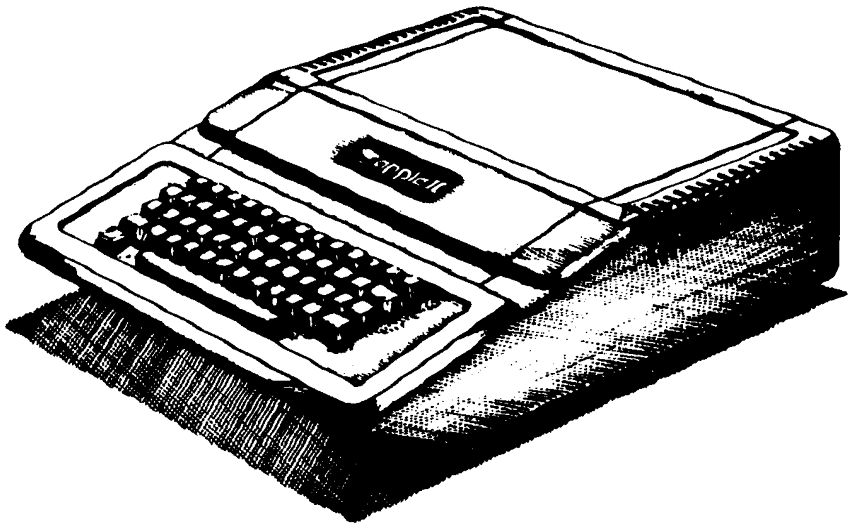
Apple 2 Computer Technical Information



Apple][Computer Family Information

AppleSoft BASIC Info:
Notes on Hi-Res Graphics Routines in AppleSoft

Mesztenyi – Apple Orchard – Spring 1981

Document # 47

Ex Libris David T. Craig

Apple 2 Computer Information • Document 047

NOTES ON HI-RES GRAPHICS ROUTINES IN APPLESOFT

by C. K. Mesztenyi
Washington Apple Pi

Checking out the entry points given by J. Crossley in the article "APPLESOFT INTERNAL ENTRYPOINTS" in the March/April 1980 Apple Orchard, I found the given entry points were 4 bytes off from the given ones in our APPLE II Plus. Furthermore, after checking out the routines in more detail, I thought to share my experiments with other APPLE II Plus owners interested in machine language programming. In the first section, I describe the essential data storage area, in the second I give the entry points of the subroutines somewhat more detailed than in the above article, and in the last section I give some listings of instructions following the entry points so that one could identify it for different versions of Applesoft.

1. DATA STRUCTURE

There are four data in five memory locations which specify a point on the high resolution screen (whether the screen is displayed or not, is irrelevant). I call these data collectively as external cursor data. The five memory locations, and their contents are as follows:

$E0:	Low order bits of the horizontal screen coordinate
$E1:	High order bit of the horizontal screen coordinate
$E2:	Vertical screen coordinate
$E4:	Color masking word from the color table ($F6F6-$F6FD)
$E6:	Page indicator ($20 for Page 1, $40 for Page 2)

I have called the above set of data as external cursor data since the actual point plot is performed by the following five instructions:

```
LDA  $1C
EOR  ($26),Y
AND  $30
EOR  ($26),Y
STA  ($26),Y
```

which uses data located at $1C, $26, $27, register Y and $30. The contents of register Y are always picked up from location $E5 prior to the above instructions, thus we may call the data in the following five locations as internal cursor data:

$1C:	The color masking byte shifted for odd address and none black or white, unchanged otherwise.
$26, $27:	(Low, high order) address of the byte corresponding to the page, vertical coordinate and leftmost seven points of the screen.
$E5	(register Y): The integer part of the horizontal screen coordinate divided by 7.
$30:	The bit position taken from Bit Position Table corresponding to the remainder of the horizontal coordinate divided by 7.

These two cursor data (external and internal) are equivalent in the sense that given one, the other can be derived from it. There would be no need to make any distinction if they would correspond to each other all the time, but unfortunately, this is not always the case, e.g. the following sequence of BASIC instructions:

```
HCOLOR=1
HPLOT 0,0 TO 10,10
HCOLOR=2
HPLOT TO 10,50
```

plots two lines, (0,0) to (10,10) and (10,10) to (10,50), both with color 1, i.e. HCOLOR=2 has no effect. Actually it resets the color code in $E4 but it does not change $1C, and the statement HPLOT TO picks up whatever was left in $1C.

A machine language programmer can write his/her own graphics routines which takes time and uses sometimes much-needed memory space. Thus using the available programs in Applesoft ROM can be advantageous. If execution time is also important, as in the case of animation, then one should concentrate only on the internal cursor data, and modify the external cursor only when it is necessary. The entry points INTX and INTY, provide the basic routines for incremental plotting which are not available directly in BASIC. Also, modifying the external cursor coordinates allows the use of HLINE with off-set.

"DTCA2DOC-047-01.PICT" 242 KB 2001-03-29 dpi: 300h x 300v pix: 2232h x 3093v

2. ENTRY POINTS IN APPLESOFT

Page and Color:

HGR2 ($F3D8): Displays page 2 with all graphics mode, sets $E6 to $40, clears page 2 (black) and sets $1C to zero (black I).

HGR ($F3E2): Displays page 1 in mixed mode, sets $E6 to $20, clears page 1 (black) and sets $1C to zero (black I).

BKGND ($F3F4): Clears the page defined by $E6 to the color defined by the contents of register A, which should be one from the Color Masking Table. Also stores register A in $1C.

HCOLOR ($F6F0): Assumes register X contains the color index (0 to 7). The routine picks up the appropriate color code from the Color Masking Table and stores it in $E4.

Positioning Entries:

HPOSN ($F411): Assumes the input upon entry in the registers as:

register X = low order bits of the horizontal screen coordinate,

register Y = high order bit of the horizontal screen coordinate,

register A = vertical screen coordinate.

The routine stores the registers in $E0, $E1 and $E2. Then, using $E6, sets $26, $27, $30 and $E5 together with register Y, and sets $1C to the contents of $E4. Thus this routine makes the internal cursor equivalent to the external one.

INTX ($F465): Modifies the internal cursor data in $1C, $E5, register Y and $30 so that it corresponds to incrementing/decrementing the horizontal screen coordinate X by one. Upon entry, if the N-flag is zero (positive) then it increments; if N is set (negative) then it decrements. The modification has a wrap around feature, i.e., incrementing/decrementing at the extreme sides of the screen defined by the internal cursor causes it to come back on the other side. The routine assumes that register Y corresponds to $E5 upon entry, and leaves the routine correctly modified if necessary.

Upon testing the N-flag the routine jumps to DECRX or INCRX.

DECRX ($F467): The routine modifies the internal cursor data by decrementing the horizontal screen coordinate by 1 (see INTX).

INCRX ($F48A): The routine modifies the internal cursor data by incrementing the horizontal screen coordinate by 1 (see INTX).

INTY ($F4D3): Modifies the internal cursor data in $26, $27, so that it corresponds to incrementing/decrementing the vertical screen coordinate by one. Upon entry, the N-flag is checked, and if it is set (negative) then goes to INCRY to increment by one, if it is not set (positive) then goes to DECRY to decrement by one. Note that the sign convention is used opposite of INTX. These entries also have the wraparound features, i.e. if the incrementation/decrementation causes the cursor to leave the screen on the bottom/top, then it comes back on the top/bottom.

DECRY ($F4D5): The routine modifies the internal cursor data by decrementing the vertical screen coordinate by 1 (see INTY).

INCRY ($F504): The routine modifies the internal cursor data by incrementing the vertical coordinate by 1 (see INTY).

IPOSN ($F5CB): Sets the external cursor data in $E0, $E1, $E2 equivalent to the internal cursor coordinate data.

Plotting Entries:

HPLOT ($F457): Assumes input data in the registers as HPOSN:

register X: low order bits of horizontal screen coordinate,

register Y: high order bit of horizontal screen coordinate,

register A: vertical screen coordinate.

The routine calls HPOSN with the above data, then goes to PLOT.

PLOT ($F45A): The routine executes the five instructions listed in the beginning of the article which plots a point using the internal cursor data. If this entry is used directly, then the user should make sure that register Y contains the data from $E5.

HLINE ($F53A): The routine assumes input in the registers:

register A: low order bits of horizontal screen coordinate,

register X: high order bit of horizontal screen coordinate,

register Y: vertical screen coordinate.

(Note that it is in different order than (HPOSN.)

The routine draws a line from the internal cursor position to the point defined by the input. Upon exit, it leaves the external cursor data corresponding to the input, the internal cursor data corresponding to the last plot point of the line. If the internal and external cursor data were not equivalent, then an off-set occurs. This can be visualized as follows:

Draw a line segment from the external cursor coordinates to the input coordinates. Now move this line segment parallel to itself so that the endpoint at the external cursor position gets into the internal cursor position. This is the actual line segment which will be drawn. If it gets outside of the screen, then a wraparound occurs, i.e. it comes back on the opposite side of the screen.

APPENDIX

The first few instructions are listed for each entry point so that one could identify them using the Monitor list feature.

Bit Position Table:

```
$F5B2:  $81=10000001
$F5B3:  $82=10000010
$F5B4:  $84=10000100
$F5B5:  $88=10001000
$F5B6:  $90=10010000
$F5B7:  $A0=10100000
$F5B8:  $C0=11000000
```

Color Masking Table:

```
$F6F6:  $00=00000000 (black I)
$F6F7:  $2A=00101010
$F6F8:  $55=01010101
$F6F9:  $7F=01111111 (white I)
$F6FA:  $80=10000000 (black II)
$F6FB:  $AA=10101010
$F6FC:  $D5=11010101
$F6FD:  $FF=11111111 (white II)
```

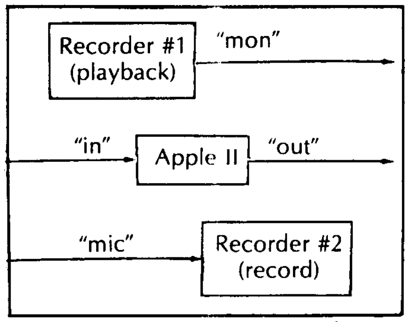
```
HGR2:      $F3D8:     BIT $C055
                      BIT $C052
                      LDA #$40
                      BNE $F3EA

HGR:       $F3E2:     LDA #$20
                      BIT $C054
                      BIT $C053
                      STA $E6
 ...
BKGND:     $F3F4:     STA $1C
                      LDA $E6
                      STA $1B
                      LDY $#00
 ...
HCOLOR:    $F6F0:     LDA $F6F6,X
                      STA $E4
                      RTS

HPOSN:     $F411:     STA $E2
                      STX $E0
                      STY $E1
                      PHA
                      AND #$C0
                      STA $26
 ...
IPOSN:     $F5CB:     LDA $26
                      ASL
                      LDA $27
                      AND #$03
                      ROL
                      ORA $26
 ...
INTX:      $F465:     BPL $F48A
DECRX:     $F467:     LDA $30
                      LSR
                      BCS $F471
                      EOR #$C0
 ...
INCRX:     $F48A:     LDA $30
                      ASL
                      EOR #$80
                      BMI $F46E

INTY:      $F4D3:     BMI $F505
DECRY:     $F4D5:     CLC
                      LDA $27
                      BIT $F5B9
                      PNE $F4FF
 ...
INCRY:     $F505:     CLC
                      LDA $27
                      ADC #$04
                      BIT $F5B9

HPLOT:     $F457:     JSR $F411
                      LDA $1C
                      EOR ($26),Y
                      AND $30
                      EOR ($26),Y
                      STA ($26),Y
                      RTS
HLINE:     $F53A:     PHA
                      SEC
                      SPC $E0
                      PHA
                      TXA
                      SPC $E1
```

AN APPLE II QUICKIE

by Gordon Stallings
Tulsa Computer Society

Here is a simple program which has been in use by the TCS Apple Users for the past year, and which should be more widely known:

To copy a cassette tape from one recorder to another, you can use the Apple II as an intermediary to restore the signal levels.



Put this program into the Apple II:

```
0000 20 FD FC JSR  $FCFD
0003 AD 20 C0 LDA  $C020
0006 4C 00 00 JMP  $0000
```

Start the program: *0G

As long as this program is running, any signal received from recorder #1 will be sent to recorder #2. For best results, the following tips should be observed:

1. Be sure that Recorder #1 is set up to reliably read the tape which you are wanting to copy–volume, tone, and head alignment must be set so that the Apple can read the tape.
2. Recorder #2 should have correct head alignment so that the new tape will be compatible with other machines.

HOW IT WORKS

The JSR calls subroutine $FCFD in the Monitor ROM, which watches the cassette input port waiting for a change of state, which indicates a zero-crossing on the playback tape. When the transition occurs, the subroutine RETURNS. The LDA $C020 toggles the cassette output port, recording a transition on the new tape. The JMP closes a program loop which can only be broken by RESET.