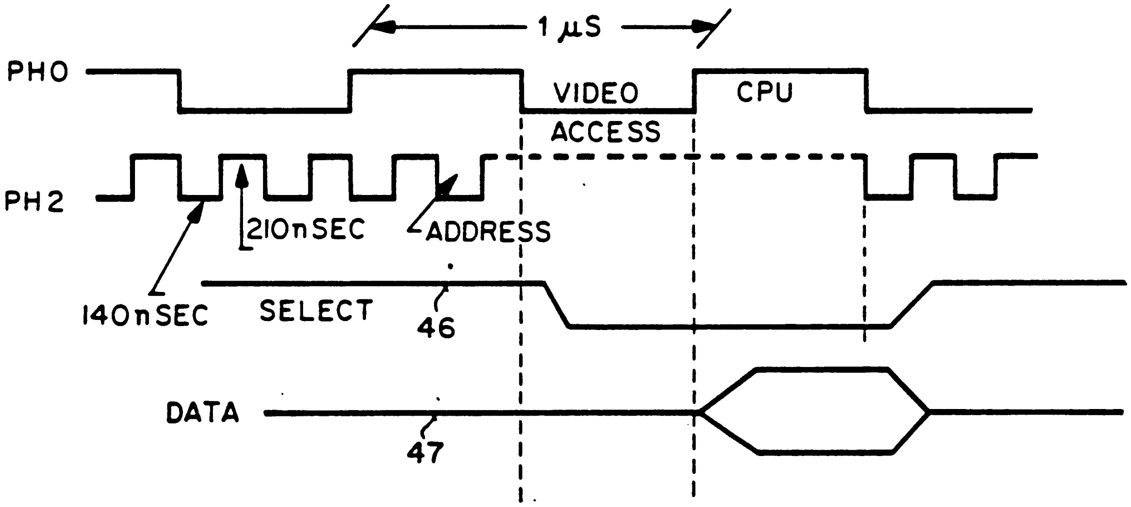

1 µS
PHO
VIDEO ACCESS
CPU
PH2
210nSEC
ADDRESS
140nSEC
SELECT
46
DATA
47

Fig. 3

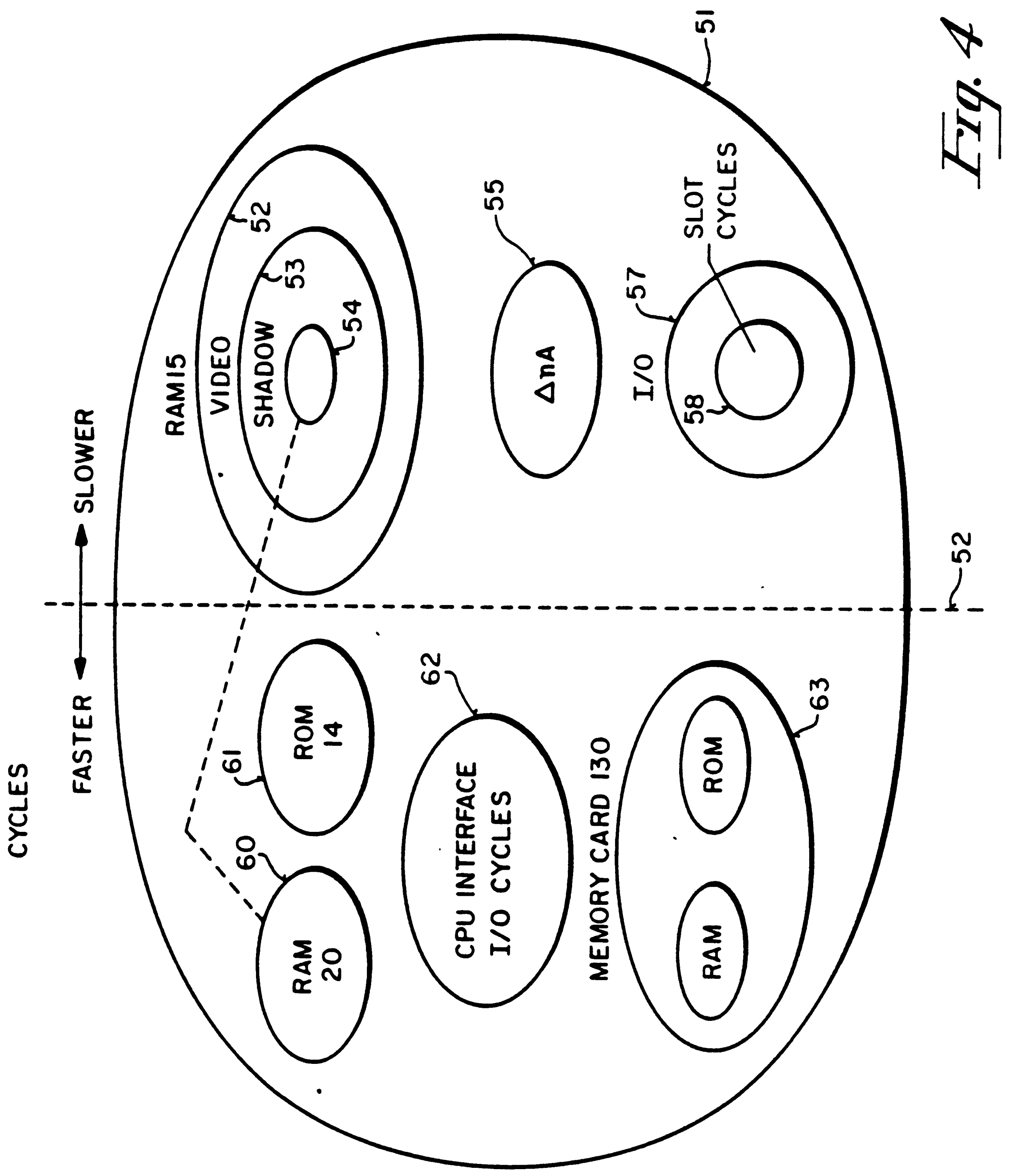

51
52
53
54
RAM15
VIDEO
SHADOW
55
ΔnA
SLOT
CYCLES
57
I/O
58
SLOWER
FASTER
CYCLES
52
62
63
61
ROM
14
CPU INTERFACE
I/O CYCLES
MEMORY CARD 130
ROM
RAM
60
RAM
20

Fig. 4

U.S. Patent Sep. 18, 1990 Sheet 4 of 9 4,958,304

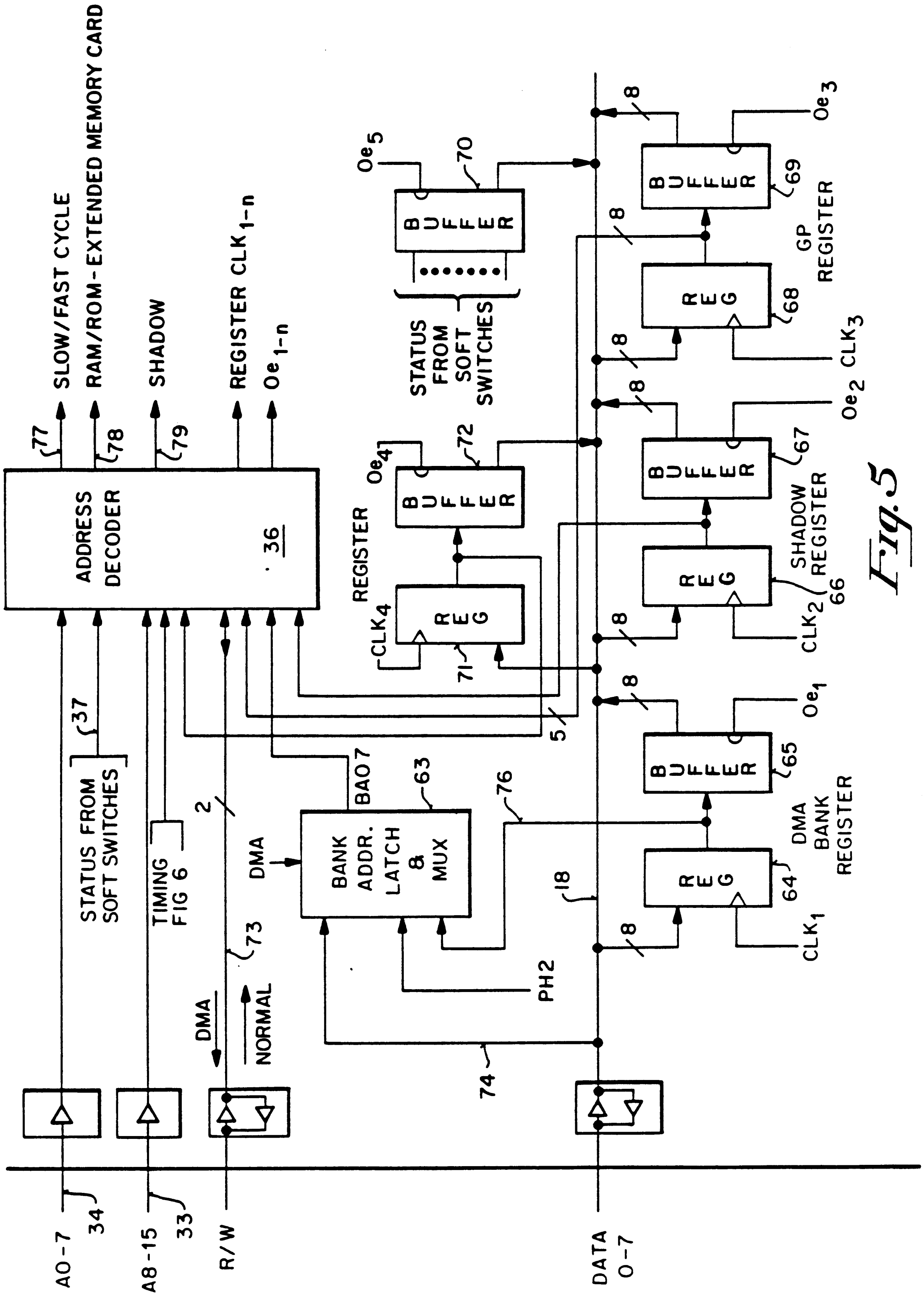


Fig. 5

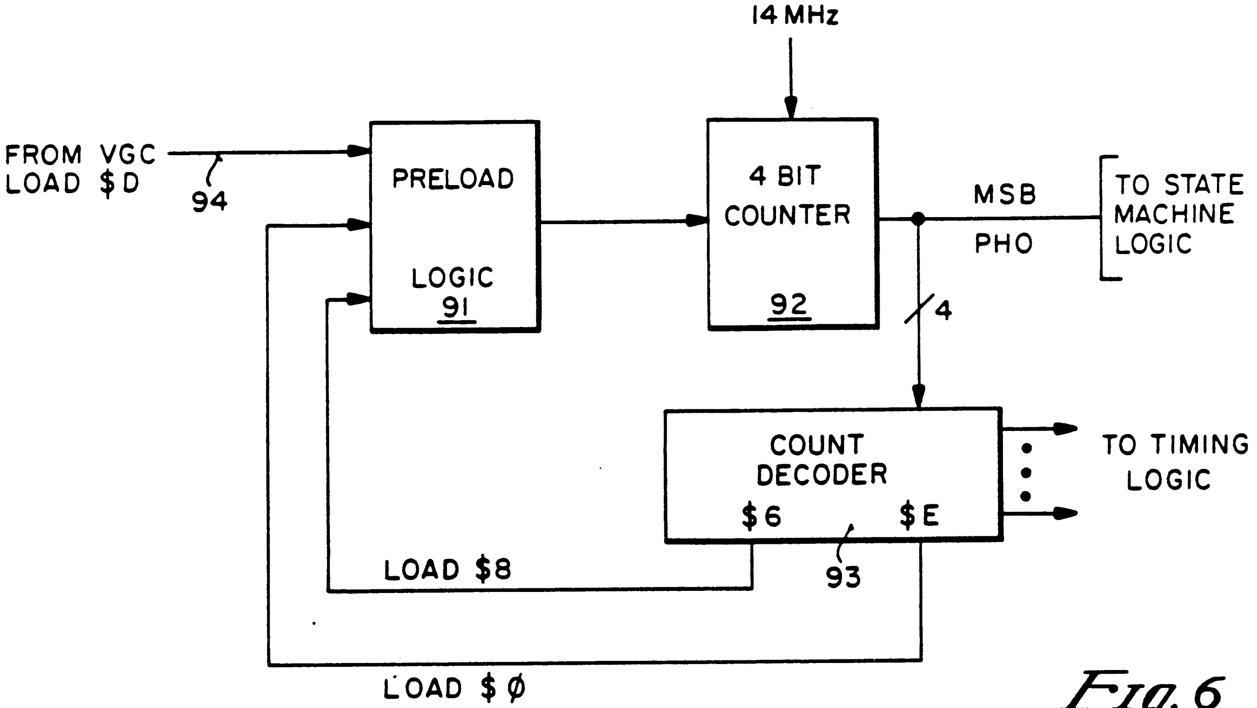

14 MHz
FROM VGC
LOAD $D
94
PRELOAD
LOGIC
91
4 BIT
COUNTER
92
MSB
PHO
TO STATE
MACHINE
LOGIC
4
COUNT
DECODER
$6
$E
93
TO TIMING
LOGIC
LOAD $8
LOAD $Ø

Fig. 6

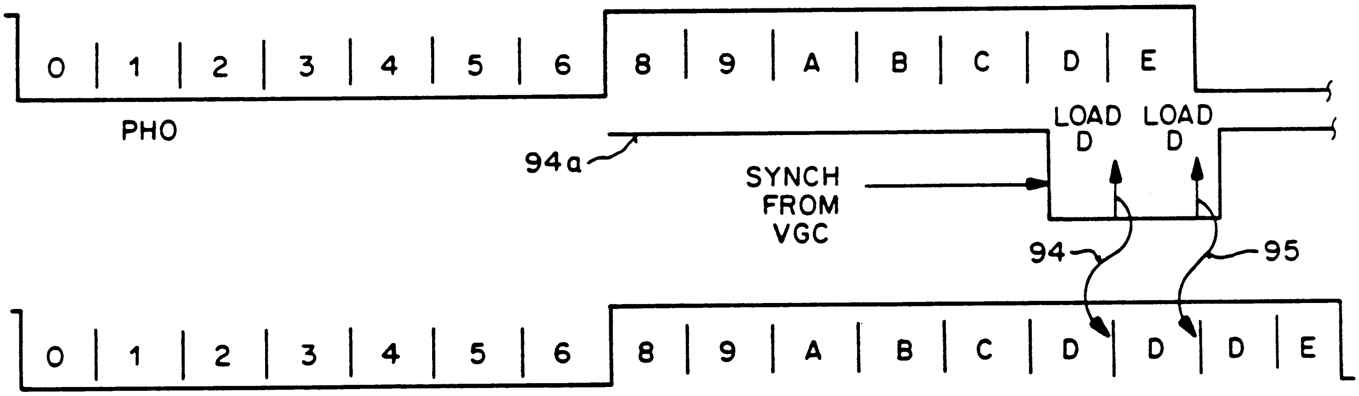

0
1
2
3
4
5
6
8
9
A
B
C
D
E
PHO
94a
SYNCH
FROM
VGC
LOAD
D
LOAD
D
94
95
0
1
2
3
4
5
6
8
9
A
B
C
D
D
D
E

Fig. 7

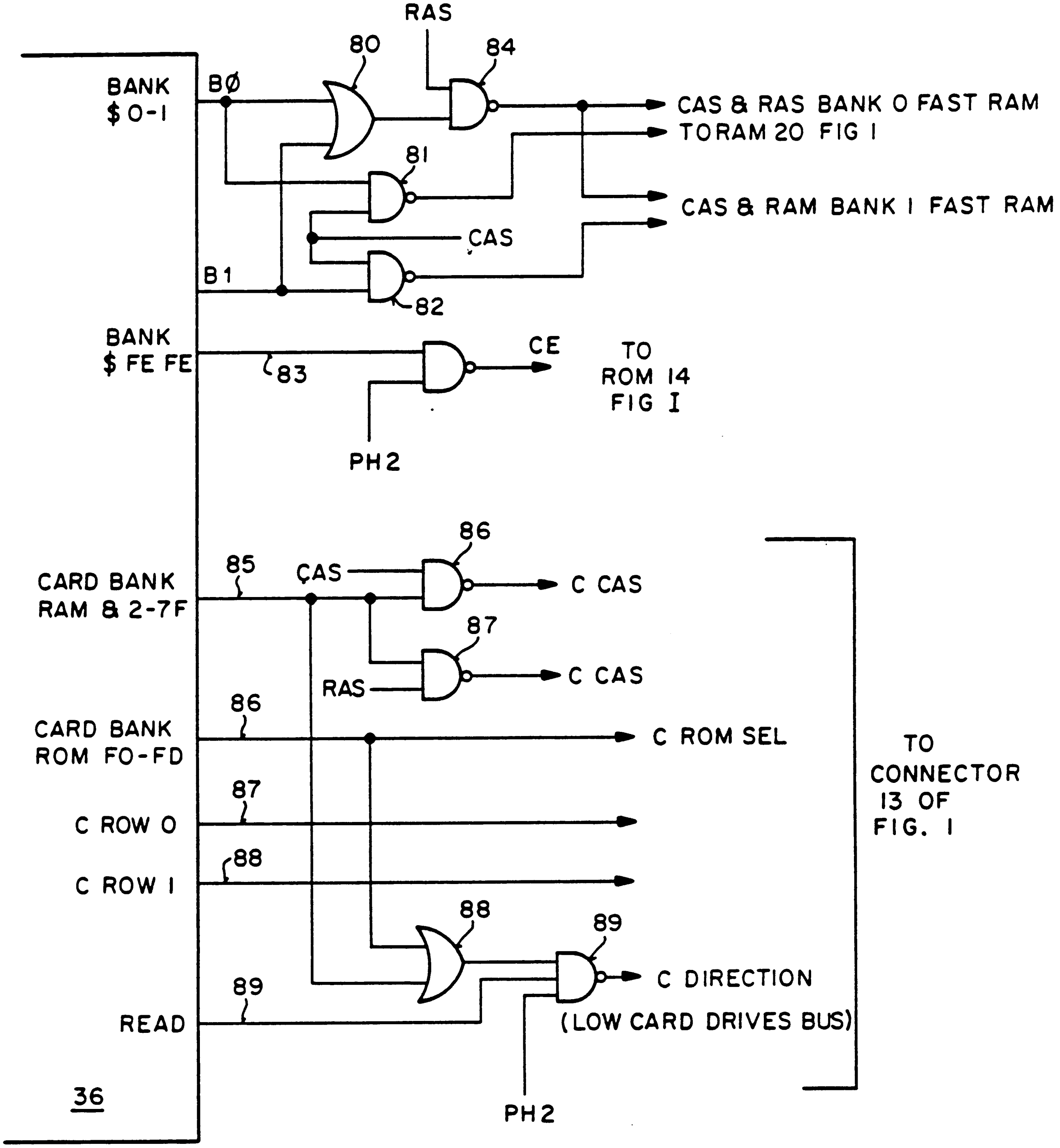

RAS
80
84
BANK
$0-1
BØ
CAS & RAS BANK 0 FAST RAM
TORAM 20 FIG I
81
CAS & RAM BANK I FAST RAM
CAS
B1
82
BANK
$ FE FE
83
CE
TO
ROM 14
FIG I
PH2
86
CARD BANK
RAM & 2-7F
85
CAS
C CAS
87
C CAS
RAS
CARD BANK
ROM FO-FD
86
C ROM SEL
TO
CONNECTOR
13 OF
FIG. I
C ROW 0
87
C ROW I
88
88
89
C DIRECTION
(LOW CARD DRIVES BUS)
READ
89
36
PH2

Fig. 8

BANK MEMORY MAP

Fig. 9

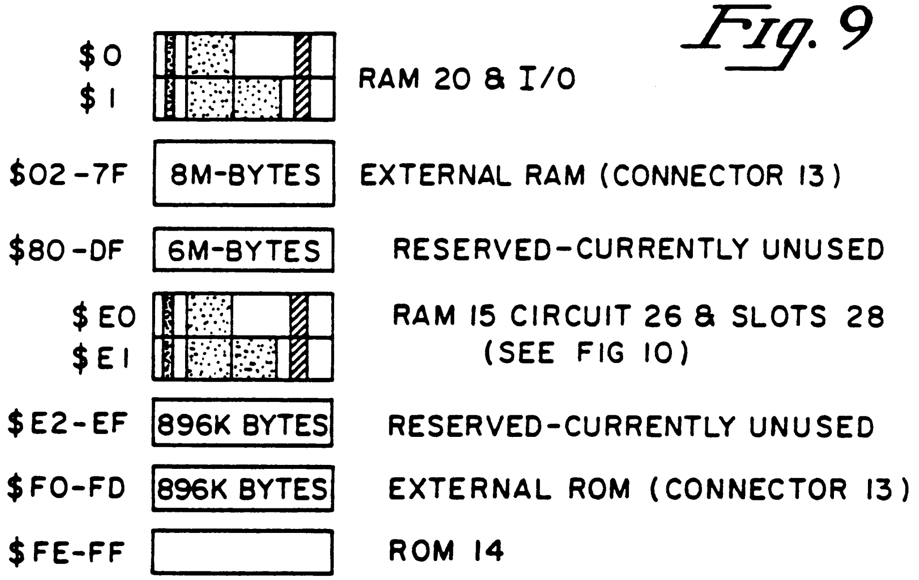

$0
$1
RAM 20 & I/O
$02-7F
8M-BYTES
EXTERNAL RAM (CONNECTOR 13)
$80-DF
6M-BYTES
RESERVED-CURRENTLY UNUSED
$E0
$E1
RAM 15 CIRCUIT 26 & SLOTS 28
(SEE FIG 10)
$E2-EF
896K BYTES
RESERVED-CURRENTLY UNUSED
$F0-FD
896K BYTES
EXTERNAL ROM (CONNECTOR 13)
$FE-FF
ROM 14

Fig. 10

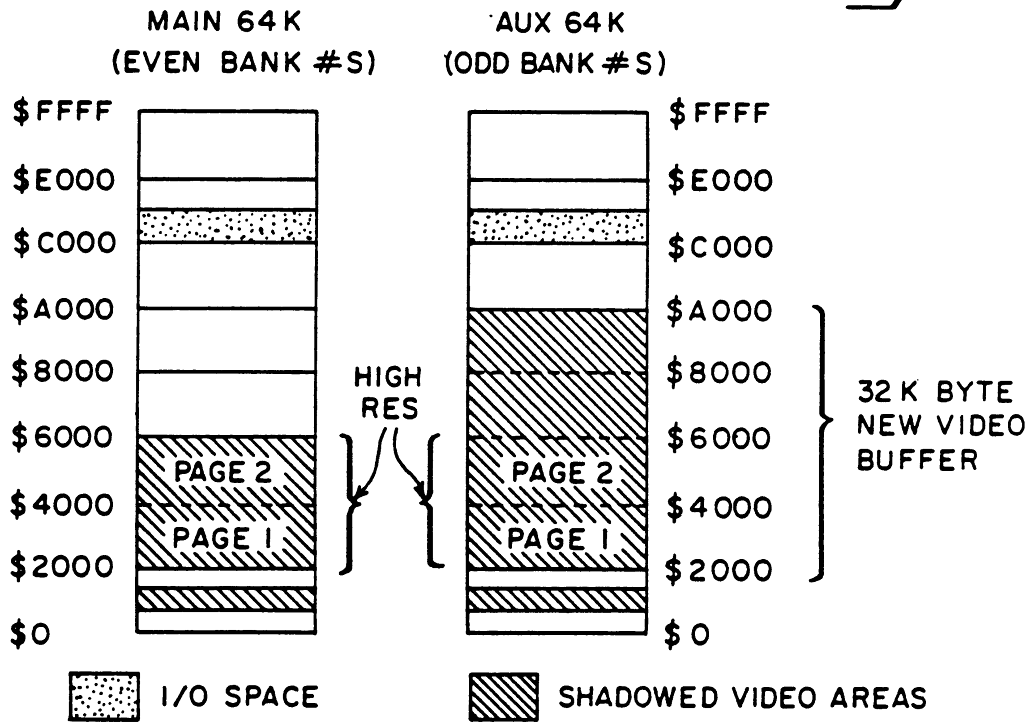


Fig. 11 SHADOW REGISTER

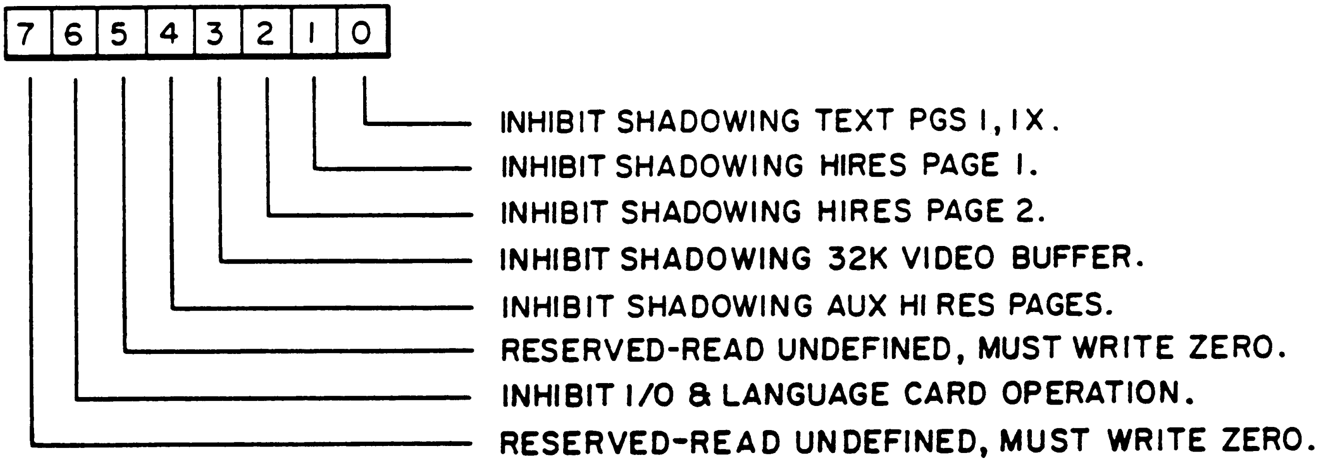

7
6
5
4
3
2
1
0
INHIBIT SHADOWING TEXT PGS 1, 1X.
INHIBIT SHADOWING HIRES PAGE 1.
INHIBIT SHADOWING HIRES PAGE 2.
INHIBIT SHADOWING 32K VIDEO BUFFER.
INHIBIT SHADOWING AUX HI RES PAGES.
RESERVED-READ UNDEFINED, MUST WRITE ZERO.
INHIBIT I/O & LANGUAGE CARD OPERATION.
RESERVED-READ UNDEFINED, MUST WRITE ZERO.

Fig. 12

GENERAL PURPOSE REGISTER

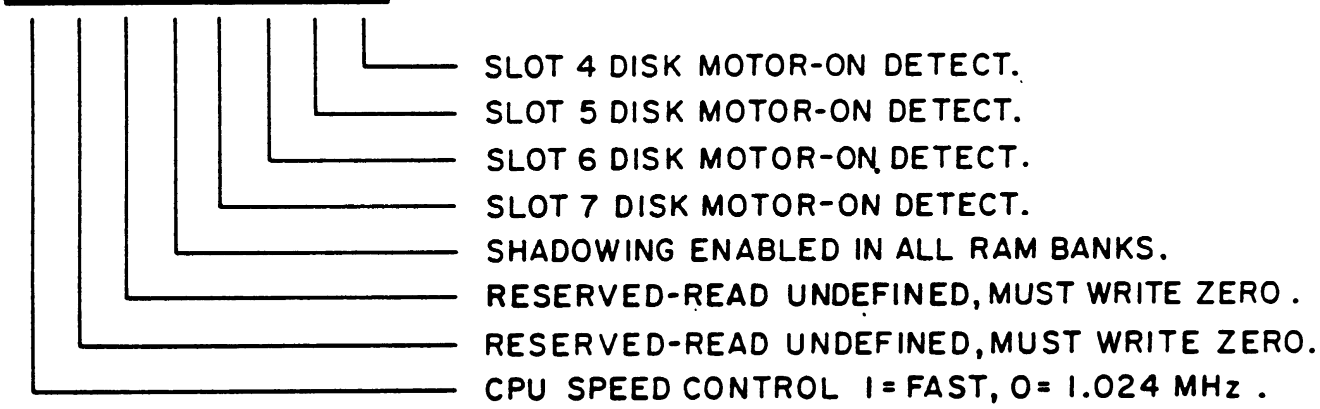

7
6
5
4
3
2
1
0
SLOT 4 DISK MOTOR-ON DETECT.
SLOT 5 DISK MOTOR-ON DETECT.
SLOT 6 DISK MOTOR-ON DETECT.
SLOT 7 DISK MOTOR-ON DETECT.
SHADOWING ENABLED IN ALL RAM BANKS.
RESERVED-READ UNDEFINED, MUST WRITE ZERO.
RESERVED-READ UNDEFINED, MUST WRITE ZERO.
CPU SPEED CONTROL 1 = FAST, 0 = 1.024 MHz.

Fig. 13

SLOT ROM REGISTER

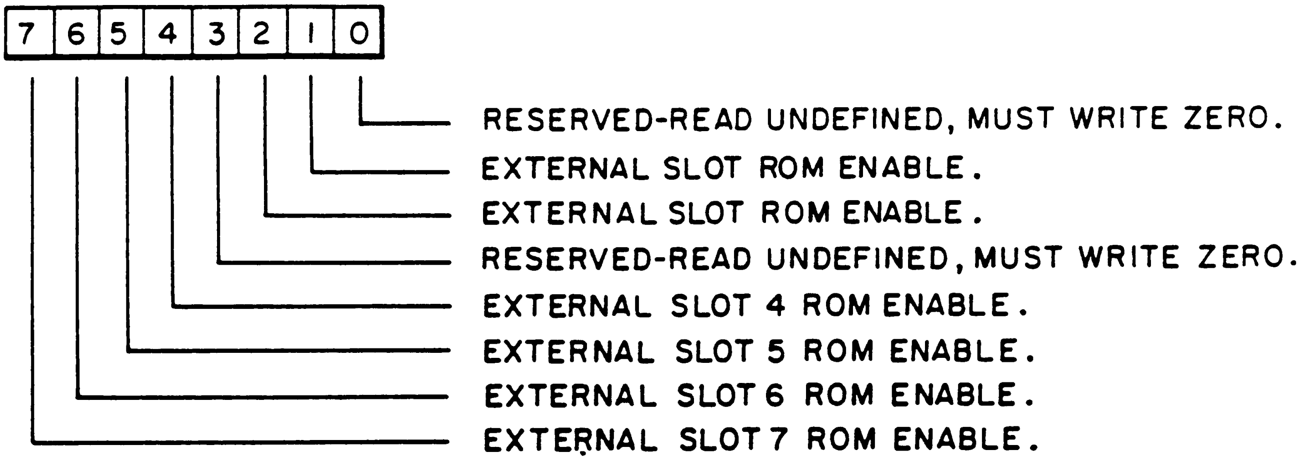
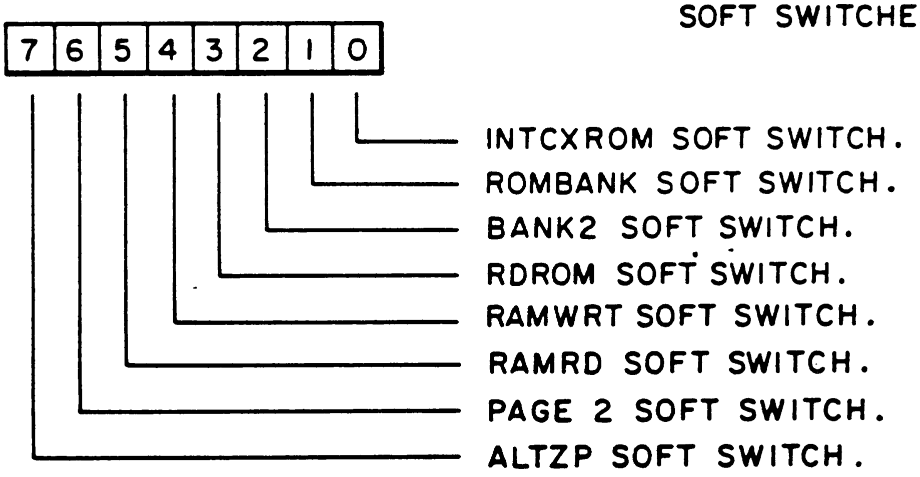

7 6 5 4 3 2 1 0
RESERVED-READ UNDEFINED, MUST WRITE ZERO.
EXTERNAL SLOT ROM ENABLE.
EXTERNAL SLOT ROM ENABLE.
RESERVED-READ UNDEFINED, MUST WRITE ZERO.
EXTERNAL SLOT 4 ROM ENABLE.
EXTERNAL SLOT 5 ROM ENABLE.
EXTERNAL SLOT 6 ROM ENABLE.
EXTERNAL SLOT 7 ROM ENABLE.

Fig. 14

SOFT SWITCHES



COMPUTER WITH INTERFACE FOR FAST AND SLOW MEMORY CIRCUITS

This is a continuation of application Ser. No. 020,599, filed Mar. 2, 1987, now abandoned.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention relates to interface circuits for central processing units of digital computers.

2. Prior Art

More has been written about the Apple II series computers than perhaps any other computer. Beginning in the late 1970's with the introduction of the Apple II computer, followed by the Apple II+, Apple IIe, and Apple IIc, these computers have found wide application in education, science, business and the home. In addition to the voluminous texts, there are literally thousands of commercially available computer programs for the Apple II series computers.

The initial Apple II computer used a central processing unit (the 6502) which operated at a rate of 1 mHz. The computer included a read-only memory (ROM) and a random-access memory (RAM). The RAM stored data for the video display. The 1 mHz timing was used for all ROM and RAM access cycles including accessing by the video circuits for the display. A unique timing mechanism was also used which "stretched" certain timing signals to prevent a phase reversal between the color reference signal and the color video signal (see U.S. Pat. No. 4,136,359). For other aspects of the Apple II computer, see U.S. Pat. Nos. 4,210,959 and 4,278,972.

Since the introduction of the first Apple II computer, substantial progress has been made in semiconductor technology. Microprocessors or central processing units (CPUs) are commercially available which operate at much faster rates with larger data words and addresses.

The present invention deals with the problem of adapting a faster CPU to an Apple II computer. The video timing of the Apple II computer makes it difficult to adapt a faster CPU to the circuitry of the Apple II computer if compatibility with existing programs and certain hardware is to be maintained.

As will be seen, the present invention provides a CPU interface with allows a faster CPU to be "mated with" the slower cycle times associated with the Apple II series computer while still taking advantage of the greater capacity of the faster CPU.

SUMMARY OF THE INVENTION

A computer which provides a video signal for a display and includes a unique interface circuit is described. The central processing unit (CPU) executes a program at the faster cycle time. The CPU communicates with a first random-access memory (RAM) at the slower rate and a second RAM at the faster rate. The first RAM is accessed a the second (slower) rate by video circuits to generate the video signal. The interface circuit includes a decoder which decodes the addresses from the CPU and determines which of the memories is to be accessed. An image of the video data is stored in both the first and second RAMs. When the data is read by the CPU for purposes of updating the display, it is read only from the second RAM (at the faster rate). Since typically many more read cycles of the video data are needed compared with the number of write cycles, substantial time is saved by operating at the faster rate. Moreover, the CPU can execute the program at the faster rate from the second RAM.

Other aspect of the present invention will be apparent from the detailed description.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a block diagram of a computer which includes the present invention; the diagram illustrate the "faster side" and "slower side" of the computer.

FIG. **2** is a general block diagram of the CPU interface circuit.

FIG. **3** is a timing diagram showing the faster cycle and slower cycle timing signals.

FIG. **4** is a diagram used to illustrate different types of faster cycles and slower cycles.

FIG. **5** is a more detailed block diagram of portions of the CPU interface circuit.

FIG. **6** is the portion of the CPU interface circuit which provides synchronized timing with the slower side of the computer for certain operations.

FIG. **7** is a timing diagram used to explain the operation of the circuit of FIG. **6**.

FIG. **8** illustrates some outputs from the CPU interface circuit and logic circuits associated with these outputs.

FIG. **9** shows the bank memory mapping used in the currently preferred embodiment.

FIG. **10** shows the memory space used on the slower side of the computer.

FIG. **11** describes the contents of the shadow register of the interface circuit.

FIG. **12** describes the contents of the general purpose register of the interface circuit.

FIG. **13** describes the contents of the slot ROM register of the interface circuit.

FIG. **14** describes the contents of the soft switches of the interface circuit.

DETAILED DESCRIPTION OF THE INVENTION

A computer with its interface circuit which permits operations at two different rates is described. In the following description, numerous specific details are set forth such as specific cycle times, mapping, bit designations, etc., in order to provide a thorough understanding of the present invention. It will be obvious, however, to one skilled in the art that the present invention may be practiced without these specific details. In other instances, well-known circuits are not set forth in detail so that the present invention is not unnecessarily obscured.

OVERVIEW OF THE PRESENT INVENTION

Referring first to FIG. **1**, the dotted line **50** serves to separate the computer (for purposes of explanation) into a faster side and slower side. That portion of FIG. **1** to the right of and above line **50** is referred to as the slower side of the computer; the portion to the left of and below line **50** as the faster side.

The CPU **12** which in the currently preferred embodiment is a 65C816 operates at a clock rate of 2.8 mHz, approximately 3 times faster than the 6502 initially used in the Apple II computer. The CPU interface circuit **10** is coupled to the CPU **12** and provides control for the faster cycles and slower cycles by varying the timing of the CPU clock signal PH2. The computer includes two RAMs, the first RAM **15** operates at the slower rate and a second RAM **20** operates at a faster rate. While programs may be executed by the CPU from either RAM **15** or **20**, they are generally executed from RAM **20** to take advantage of the RAM's faster cycle times. (RAMs **15** and **20** are both DRAMs fabricated from the same "chips"; thus it is not the inherent rate at which the RAMs can be accessed that determines the slower and faster rate, but rather the rates at which they are accessed under control of the circuit **10**.)

The video data read for generation of the video signal is stored in RAM **15**; this data is read under the control of the slower computer circuits **26** and through the video graphics controller **24** and video section **25** provides the video signal. An image of this video data is also stored in the fast RAM **20**. A "shadow"0 of the data is stored, hence the name shadow cycles is used to identify cycles for writing this shadow into memory. When the CPU **12** is executing a program and needs to read the video data (for example, to update it), the circuit **10** causes the data to be read only from the RAM **20**. When new video data for the display is computed, it is written into RAM **20**, then into RAM **15** in a shadow cycle. Since there are substantially more read cycles than write cycles associated with the video data, substantial time is saved in updating the display in this manner.

PREFERRED EMBODIMENT OF THE COMPUTER

1. Computer Layout

A. Fast Side

Referring to FIG. **1**, the fast side of the computer comprises the CPU **12**, CPU interface circuit **10**, the RAM **20**, read-only memory (ROM) **14**, a connector **13**, and interconnecting bused and lines.

The CPU **12**, as mentioned, is a commercially available microprocessor, the 65C816. The CPU **12** is coupled to the 8 bit data bus **18** and the address buses **33** and **34**. As shown, address bus **34** receives the address signals AO–8, while address bus **34** receives the address signals A8–15. Eight additional "bank" address bits are multiplexed over the data bus. The PH2 (2.8 mHz) clock signal is coupled to the CPU **12** from the interface circuit **10**. This timing signal is generated from oscillator **29** found in the slower side of the computer and is derived from the 14 mHz signal. The read-only memory (ROM) **14** in the presently preferred embodiment has a capacity of 128 K. This ROM stores a monitor program which performs such functions as initialization, etc. These functions are similar to those perfomed by the system monitor (F8) used in the Apple II computer. Other programs, such as "Apple QuickDraw" are also stored in this ROM. The ROM is coupled to the data and address buses. Expansion ROM can be provided at the connector **13** and thus address signals A10–A15 as well as the data bus are shown coupled to the connector **13**. Addresses FRAO-9 and CAS/RAS, discussed later are also coupled to connector **13**.

The CPU interface circuit **10** is described in detail with use of FIGS. **2**, **5** and **8**, and certain timing is described in connection with FIGS. **6** and **7**. In general, as mentioned, it is the interface circuit **10** which determines if a faster or slower cycle is needed, and then controls the computer appropriately. This determination is made by decoding the address signals and examining certain flags ("soft switches"), within the circuit **10**, which are set by signals sent over the data bus.

The fast RAM **20** is fabricated from ordinary 64 K dynamic RAMs. It is organized as shown in four sections of 64 K×4 bits. It is addresses from the circuit **10** by the FRAO-7 address bits (faster RAM address). Accessing of the memory from the bus **18** is controlled in an ordinary manner by the row address strobe (RAS) and column address strobe (CAS) from circuit **10**. (These signals, the chip enable signal to ROM **14** as well as the CAS and RAS signal to the connector **13**, are shown in FIG. **8**.)

The computer of FIG. **1** (faster and slower side) is fabricated on a "motherboard" and includes slots and connectors for receiving external cards. The connector **13** in the currently preferred embodiment is a 44 pin, edge connector for receiving expansion memory (RAM or ROM) for access at the faster rate. (As with the entire block diagram of FIG. **1**, certain control signals, power lines, grounding lines which are well-known and are not needed for an explanation of the present invention are not shown or discussed.)

B. Slower Side

The slower side of the computer is very much like the prior art Apple II computer, except, of course, it does not include a CPU. The slower side of the computer uses CPU **12** which operates at the fast rate. For the most part, there is a one-to-one correlation between many of the circuits on the slower side of the computer and the Apple II computer, except for the video graphics controller **24**. The video graphic controller is described in copending application, Ser. No. 906,753, filed Sept. 12, 1986, entiled ENHANCED VIDEO GRAPHICS CONTROLLER, (and assigned to the assignee of the present invention).

The slower side of the computer communicates with the data bus **18** through bidirectional buffer **22** and the address buses **33** and **34** through the buffer **21**. Both these buffers receive control signals from the CPU interface circuit **10**. The slower side includes a game port **31** for coupling to a joystick. Coupled to the data bus **18** on the slower side is a circuit **42** for providing serial communications (e.g., 26LS30/32). A disk controller **43** permits communications with a disk system. A keyboard microprocessor **44** (e.g., 50740 A) which is coupled to a keyboard, provides scanning as is well- known. The sound section **45** is used for developing audio signals.

The slots **28** are similar to the slots on Apple II computers (excluding the Apple IIc), and comprises seven 50-pin edge connectors for receiving circuit boards. The pin expander **27** provides decoding and timing signals for the slots as is well-known. The slots, of course, communicate with the data and address buses.

The slower computer circuits **26** contain many of the logic circuits found in the current Apple IIe computer. These are control and video circuits all of which are well-known in the art. Additionally, these circuits include means for generation of a RGB signal in a manner described in copending application, Ser. No. 785,220, filed Oct. 7, 1985, entitled METHOD AND APPARATUS FOR GENERATING RGB COLOR SIGNALS FROM COMPOSITE DIGITAL VIDEO SIGNAL, (and assigned to the assignee of the present invention).

The oscillator **29** provides timing signals for both the faster and slower side, and as mentioned, provides a 14 mHz signal to the interface circuit **10**.

The RAM **15** is organized in a similar manner to RAM **20** and is fabricated again from 64 K dynamic memories and organized in four section of 64 K×4. The

RAS and CAS signals are generated within circuits 26. Refresh control is also maintained by circuit 26.

2. FAST CYCLE/SLOW CYCLE TIMING AND TYPES

As mentioned, the RAM 15 of FIG. 1 is accessed at a slower rate so that compatibility is maintained with peripherals and displays of video Apple II series computers. In FIG. 3, the PHO waveform derived from the oscillator 29 output controls the slower cycles for the slower side of the computer. This 1 mHz signal (actual) period 980 nsec.) has two states. During the low state, the video circuitry accesses RAM 15 to provide the video signal; and, during the high state, the CPU has access to the RAM 15.

The faster side of the computer is controlled by the PH2 signal. This approximately 3 mHz signal has a low state of 140 nsec. When addresses are tansmitted by the CPU and a high state of 210 nsec. for data transfer.

Assume now that the CPU 12 needs to access the RAM 15, for instance, to write video data into the RAM 15 during a shadow cycle. The CPU interface circuit 10 determines when this is necessary and then provides a select signal having the waveform 46. When this occurs, the CPU is put in a hold mode and in effect the PH2 signal is then synchronized with the PHO signal. During the next PHO signal, as shown by line 47, data is accepted from the CPU through the buffer 22 and into the slow RAM 15. Thereafter, the CPU continues to operate at the faster rate under control of the PH2 signal. Note that the PH2 signal is not otherwise synchronized with the PHO signal.

FIG. 4 illustrate the type of faster cycles and slower cycles for the computer. The various cycles are all enclosed within ellipse 51. The slower cycles are shown to the right of dotted line 52 and the faster cycles to the left of dotted line 52. On the slower side, the RAM 15 cycles are shown within the ellipse 52. Some of the cycles associated with RAM 15 are for the video display and those are shown within the ellipse 53. A subset of these are identified by ellipse 54 as shadow cycles. These cycles also write data into the display image in RAM 20. Another type of slower cycles are for direct memory access (DMA), and these are shown by ellipse 55. Another category of slower cycles not associated with the memory are the input/output cycles represented by circle 57. These can include, for example, inputs from the game port, keyboard, etc. A subset of these are the slot cycles for the slots 28 of FIG. 1, represented by circle 58.

On the faster side, the fast cycles include fast memory access cycles to the RAM 20 represented by ellipse 60 and fast cycles to the ROM 14 represented by ellipse 61. The ellipse 62 illustrates the faster cycles associated with the interface circuit. For example, there are registers shown in FIG. 5 which are addressable and receive data under control of the circuit 10's decoder. Additionally, these cycles include setting of the soft switches. The cycles associated with the connector 13 are represented by the ellipse 63 and identified as memory card 130 cycles. They include RAM cycles and ROM cycles, since, as mentioned, the expandable memory for connector 13 can include RAM or ROM.

The dotted line connecting ellipses 60 and 54 indicates that there are two memory cycles for shadowed video data, that is, it is written in both RAMs.

3. CPU INTERFACE

First, referring to FIG. 2, the major functional blocks of the CPU interface are illustrated as the address decoder 36, soft switches 37, bus control logic 38, RAM refresh controller 40 and the state machine and timing generation 39.

In general, the address decoder 36 receives the address signals and then decodes then to select one of the cycles of FIG. 4. The specific address range for the bank mapping is shown in FIG. 9. The soft switches 37 are flags which are set through software to indicate certain conditions within the computer. Many of the flags serve the same function as used in the Apple II computer.

The bus control logic 38 controls various signals on the bus and other signal flow which will be more apparent from FIG. 5. The RAM refresh controller 40 performs the well-known function of providing refresh signals for the dynamic RAM 20. This controller is not described since it is not needed to understand the present invention. The state machine and timing generator performs the basic control function for the interface circuit. A portion of the timing circuit pertinent to the present invention is described in conjunction with FIG. 6.

The portion of the CPU interface circuit 10 of FIG. 5 again shows the address decoder 36. The decoder receives the address signals AO-7 (bus 34) and A8-15 (bus 33). The timing signals from FIG. 6 are also applied to the decoder 36. The decoder 36 receives two read/-write control signals during normal operation on lines 73. During DMA operations, these lines function as bidirectional lines and read/write signals are provided by the decoder itself.

As mentioned, 8 additional bits of an address are provided on data bus 18. These address bits are coupled to the address decoder from the bus 18 via lines 74 through the bank address latch and multiplexer 63. The latching of these addresses is controlled by the PH2 clock. During DMA operations, the DMA signal causes the multiplexer 63 to select addresses from the register 64.

There are a plurality of registers coupled to the address bus 18. These registers receive data from the bus 18 under the control of the address decoder 36. The information from these registers is then coupled through the data bus to various portions of the computer.

The register 64 is the DMA bank register, the output of which as already mentioned is coupled through the multiplexer 63 on line 76 to the decoders. The CLK1 signal causes the register to read information from bus 18. The information is returned to the bus 18 via buffer 65 on command of the output enable (OE1) signal.

The shadow register 66 contains data which controls accessing of RAM 15. The specific signals stored in register 66 are shown in FIG. 11. The mapping of FIG. 10 shows the shadowed video areas. The signals in register 66 are returned through buffer 67 to bus 18 on command of the OE2 signal.

The general purpose register 68 stores various signals under control of the CLK3 signal and returns these signals to the bus 18 via buffer 69 on command of the OE3 signal. The signals stored in this register are shown in FIG. 12.

The SLOT ROM register 71 contains signals that are used to determine whether to read data from each I/O slot or its corresponding addresses in ROM. The contents of register 71 are coupled to the bus 18 through the buffer 72 on command of the OE4 signal. The signals contained in this register are shown in FIG. 13.

The buffer **70** is coupled to the soft switches and permits the status of these switches to be read onto bus **18** on command of the OE5 signal. FIG. **14** shows the function of these signals.

In operation, the addresses from the CPU which are coupled to the decoder **36** include addresses which are recognized by the decoder as addresses of the registers **64**, **66**, **68**, and **71**. The specific addresses are set forth in Table 1. The decoder **36** provides the appropriate signal, CLK **1**-N to the register to permit the data to be read from the bus **18** into the register. The contents of these registers is coupled to the decoder **36** for decoding and is used in selecting the appropriate cycle. The decoder also recognizes the read/write signal which determines correct action (e.g., write into or read from the register) to be read through their respective buffer back onto the data bus. This is implemented through the decoder by the OE1 -N signals.

TABLE 1

Address	Contents	Function
$CO2D	Slot ROM Register	Controls Internal/external device selection
$CO35	Shadow Register	Controls which display areas are shadowed
$CO36	General Purpose Register	Controls speed, disk motor detect, and shadow enable in all banks
$CO37	DMA Bank Register	Holds upper 8-bits of DMA address
$CO68	Soft Switches	Map eight switches to an 8-bit R/W Reg.

An output of the decoder selects slower or faster cycle as indicated by line **77**. Another output (line **78**) selects the RAM **20** or ROM **14** as opposed to the extended memory card for connector **13**. The output on line **78** selects the shadow memory cycle. Other outputs of the decoder are described in conjunction with FIG. **8**.

The interface circuit **10** provides control signals for the buffer **22**. The signals consist of a directional signal to indicate which direction data will flow through the buffer and an enable signal. As implemented, data is always enabled through the buffer **22** even if it is not needed on the slower side of the computer, except during DMA operations through the circuit **26**. The capacitance of the bus itself is relied upon to store charge during the latter part of the slower memory cycle when data is being written into the RAM **15** or otherwise transferred into the slower side of the computer. There is, in effect, a "sample and hold effect" on the bus. (Thus, special timing signals are not required to the buffer **22** for the slower memory cycles.). The buffer **21** is also always driven except for DMA operations. Even though these buffers are driven when addresses/data are not being transferred to/from the slower side does not mean the data is "accepted" on the slower side. Enable signals, select signals, address signals prevent "acceptance" of the data.

FIG. **8** illustrates the CAS and RAS control signals from the decoder **36** for the RAM **20** and ROM **14** and the RAM and ROM connected to the connector **13**. Also chip enable signals are shown, one for enabling ROM **14** and another for enabling selection of ROM connected to the connector **13**.

The bank select signals B0 and B1 are coupled through the gates **80**, **81**, **82** and **84** along with a CAS and RAS signal to provide CAS and RAS signals for bank **0** and bank **1** of RAM **20** of FIG. **1**. The bank select signal on line **83** is gated by the PH2 signal to provide the chip enable signal for selection of the ROM **14**. The remaining signals on lines **85**, **86**, **87**, **88** and **89** are all coupled through gates **86**, **87**, **88** and **89** as shown to the connector **13** of FIG. **1**. They provide the CAS and RAS signal for external RAM, the directional control signal for the RAM and the other signals as shown.

4. SYNCHRONIZATION BETWEEN FASTER CYCLES AND SLOWER CYCLES

Referring again to FIG. **3** when access is needed to the slower side of the computer, the CPU waits until the appropriate time, based on the PH2 signal to access the slower sides of the computer as indicated by waveforms **46** and **47**. For this reason, it is necessary for the interface circuit to keep track of the PHO signal.

There is an added dimension to keeping track of the PHO signal because of the unusual timing used in the Apple II series computers. Periodically, the PHO clock is "stretched" to provide additional counts as described in U.S. Pat. No. 4,136,359. Therefore, it is necessary for the interface circuit to keep track of these stretch cycles. The circuit for doing this is shown in FIG. **6**.

The interface circuit receives a synchronization signal from the video graphics controller on line **94**. The waveform for this signal is shown in FIG. **7** on line **94***a*. This signal indicates the stretched PHO signal.

The circuit of FIG. **7** includes a 4 bit counter **92** which is clocked by the 14 mHz signal. The 4 bit count at the output of this counter is connected to the state machine and logic circuits of interface circuit **10**, and as will be seen, the most significant bit of this signal is in fact the PHO signal. This decoder **93** examines the 4 bits from counters and determines when a $6 count or $E count is present at the output of counter **92**. When a $6 count is present, a signal is coupled to the preload logic **91** causing $8 to be loaded into the counter **92**. Similarly, when decoder **93** detects a $E at counter **92**, it couples a signal to the logic **91** causing $0 to be loaded into counter **92**. The signal on line **94** causes a $D to be loaded into the counter **92**.

Referring to FIG. **7**, the uppermost waveform represents the PHO signal. Initially assume the counter **92** has all 0's and the count proceeds as shown from $0-6. When the count $6 is reached, the decoder **93** and logic **91** cause $ 8 to be loaded into counter **92** a the counter continues counting from $ 8, that is hexadecimal **9**, A, B C, D and E. When $E is reached, the decoder and logic **91** cause $ 0 to be loaded into counter **92** and the waveform shown on the upper line of FIG. **7** is repeated. As is apparent, the most significant bit of the counter is in fact the PHO signal.

The stretched cycles cause the PHO signal to be extended by two cycles of the 14 mHz clock. As shown in FIG. 7, when the counter contains $D, the synchronization signal is received on line **94**, indicating the stretched cycle. $D is loaded into the counter **92** for two cycles of the 14 mHz clock, and then the count proceeds to the final $E. The count within the counter **92** for these stretched cycles is shown on the lower waveform of FIG. **7** with the interaction between the synchronization signal from the video graphics controller and the logic **91** being shown by the lines **94** and **95**.

Thus, a computer has been described which includes two RAMs, one of which is operated at a faster rate than the other. The interface circuit provides control for the faster and slower cycles in a manner which allows the CPU to operate a substantial portion of its time at the faster rate.

I claim:

1. A computer which provides a video signal for a display comprising:

a central processing unit (CPU) which executes a program to provide said video signal for said display;

first and second random-access memories (RAMs) couples to said CPU, both of said memories storing video data, and said CPU accessing said first RAM at a first rate and said second RAM at a second rate, said second rate being faster than said first rate;

video circuits coupled to said first and second RAM, and to said display for generating said video signal from said video data stored in said first RAM for said display, said circuits accessing said first RAM at said first rate, said video data being updated and stored in both said first RAM and said second RAM;

an interface means for providing control between said CPU and said first and second RAMs such that when said CPU is executing said program and needs to read said video data, said interface means causes said video data to read only said second RAM by said CPU thereby allowing said CPU to operate a substantial portion of its time at said second rate.

2. The computer defined by claim 1 including at least one first connector and at least one second connector, said first and second connectors being coupled to said CPU, said interface means for permitting data to be accessed at said first connector at said first rate and at said second connector at said second rate.

3. The computer defined by claim 2 including a read-only memory (ROM) coupled to said CPU and said interface means, said interface means causing said ROM to be accessed by said CPU at said second rate.

4. The computer defined by claim 3 wherin said interface means includes a decoder which decodes addresses from said CPU and from said addresses provides said control at one of said first and second rates.

5. The computer defined in claims 1 or 4 wherein said computer includes a data bus and an address bus, said CPU providing certain address signals on said data bus during predetermined periods.

6. The computer defined by claim 5 wherein said interface means includes a plurality of registers coupled to said data bus for receiving said certain addresses.

7. A computer which provides a video signal for a display comprising:

a central processing unit (CPU); a data bus coupled to said CPU;

an address bus coupled to said CPU;

a first random-access memory (RAM) coupled to said data bus and said address bus;

a second RAM coupled to said data bus said address bus, said first and second RAMs storing video data, said video data written by said CPU into said second RAM and then into said first RAM during a shadow cycle, said CPU accessing said first RAM at a first rate and said second RAM at a second rate, said second rate being faster than said first rate;

video circuits coupled to said first and second RAM, and to said display for generating said video signal from said video data stored in said first RAM for said display, said circuit accessing said first RAM at said first rate, said video data being updated and stored in both said first RAM and said second RAM;

an interface means coupled to said data bus, said address bus, said first RAM and said second RAM, for controlling first memory cycles between said CPU and said first RAM at said first and second memory cycles between said CPU and said second RAM at said second rate;

said interface means including decoding means for decoding addresses from said CPU to select between said first and said second memory cycles such that when said CPU is executing a program and needs to update said video data, said interface means causes sia video data to be read only from said second RAM by said CPU thereby allowing said CPU to operate a substantial portion of its time at said second rate;

said interface means also including timing means for synchronizing cetain memory cells with said first RAM.

8. The computer defined by claim 7 wherein said first and second RAM are fabricated from the same dynamic access memory ports.

9. The computer defined by claim 7 wherein said video circuits include a video graphics controller which provides a timing signal to said timing means of said interface means.

10. The computer defined by claim 9 wherein said video circuits periodically have an extended cycle and wherein the occurrence of said extended cycle triggers said timing signal.

11. A computer which provides a video signal for a display comprising:

a central processing unit (CPU) which executes a program;

a data bus coupled to said CPU;

an address bus coupled to said CPU;

a first random-access memory (RAM) coupled to said data bus and said address bus;

a second RAM coupled to said data bus and said address bus, said first and said second RAMs both storing the same video data, said video data being written by said CPU into said second RAM and then into said first RAM during a shadow cycle, said CPU accessing first RAM at a first rate and said second RAM at a second rate, said second rate being faster than said first rate;

video circuits coupled to said first and second RAM, and to said display for generating said video signal from said video data stored in said first RAM for said display, said circuits accessing said first RAM at said first rate which is compatible with the video timing requirements of said computer, said video data being updated and stored in both said first RAM and said second RAM;

an interface means coupled to said data bus, said address bus, said first RAM and said second RAM, for controlling first memory cycles between said CPU and said first RAM at said first rate, and second memory cycles between said CPU and said second RAM at a second rate, said first rate being slower than said second rate;

said interface means including decoding means for decoding addresses from said CPU to select between said first and second memory cycles, and registers coupled to said decoding means and said bus for receiving certain address signals from said data bus during predetermined periods such that when said CPU is executing said program and needs to update said video data, said interface means causes said video data to be read only from said second RAM by said CPU thereby saving substantial time during updating of said display and allowing said CPU to operate a substantial portion of its time at said second rate, updated video data being written into said first and second RAMs by said CPU

12. The computer defined by claim **11** wherein a first of said registers receives signals representing locations in said first RAM into which digital signals representing a video display are written.

13. The computer defined by claim **11** wherein a second of said registers receives part of a direct memory access address.

14. The computer defined by claim **11** wherein said computer includes a plurality of slots for receiving additional circuits and a third of said registers receives signals directing access to said slots.

15. A computer comprising a central processing unit CPU);

a first random-access memory (RAM) coupled to said CPU;

a second random-access memory (RAM) coupled to said CPU, said first and said second RAMs both storing the same video data, second RAM being accessed at a faster rate as compared to said second RAM;

video circuits for providing video signals for a video display, said video signals being developed from said video data stored in said second RAM, said second RAM being coupled to said video circuits;

CPU interface means for providing control between said CPU and said first and second RAMs, said interface means providing a control signal to said CPU and said first and second RAMs to selectively control access of said CPU to said RAMs such that when said control signal is in one state said video circuitry reads said second RAM to provide said video signal to said display, and when said control signal is in another state said CPU reads said first RAM to update said video data, said CPU never reading said video data from said second RAM during the updating of said video data;

said interface means also writing said updated video data into both of said first and said second RAMs simultaneously.

* * * * *

United States Patent [19]

Gruenberg et al.

[11] **Patent Number: 4,959,887**

[45] **Date of Patent: Oct. 2, 1990**

[54] **CENTER PIVOT COVER**

[75] Inventors: **Eric I. Gruenberg,** Soquel; **James J. Halicho,** Sunnyvale; **Melvin J. Phillips,** Cupertino; **Troy K. Hulick,** San Jose, all of Calif.

[73] Assignee: **Apple Computer, Inc.,** Cupertino, Calif.

[21] Appl. No.: **352,777**

[22] Filed: **May 16, 1989**

[51] **Int. Cl.**^{5} **G06F 1/00**
[52] **U.S. Cl.** **16/223;** 364/708
[58] **Field of Search** .... 364/708; 16/376, 377, 16/374, 355, 356, DIG. 13, 223

[56] **References Cited**

U.S. PATENT DOCUMENTS

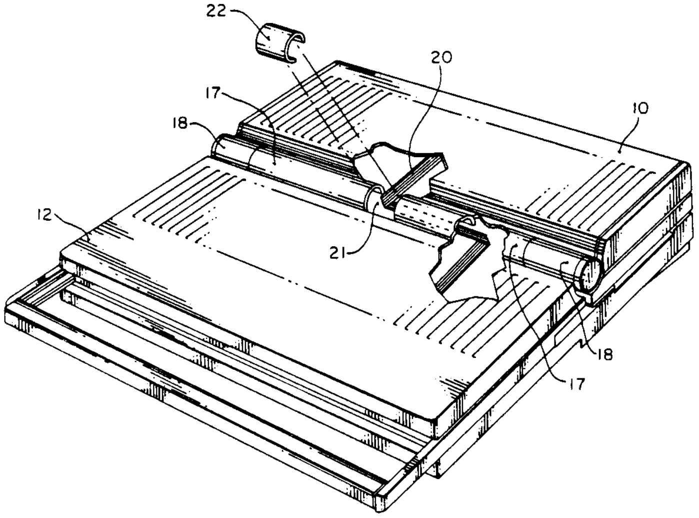
2,811,741	11/1957	Miller et al.	16/376
4,434,525	3/1984	Labelle	16/374
4,742,478	5/1988	Nigro, Jr. et al.	364/708
4,825,395	4/1989	Kinser, Jr. et al.	364/708

Primary Examiner—Richard K. Seidel
Assistant Examiner—James Miner
Attorney, Agent, or Firm—Blakely, Sokoloff, Taylor & Zafman

[57] **ABSTRACT**

An improved hinge assembly for a personal computer system. The hinge is comprised of a tubular portion which is rigidly connected to the cover unit of the computer. A part of the tubular hinge is left open to form a passageway into the cover unit. The tubular portion rotates relative to the main body section of the computer. The tubular portion has a longitudinal section removed from near its mid-point, forming a gap. This gap is closed by a curved cover element. The cover element is held stationary with respect to the main body section of the computer. Thus, when the cover unit is opened, the tubular portion rotates relative to the cover element. A cable connecting the electrical components in the cover unit to the main body section passes through an opening in the curved cover element, is bent, travels through the tubular section, parallel to its central axis, is bent again and passes into the cover unit.

10 Claims, 4 Drawing Sheets



Macintosh Portable

FIG 1

(PRIOR ART)

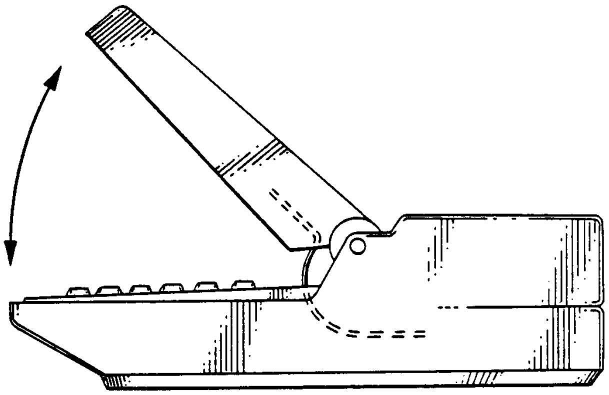


FIG 2

(PRIOR ART)

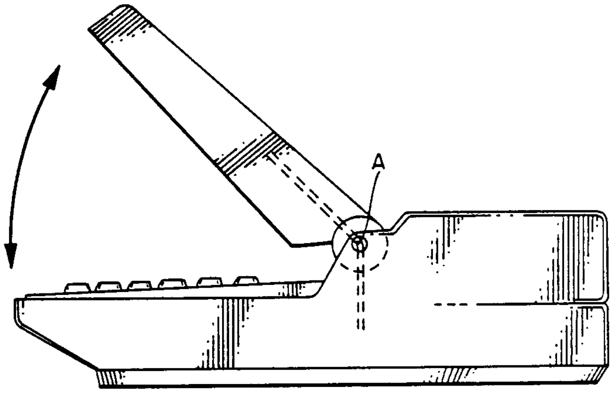
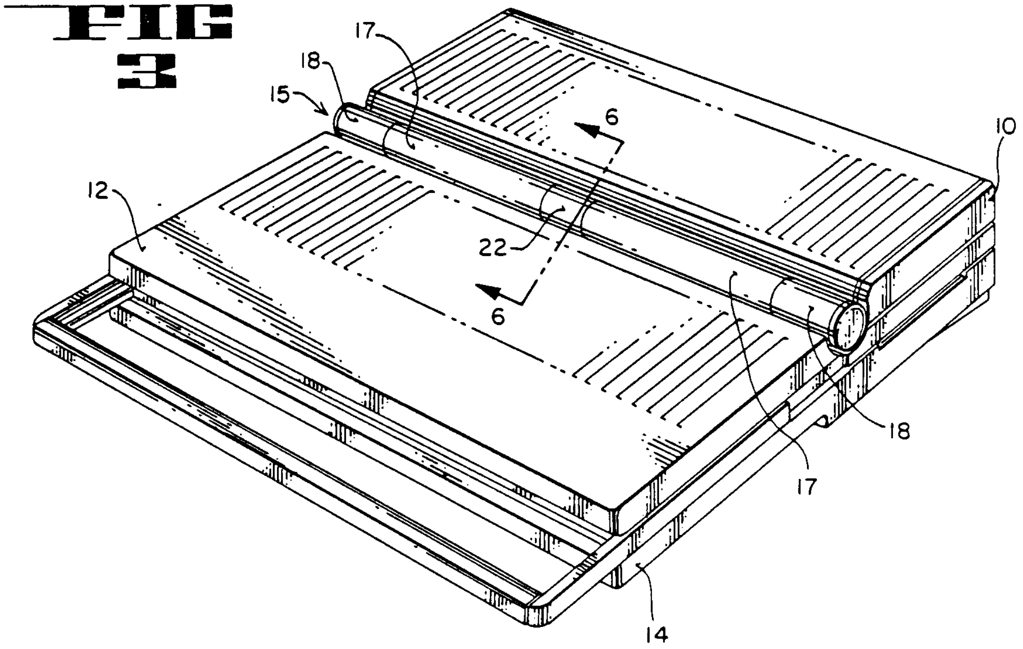


FIG 3



U.S. Patent Oct. 2, 1990 Sheet 2 of 4 4,959,887

FIG 4

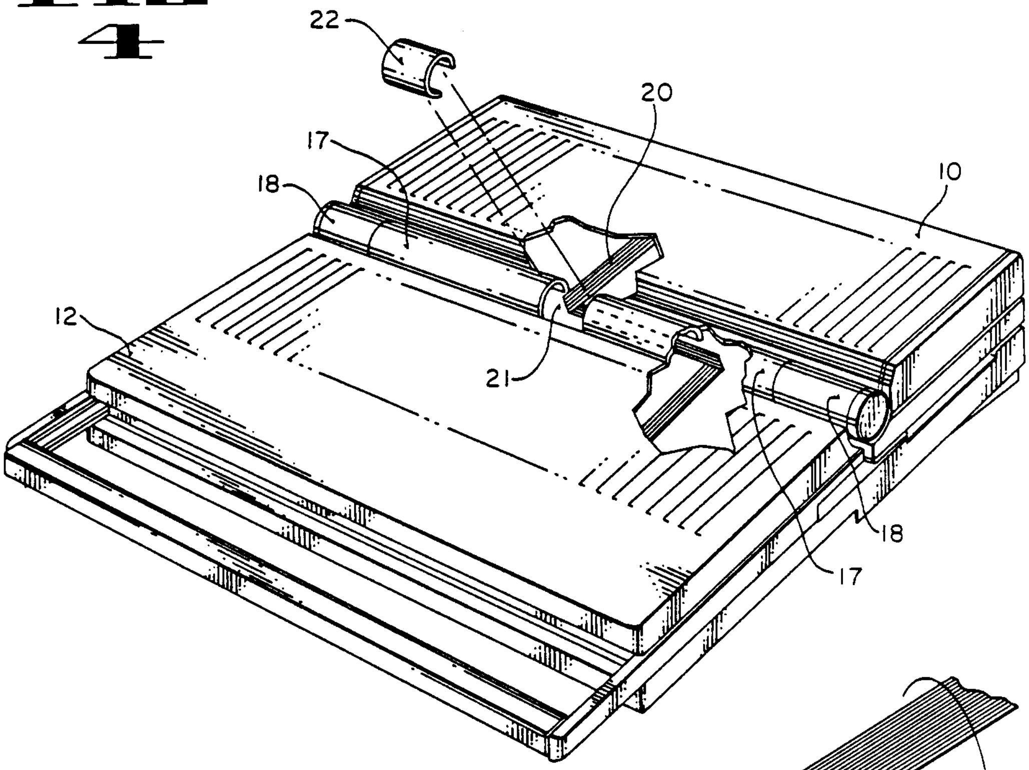
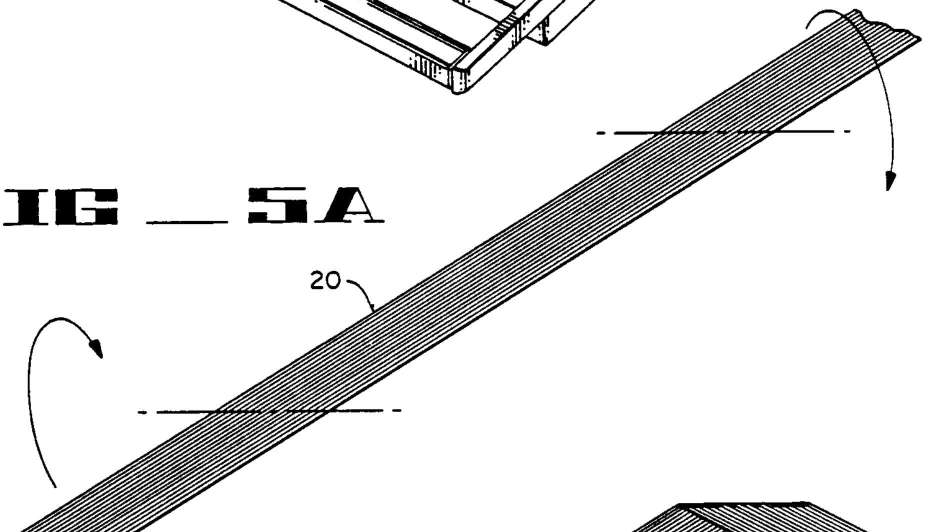


FIG _ 5A



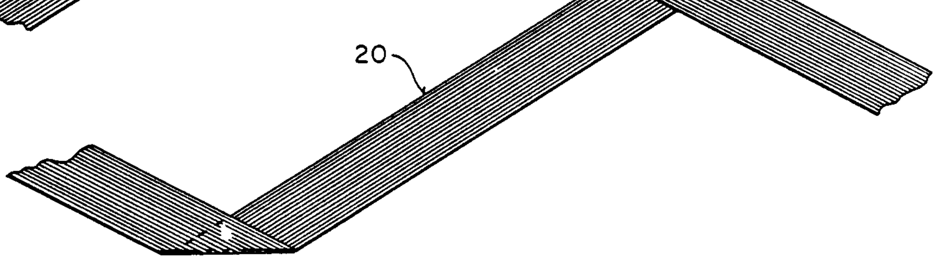


FIG _ 5B

FIG_6

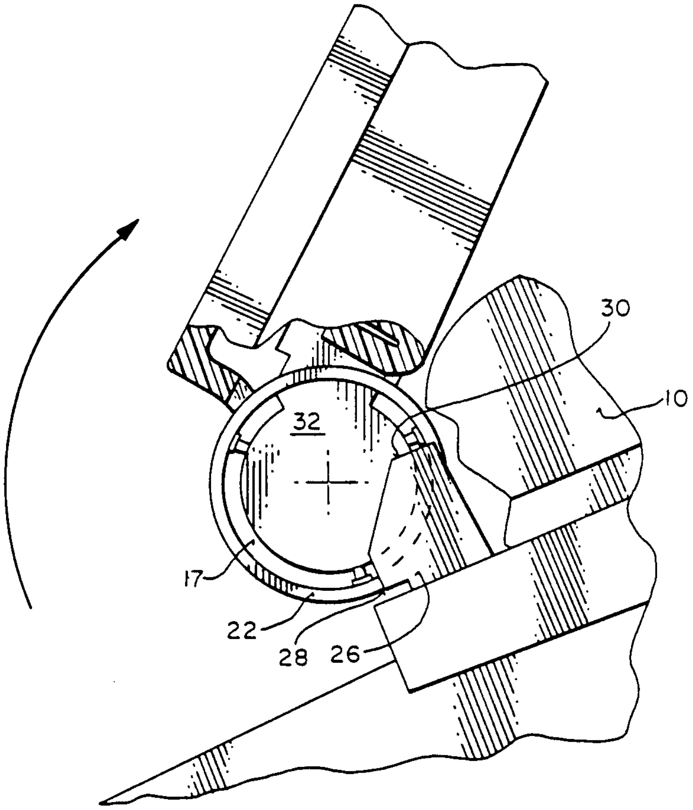


FIG_7

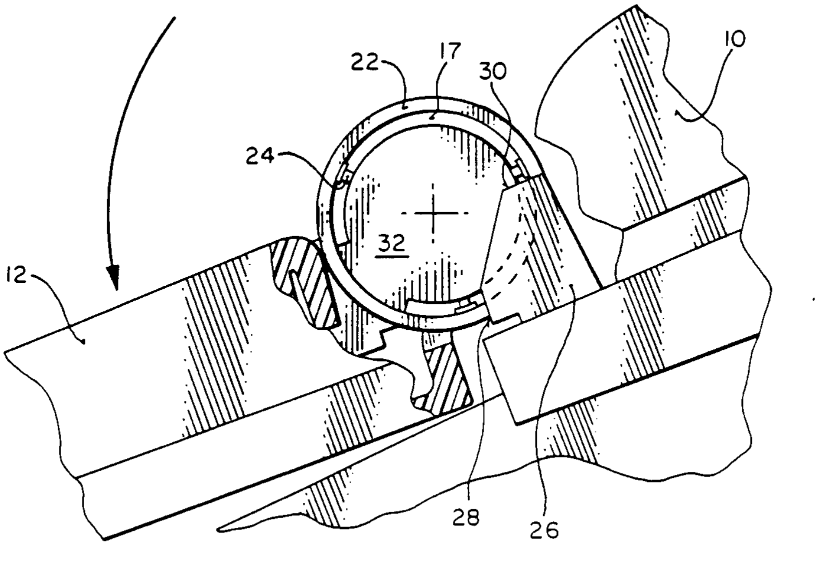


FIG__8

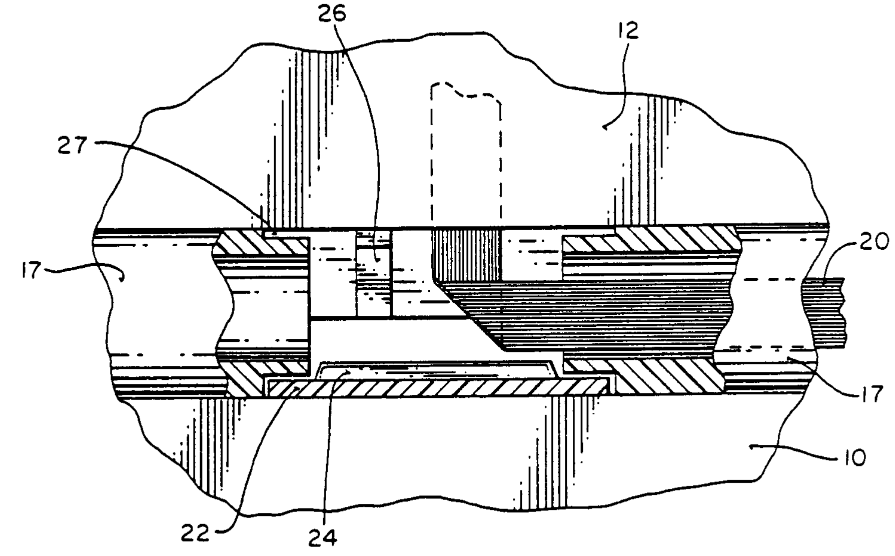


FIG__9

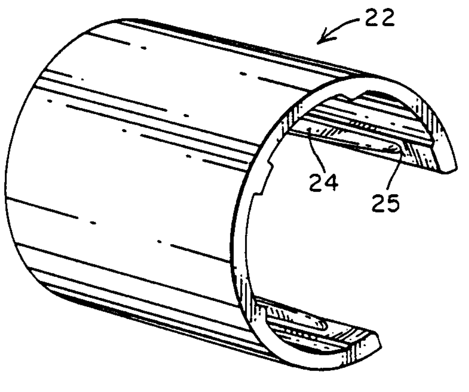


FIG__10

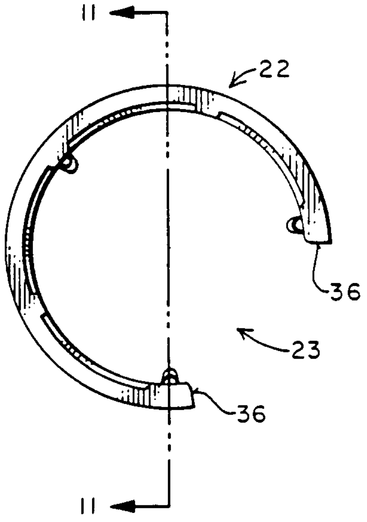
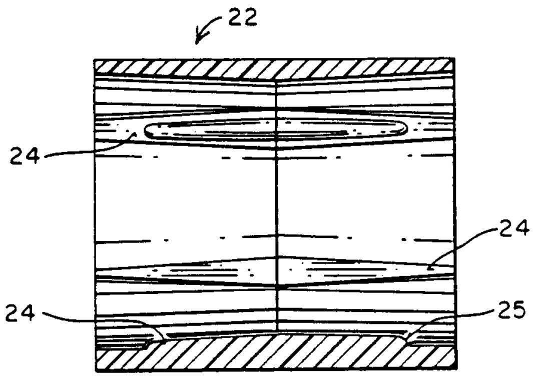


FIG__11



CENTER PIVOT COVER

BACKGROUND OF THE INVENTION

1. Field of the Invention:

The present invention relates to the field of hinge and connection devices, and more particularly to an improved hinge and cable assembly for a portable personal computer.

2. Art Background

In the field of personal computers, computers which are portable and may easily be carried from place to place are commonly referred to as "lap-top" computers. Typically lap-top computers are compact in size and relatively light in weight. Because of their portable nature, lap-top computers are usually configured differently then most other personal computer systems. A lap-top computer is usually a single, integrated, unit. All of the elements of the computer are placed within one housing. This is in contrast to most personal computer systems where the elements of the system, such as the keyboard, video display, and Central Processing Unit (CPU) are physically distinct entities. The integrated nature of lap-top computers is desirable because it enhances their portability.

Most lap-top computers have a main body section which contains all of the essential circuitry of the computer such as the CPU, the power supply, and data storage devices, such as a floppy or hard disk. Attached to the front of the main body section is a keyboard unit which allows a user to communicate with the computer. There is a top cover unit which is placed over the keyboard unit. The cover unit is connected to the main body section by a hinge. The hinge allows the cover to be opened upwards, revealing the keyboard.

The top cover unit performs several functions. First, when closed, it covers the keyboard, thus protecting the keys while the computer is being transported. The cover also usually holds the display unit of the lap-top. When the cover is lifted upwards, the display unit is visible to the user. In this manner, the display unit is also protected. The display unit can be a video monitor, a liquid crystal display, or any other equivalent device. Often, when the cover is rotated upwards the computer is turned on.

Because the computer's video display is located in the cover unit, there must be some method of communicating between the video display and the electronic circuitry in the main body section. The video display must be provided with a power supply as well as control signals for the actual display device. Typically, this communication is accomplished through the use of an electrical cable.

Several methods of routing the cable between the main body section and the cover unit are known in the prior art. However all of these previously disclosed arrangements have some form of an inherent limitation. In one method, for example, the cable simply exits the housing at some location near the hinge and then reenters the cover some distance away. This method is illustrated in FIG. 1. This method is obviously undesirable because it exposes the cable to the outside environment. Such exposure could easily lead to damage in the cable or even a potential electrical shock to a user of the computer.

In a second method known in the prior art, the cable is simply routed through the hinge in the manner as shown in FIG. 2. In this manner, the cable is covered. However, this method is also undesirable for several reasons. First, when the cover is opened and closed, the cable bends at point A. This repeated bending can lead to a work-hardening of the metal conductors within the cable. Eventually, the conductors can become brittle and break which will lead to a failure in the cable. Another drawback with this method is that the space provided for the cable within the hinge is very limited. As a result, the cable must be passed through the hinge before the connectors which are coupled to the ends of the cable are attached. Attaching the connectors while the cable is within the computer is a difficult task. As such, the time needed to manufacture the computer and its associated costs are both increased.

SUMMARY OF THE INVENTION

The present invention overcomes the limitations of the prior art by providing an improved hinge assembly for a portable personal computer. With the present invention the hinge is comprised of a tubular portion which is rigidly connected to the cover unit of the computer. A part of the tubular hinge is left open to form a passageway into the cover unit. The tubular portion rotates relative to the main body section of the computer. The tubular portion has a longitudinal section removed from near its mid-point, forming a gap. This gap is closed by a curved cover element. The cover element is held stationary with respect to the main body section of the computer. Thus, when the cover unit is opened, the tubular portion rotates relative to the cover element. A cable connecting the electrical components in the cover unit to the main body section passes through an opening in the curved cover element, is bent, travels through the tubular section, parallel to its central axis, is bent again and passes into the cover unit.

SUMMARY OF THE DRAWINGS

FIG. **1** is an illustration of a cable routing arrangement that was used in the prior art.

FIG. **2** is an illustration of an alternative cable routing arrangement that was also used in the prior art.

FIG. **3** shows a portable personal computer system that employs the center pivot cover of the present invention.

FIG. **4** is a view similar to FIG. **3** except that portions of the casing of the computer system have been cut away and the cover section is shown in exploded format.

FIGS. **5*a*** and **5*b*** illustrate the preferred embodiment of the cable which is used in the present invention.

FIG. **6** is a cross-sectional view of the hinge with the cover in the open position taken along the line **6—6** in FIG. **3**.

FIG. **7** is a cross-sectional view of the hinge with the cover in the closed position taken along the line **6-6** in FIG. **3**.

FIG. **8** is a top detail view of the hinge showing how the cover section is coupled to the tubular portions.

FIG. **9** illustrates the cover section as used in the present invention.

FIG. **10** is a side view of the cover section.

FIG. **11** is a cross sectional view of the cover section taken along the line **11—11** in FIG. **10**.

DETAILED DESCRIPTION OF THE INVENTION

A center pivot cover for use in a portable personal computer system is described. Throughout the following specification, various details such as specific component shapes and arrangements, are set forth in order to provide a more complete description of the present invention. In other instances well known elements and methods of manufacture are not described in detail so as not to obscure the present invention unnecessarily. Moreover, throughout the following specification, the present invention is described with reference to to use in a portable personal computer system. It will be apparent to those skilled in the art, however, that the center pivot cover and hinge arrangement can be adopted for use in any electronic system that contains two or more elements which must be movably connected together.

Referring first to FIG. **3**, a perspective view of a portable personal computer system using the center pivot cover of the present invention is shown. The computer system consists of the main body section **10** and a cover unit **12**. In FIG. **3** the cover unit is shown in a closed position. However, the cover unit rotates upward to an open position. In the open position the keyboard and the video display of the computer system are exposed. The keyboard is contained in the keyboard section **14** which is located underneath the cover unit **12**.

The cover unit **12** is connected to the main body section **10** by means of a hinge **15**. The hinge is cylindrical in shape with a substantially circular cross-section. In the preferred embodiment, it extends substantially across the entire width of the computer system. It will be apparent to those skilled in the art, however, that the hinge is not required to extend across the entire width of the computer system and that the same results can be achieved with a hinge that covers only part of the computer system. The hinge comprises two tubular portions **17**, a curved cover element **22** and two clutch units **18**. The tubular portions **17** are rigidly connected to the cover section **12**. The clutch units **18** are disposed at opposite ends of the hinge **15** and are rigidly connected to the main body section **10**. The hinge is connected to each of the clutch units and rotates with respect to the clutch units **18**. The clutch units contain a mechanism which hold the cover unit **12** in place after it has been opened. The term "center pivot cover" refers to the hinge and cover element arrangement that is used in the present invention.

Referring next to FIG. **4**, a second perspective view of the computer system is shown. In this illustration, portions of the casing of the computer system have been cut away so as to more clearly illustrate the interior details of the computer and the relationship of the curved cover element **22** to the various other elements of the computer system. In FIG. **4**, the curved cover element **22** removed from the computer system with the phantom lines indicating how the cover section is coupled to the hinge. As can be seen, a longitudinal gap **21** is present between the two tubular sections **17**. This gap is covered and closed by the cover section **22**. FIG. **4** also illustrates how the cable **20** passes through the hinge **15** and into the cover unit **12**.

FIG. **8** illustrates a top detail view of the computer system showing how the cover element **22** is coupled to the tubular portions **17** and the hinge **15**. FIG. **4** also illustrates how the cover element **22** is coupled to the tubular portions. The cover element is simply snapped into place to cover the gap **21** between in the tubular portions **17** of the hinge **15**. The cover piece is not fixed to the tubular portions **17**, but is free to move relative thereto. In FIG. **8**, various sections of the tubular portions **17** and the curved cover element **22** have been cut away in order to more clearly show the elements of the present invention. As can be seen, the ends of the tubular portions **17** are of a smaller diameter than the main body of the tubular portions. This reduction in diameter provides a recessed area **27** into which the cover section **22** is placed. The recessed area **27** works together with the stiffeners **24** located on the inner surface of the cover section **22** to properly locate the cover section **22**. When the cover section is in place, it entirely covers the gap **21** between the tubular portions **17**. As described in more detail below, the opening **23** in the cover section is held fixed so as not to be visible by a user of the computer system. Also, in the preferred embodiment, the cover element has a diameter which is substantially equal to the diameter cf the tubular portions. In this manner, it appears as though the hinge is one seamless unit to the user of the computer system.

FIGS. **9** through **11** illustrate the curved cover element **22** which is used in the present invention to close the longitudinal gap **21** between the tubular portions **17** of the hinge **15**. In crosssection the curved cover piece has a shape which comprises a portion of the arc of a circle, An opening **23** is left between the ends **36** of the cover. The cable **20** passes through this opening. The exterior surface of the preferred embodiment of the cover is relatively smooth. The interior surface has located on it a plurality of stiffeners **24** to help maintain the shape of the cover piece **22**. The stiffeners **24** are simply raised portions that are formed integrally with the cover piece. The stiffeners **24** are disposed longitudinally along the interior surface and are substantially parallel to the central axis of the hinge **15**. The stiffeners **24** do not extend all the way to the edge of the cover piece. Instead there is a gap between the end of the stiffeners **24** and the edge of the cover piece **22**. This gap forms a shoulder portion **25**. When the cover section is coupled to the tubular portions **17** of the hinge, the shoulder acts as a stop to keep the cover piece properly located in the longitudinal direction. In the preferred embodiment, the cover section is manufactured from injection molded plastic.

When the computer is in its assembled form, the cover element **22** is held in a fixed orientation with respect to the main body section **10** of the computer. This is best illustrated with reference to FIGS. **6** and **7**. In FIG. **7**, the cover unit is closed. In FIG. **6** it is open. The curved cover element **22** is prevented from moving by physical stop member **26**. This element is an extended arm which projects upwardly from the base of the main body section **10**. In the preferred embodiment, the physical stop member is manufactured from injection molded plastic. The physical stop member has a vertical surface **28**, and a horizontal surface **30** which contacts the edges of the curved cover element **22**. This prevents the cover element from moving when the cover unit **12** is opened and closed. Thus, regardless of the position of the cover unit **12**, the curved cover element is always oriented so that the opening **23** is located inside of the computer system and is not visible to the user.

Referring again to FIG. **8**, the relationship between the cover element **21** and the physical stop member **26** is further illustrated. As can be seen, the physical stop member 26 is relatively narrow and does not take up the entire width of the cover section. Instead, the cable 20 is inserted into the hinge at this point. The manner in which the cable is routed into the hinge is described in more detail below.

Referring again to FIGS. 6 and 7, two cross-sectional views of the hinge 15 and cover unit 12 are shown. As can be seen, the tubular portion 17 is not a complete circle. Where it is connected to the cover unit 12, there is a passageway 32. Passageway 32 is present to allow the cable to pass from the hinge 15 into the cover unit 12. The tubular portion 17 has a relatively thin surface wall. It may also contain reinforcing panels (not shown) to help maintain the cylindrical shape of the hinge.

One of the major advantages of the present invention is that it allows the cable 20 connecting the electronic components in the main body section 10 and the cover unit 12 to be routed in a manner that is easy to assemble and does not place a large amount of strain on the cable. The cable 20 enters the hinge 15 through the gap 23 in the curved cover piece 22. In the preferred embodiment, the cable 20 is a .flat cable with a plurality of parallel conductors. This type of cable changes direction by being folded along approximately a 45° line so that it makes a right angle bend. After the cable enters the hinge, it is bent so as to travel substantially parallel to the central axis of the hinge 15. The cable 20 extends a predetermined length through the hinge. At that point, the cable is bent again and passes into the cover unit 12, through passageway 32, where it converts to the electronic components associated with the video display of the computer system. The manner in which the cable is folded in the preferred embodiment is shown in FIGS. 5*a* and 5*b*.

This arrangement of the cable 20 is advantageous in that it does not place a large amount of strain on the cable when the cover unit is opened and closed. In this arrangement, the cable 20 twists along the entire length which is in the hinge whenever the cover unit 12 is moved. This is in contrast to the prior art where the cable would bend at a single point. By twisting along a length, the conductive elements in the cable are not subjected to work hardening and therefore do not become brittle over time. This leads to a longer, useful life for the cable. Another advantage of the present arrangement is that the cable can be completely assembled before it is placed in the computer system. It is not necessary to add connectors to the cable after it has been placed in the computer. That manufacturing step can be accomplished while the cable 20 is outside the computer. This greatly simplifies the manufacture of the computer system.

What is claimed is:

1. An improved hinge assembly for use in a computer system, said computer system having at least a main body section and a cover unit comprising:

a hinge means fixedly coupled to said cover unit, said hinge means having removed therefrom a longitudinal section so as to form a gap;

a cover means movably coupled to said hinge means, covering said gap;

a stop means coupled to the main body section for holding said cover section in a fixed orientation relative to the main body section.

2. The device of claim **1** wherein said cover means has formed , therein an opening so as to allow a cable to be passed through said opening, through said hinge means and into said cover unit.

3. The device of claim **1** wherein said hinge, said cover section and said stop member are manufactured from injection molded plastic.

4. An improved hinge assembly for use in a portable personal computer system, said hinge assembly comprising:

first and second substantially hollow tubular portions fixedly coupled to a cover unit of the computer system, said first and second tubular portions being arranged so as to form a gap therebetween:

mounting means coupled to said first and second tubular portions for rotatably coupling said cover unit to a main body section of said computer system;

a curved cover element movably coupled to said first and second tubular portions, covering said gap, said cover section having formed therein an opening;

a stop member coupled to said main body section and contacting said curved cover element so as to hold said curved cover element in a fixed orientation with respect to said main body section.

5. The device of claim **4** further comprising a cable, said cable passing through said opening in said curved cover element, entering one of said tubular portions, and travelling into said cover unit so as to electrically connect electronic components in said cover unit and said main body section.

6. The device of claim **4** wherein said first and second tubular portions and said curved cover element are all substantially circular in cross section, and have substantially identical cross sectional diameters.

7. The device of claim **4** wherein said stop member comprises an extended arm coupled to, and rising upwards from, a base of said main body section.

8. The device of claim **5** wherein said cable comprises a flat cable with a plurality of parallel conductors.

9. The device of claim **4** wherein said curved cover element is movably coupled to said tubular portions by being placed around a recessed end of each of said tubular portions.

10. The device of claim **4** wherein said tubular portions, said curved cover element, and said stop member are all manufactured from injection molded plastic.

* * * * *