ASK NIBBLE

Q *Where can you get answers to all your Apple questions?*

A *Ask Nibble of course! Send your questions to Ask Nibble, 52 Domino Dr., Concord, MA 01742.*

Q *How can I print the Hi-Res screen on my ImageWriter printer?*

A This has been one of the most frequently asked questions in the last year, and although the answer cannot be given in just a paragraph or two, this information is so useful that I'll present the key pieces of information in this month's column.

Since many of you will want the information presented for an Applesoft program, and the rest of you will want it in assembly language, I'm granting both your wishes and doing it in both languages at once! Actually, some low-level operations are very difficult to do in BASIC, and so you really need some assembly language routines. On the other hand, given an assembly language routine to handle the bits on the screen, the main program to do the screen dump can be done in a very few lines of BASIC.

In theory, printing graphics on the ImageWriter II (this same routine also works on the ImageWriter I and Scribe) is very simple. You simply send the printer an initializing code, and thereafter every set bit in each screen byte causes a single pixel to print on your graphics image. This is identical to the way Hi-Res graphics is created on the screen, so everything should be easy.

Here's the catch: The Hi-Res screen uses the first seven bits in a byte to put individual dots on the screen *horizontally* (the eighth bit is ignored). The printer, on the other hand, prints all eight bits in each byte *vertically*. So, to print the Hi-Res screen, you have to take the first bit from each of the first eight rows and combine them into the first byte that goes to the printer. Then, the second bit from each screen row is taken and put into the second byte. Note that bit 0 (the least significant bit, or LSB) in screen memory is the leftmost pixel on the screen. On the printer, the LSB of the byte sent is the topmost dot of the column printed.

Figure 1 shows the upper left-hand corner of the Hi-Res screen as drawn by the sample screen dump program (Listing 3). The program first draws a frame around the entire screen, and then draws a diagonal line from the upper-left corner toward the middle of the screen. The diagram shows an enlarged view of both the screen and printer pixels, and shows the data bytes for each. For the screen, the first group of eight bytes are $7F, $07, $0D, $19, $31, $61, $41 and $01. Each byte represents seven horizontal pixels on the screen.

Roger Wagner is the president of Roger Wagner Publishing, Inc., the publisher of MouseWrite and the Merlin Assembler, and the author of Assembly Lines: The Book *and* Apple IIGS: Assembly Language for Beginners. *The program is compatible with both ProDOS and DOS 3.3.*

To print this, the program must send out seven new bytes,each of which contains a vertical group of eight pixels. As seen in Figure 1, the bytes sent to the printer are not the same as the bytes in screen memory ($2000 to $3FFF).

The approach I have taken is to write an assembly language routine that takes a given X,Y screen coordinate and returns seven bytes containing that pixel within a block that is seven pixels wide and eight pixels high. This is convenient for a number of reasons. First, it is a logical chunk of memory to process — the lowest common denominator of the screen and the printer, as it were. Also, by having the routine process only a part of memory, some limited degree of partial screen printing is available, allowing you to print just part of a Hi-Res screen if you wish. Last of all, the 7 x 8 pattern corresponds to the most common Hi-Res character generator pattern, which means that the screen chunks printed perfectly match single characters put on the screen by most Hi-Res printing routines.

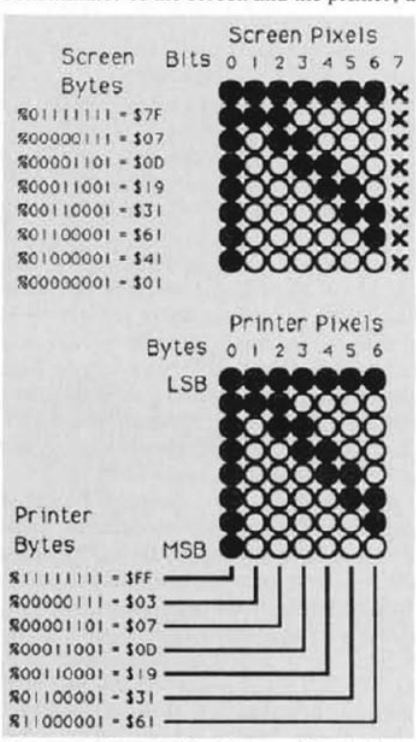

Figure 1: Converting Screen Pixels to Printer Dots

To start, Listing 1 shows you the screen processing routine. I'll summarize the main points of the routine. The routine is called with the statement CALL *AD,X,Y,A$*, where *AD* is the routine's address, *x*, *y* is the starting point of the screen block, and *A$* is the string to return. The START section checks to make sure we're not in BASIC's immediate mode. This is because the program uses part of the input buffer ($280-286) as the work area to build the bytes to send to the printer. XPOSN and TESTX then evaluate the Applesoft variable for the horizontal position, and tests to make sure X is not greater than 273, since we'll be reading from there to X=279. YPOSN does a similar function for the Y position.

The processing routine itself will set up seven bytes in memory, all set to zero, and then use the ROR instruction to move bits into each byte of the output group as the screen bytes are scanned. Locating the byte in memory that corresponds to a given screen position can be tricky, so we'll use the built-in Applesoft routine HPOSN.

SETUP clears the seven bytes in our buffer to zero; then LOCATE uses HPOSN to find the base address of the upper-left corner of the screen memory byte group. This "home" value is saved in XVAL and YVAL, so that we can return there after each vertical row of bits is scanned.

GETCOL is the beginning of the real working part of the routine, and sets a MASK value to $01, corresponding to our examination of the first bit of the screen byte in the first row. TOP initializes the pointer GBASL to use indirect addressing as we scan the bytes. We'll also set up a counter to tell us when we've scanned the eight rows of screen bytes.

GETBYTE gets the byte from screen memory; by ANDing this with MASK, we can tell whether the first pixel (bit 0) is on. If it is, the Carry bit is set in SET and pushed into the first byte in the buffer. NXTBYTE uses the Hi-Res routine INCRY, which automatically adjusts GBASL to point to the next line down for each pass through the loop. On each successive loop through GETBYTE, successive bits are pushed into this first byte, until we've pushed all eight bits. NXTCOL increments the X-register so successive loops each act on each of the bytes in the buffer. After seven loops, the bit in MASK will have moved to position 7, which makes MASK "negative" ($80), and the BPL TOP test fails.

SENDSTR then sends the eight bytes back to Applesoft as a string variable. One interesting tidbit here, as a reward for those who actually read the text of program explanations, is that this technique is different from the general purpose string-passing method presented last year. In this method, it is the *length* of the string that controls what is sent back to Applesoft, not a terminating character such as $00 or a carriage return, as is more commonly used. This is necessary because the graphic data being sent back in the string can have any value from $00 to $FF, and thus there is no "safe" terminating character that might not be found in the string data itself.

The BASIC program is in **Listing 3**. The program begins by loading our bitmap routine at $2A0. Normally, I would have chosen $300 as a location for this, but in a moment we're going to need one more routine that is used to send the bytes to the printer, and the two together just won't fit into the available space byte on page 3. Since we're already using the input buffer starting at $280, there's really no reason not to use the memory from $2A0 on for our routines. (One possible exception: Some clock cards use the last part of the input buffer, but more on the alternatives later.)

Graphics printing on the ImageWriter and compatible printers is very easy, at least in regards to setting things up. After the usual PR#1 to turn on the printer, we have to put the interface card into the "transparent mode." The transparent mode means the printer card won't be looking for a Control-I in the data stream, and likewise, won't add line feeds to carriage return bytes. This is necessary because a graphics byte can be any value from $00 to $FF, and the values $09, $0D, $89, $8D, and any others that the interface card might respond to have to be ignored. However, since the printer card won't be adding linefeeds, we will have to do this ourselves during the printout.

Escape n Escape T16 sets the character pitch and linefeed distance for a properly proportioned picture. You can experiment with other values and pitch commands to get smaller, denser pictures or larger images as you desire.

The actual graphics printing is initiated by telling the printer how many graphic bytes you are sending. The command is Escape G *nnnn*, where *nnnn* is the number of bytes (i.e., groups of eight vertical pixels) you're sending. In our case, we're sending 280 bytes for the Hi-Res screen.

Next, a nested loop scans positions 0 to 23 vertically. The actual Y coordinate is determined by multiplying the Y value by 8 (8 bytes per block). I could have used FOR Y = 0 TO 194 STEP 8, but I chose this form to show the correspondence to the positions on the text screen, in case you want to use this routine with a Hi-Res character generator.

The X loop likewise counts from 0 to 39, which is multiplied by 7 to get the true X coordinate. The call to the bitmap routine then returns the seven bytes to send to the printer.

The only catch now is printing the full range of data bytes to the printer. If you've ever experimented with this, you've found that Applesoft automatically sets the high bit on everything you try to print. If your program says PRINT CHR$(20), the printer actually receives CHR$(20+128). Fortunately, it is easy to create an alternate PRINT routine to the usual Applesoft command. **Listing 4** is a routine that will print any string to the printer without conversion.

Graphics printing on the ImageWriter is very easy, at least in regards to setting things up.

The routine is actually very simple. It just sends the characters in the string you specify directly to COUT ($FDED = "Character Output"), bypassing the Applesoft routines entirely.

Final notes: One of the big problems with assembly language routines is the problem of where to put them in memory and how to fit several of them at once in a given area. These routines have been written to be completely position-independent, so they can be used at any location you wish. That is to say, if you received the Trial Size Toolbox offered for free in this column several times in the past, you can use the Workbench program on that disk to add these routines directly to any Applesoft program without having to worry at all about finding a memory location. The syntax for the routines in that case is &"ALT.PRINT",A$ and &"ROTATE",X,Y,A$.

I hope this has helped you get the general idea behind printing graphics on your printer. If you do not have an ImageWriter printer or compatible, try looking through the reference manual that came with your printer. Chances are that except for minor differences in the printer initialization sequence, you'll be able to use the general principles provided here with very little modification. Have fun!

ENTERING THE PROGRAMS

If you have an assembler, use the source code from **Listing 1** and save the object code as ROTATE. If you don't have an assembler, use the hex code from **Listing 2** and save it with the command

```
BSAVE ROTATE,A$8000,L$AA
```

Enter the Applesoft program in **Listing 3** and save it with the command

```
SAVE IW.PRINT.DEMO
```

If you have an assembler, use the source code from **Listing 4** and save the object code as ALT.PRINT. If you don't have an assembler, use the hex code from **Listing 5** and save it with the command

```
BSAVE ALT.PRINT,A$8000,$2C
```

P.S. If you haven't gotten your Trial Size Toolbox, you can still receive one *for free* by writing to me at Roger Wagner Publishing, Inc., 1050 Pioneer Way, Suite "P", El Cajon, CA 92020, and mentioning that you're an Ask Nibble reader!

LISTING 1: ROTATE Source Code

```
*
* ROTATE Source Code
* by Roger Wagner
* Copyright(c) 1988
* MicroSPARC, Inc.
* Concord, MA 01742
* Merlin 8/16 Assembler
*

BUFFER   EQU   $280       ; PLACE TO BUILD OUR STRING

HPOSN    EQU   $F411
INCRY    EQU   $F504
GBASL    EQU   $26        ; $26,27 = BASE ADDR. OF SCREEN LINE
YCURS    EQU   $E5        ; OFFSET TO SCREEN BYTE

MASK     EQU   $3C
CTR      EQU   $3D
XVAL     EQU   $3E        ; $3F,40
YVAL     EQU   $41

FRMNUM   EQU   $DD67      ; EVALUATE NUMERIC EXPRESSION
GETADR   EQU   $E752      ; CONVERT FAC TO INTEGER
LINNUM   EQU   $50        ; $50,51

ILDIR    EQU   $E306      ; CHECK FOR DIRECT MODE
FPERR    EQU   $D412      ; APPLESOFT ERR ROUTINE

COMBYT   EQU   $E74C      ; GET COMMA AND VALUE < 256
CHRGOT   EQU   $B7
CHKCOM   EQU   $DEBE      ; CHECK FOR COMMA
PTRGET   EQU   $DFE3
CHKSTR   EQU   $DD6C
FORPNT   EQU   $85        ; $85,86
STRSPA   EQU   $E3DD
MOVSTR   EQU   $E5E2
PUTNEW   EQU   $E42A
SAVD     EQU   $DA9A

START    JSR   ILDIR      ; MAKE SURE WE'RE NOT IN IMME MODE
         JSR   CHRGOT     ; CHECK CHAR AT TXTPTR
         CMP   #','       ; CHECK FOR COMMA
         BNE   XPOSN      ; NO COMMA
         JSR   CHKCOM     ; ADVANCE TXTPTR

XPOSN    JSR   FRMNUM     ; EVALUATE EXPRESSION
         JSR   GETADR     ; CONVERT TO INTEGER
         LDA   LINNUM     ; LOW BYTE OF RESULT
         STA   XVAL       ; STORE IT
         LDA   LINNUM+1   ; HI BYTE OF RESULT
         STA   XVAL+1     ; STORE IT

TESTX    CMP   #>274      ; HIGH BYTE OF MAX VALUE (I.E., "1")
         BNE   YPOSN
         LDA   XVAL
         CMP   #<274      ; LOW BYTE OF MAX VALUE
         BCC   YPOSN
         LDX   #53
         JMP   FPERR      ; 'ILLEGAL QTY ERROR'

YPOSN    JSR   COMBYT     ; CHECK COMMA AND EVALUATE
         STX   YVAL       ; STORE Y POSN
         CPX   #185       ; MAX Y VALUE
         BCC   SETUP
         LDX   #53
         JMP   FPERR      ; 'ILLEGAL QTY ERROR'

SETUP    LDX   #$06
         LDA   #$00
:LOOP    STA   BUFFER,X   ; STORE ZERO BYTE
         DEX
         BPL   :LOOP      ; $280-286 = $00'S
                          ; (7 BYTES)

LOCATE   LDX   XVAL       ; LOW BYTE OF X POSN
         LDY   XVAL+1     ; HIGH BYTE OF X POSN
         LDA   YVAL
         JSR   HPOSN      ; SET UP $26,27=BASE ADDRESS
                          ; $E5 = HORIZ. LINE OFFSET

         LDA   GBASL      ; SAVE VALUES
         STA   XVAL
         LDA   GBASL+1
         STA   XVAL+1
         LDA   YCURS
         STY   YVAL

GETCOL   LDA   #$01       ; SET MASK TO BIT 1 = 1ST PIXEL
         STA   MASK
         LDX   #$00       ; FIRST BYTE IN BUFFER
         LDY   YCURS

TOP      LDA   #$08
         STA   CTR        ; COUNTER FOR LINE #
         LDA   XVAL
         STA   GBASL
         LDA   XVAL+1
         STA   GBASL+1    ; RESTORE HIRES CURSOR
         LDY   YVAL

GETBYTE  LDA   (GBASL),Y  ; GET SCREEN BYTE
         AND   MASK       ; TEST BIT
         BEQ   CLR        ; PIXEL NOT LIT
```

```
105
106 SET      SEC
107 INJECT1  ROR   BUFFER,X    ; INJECT SET BIT IN BYTE X OF BUFFER
108          BNE   NXTBYTE     ; ALWAYS
109
110 CLR      CLC
111 INJECT2  ROR   BUFFER,X    ; INJECT CLR BIT IN BYTE X OF BUFFER
112
113 NXTBYTE  JSR   INCRY       ; MOVE DOWN ONE LINE
114          DEC   CTR
115          BNE   GETBYTE     ; GET NEXT PIXEL DOWN
116
117 NXTCOL   INX                ; NEXT BYTE IN BUFFER
118          ASL   MASK        ; NEXT BIT IN TEST MASK
119          BPL   TOP         ; RE-POSN CURSOR AND GET NEXT COL.
120
121 SENDSTR  JSR   CHKCOM      ; MOVE PAST COMMA
122          JSR   PTRGET      ; FIND VARIABLE OR MAKE IT
123          JSR   CHKSTR      ; MAKE SURE IT'S A STRING
124          STA   FORPNT      ; A,Y = DESCRIPTOR TO STRING
125          STY   FORPNT+1
126
127          LDA   #$07        ; LEN OF STRING (7 CHARS)
128          JSR   STRSPA      ; SET UP SPACE IN MEMORY
129          LDX   #<BUFFER    ; LOW BYTE OF DATA
130          LDY   #>BUFFER    ; HIGH BYTE OF DATA
131          JSR   MOVSTR      ; MOVE DATA TO MEMORY
132          JSR   PUTNEW      ; MOVE DESCRIPTOR TO TEMP
133          JSR   SAVD        ; ASSIGN TO VARIABLE
134
135 DONE     RTS
136
137 CHKSUM   CHK               ; CHECKSUM = 4B (OMIT FOR OTHER ASSEMBLERS)
```

END OF LISTING 1

LISTING 2: ROTATE

Start: 8000 Length: AA

```
5E | 8000:20 06 E3 20 B7 00 C9 2C
A3 | 8008:D0 03 20 BE DE 20 67 DD
68 | 8010:20 52 E7 A5 50 85 3E A5
E3 | 8018:51 85 3F C9 01 D0 0B A5
16 | 8020:3E C9 12 90 05 A2 35 4C
85 | 8028:12 D4 20 4C E7 86 41 E0
39 | 8030:B9 90 05 A2 35 4C 12 D4
E1 | 8038:A2 06 A9 00 9D 80 02 CA
49 | 8040:10 FA A6 3E A4 3F A5 41
9A | 8048:20 11 F4 A5 26 85 3E A5
0E | 8050:27 85 3F A5 E5 84 41 A9
F7 | 8058:01 85 3C A2 00 A4 E5 A9
B0 | 8060:08 85 3D A5 3E 85 26 A5
AE | 8068:3F 85 27 A4 41 B1 26 25
D1 | 8070:3C F0 06 38 7E 80 02 D0
DA | 8078:04 18 7E 80 02 20 04 F5
7D | 8080:C6 3D D0 E9 E8 06 3C 10
5E | 8088:D6 20 BE DE 20 E3 DF 20
9E | 8090:6C DD 85 85 84 86 A9 07
DB | 8098:20 DD E3 A2 80 A0 02 20
99 | 80A0:E2 E5 20 2A E4 20 9A DA
26 | 80A8:60 4B
```

TOTAL: AE58

END OF LISTING 2

LISTING 3: IW.PRINT.DEMO

```
CB |  1  REM  *
47 |  2  REM  * IW.PRINT.DEMO
47 |  3  REM  * BY ROGER WAGNER
3F |  4  REM  * COPYRIGHT(C) 1988
3B |  5  REM  * MICROSPARC, INC.
BF |  6  REM  * CONCORD, MA 01742
BF |  7  REM  *
99 |  10  PRINT  CHR$ (4);"BLOAD ROTATE.A$2A0": REM
      672 DEC.
AB |  20  PRINT  CHR$ (4);"BLOAD ALT.PRINT.A$350":
      REM  848 DEC.
B9 |  30  REM  DRAW PICTURE
01 |  40  HGR
AA |  50  HCOLOR= 3
ED |  60  HPLOT 0,0 TO 100,100
AE |  70  HPLOT 0,0 TO 279,0 TO 279,191 TO 0,191 TO 0
      ,0
3A |  80  REM  PRINTER DUMP
49 |  90  PRINT  CHR$ (4);"PR#1"
92 |  100  PRINT  CHR$ (9);"K"; CHR$ (9);"Z": REM  NO
      LF W/ CR; TRANSP. MODE
4E |  110  PRINT  CHR$ (27);"n"; CHR$ (27);"T16"
A6 |  120  FOR Y = 0 TO 23
94 |  130  PRINT  CHR$ (27);"G0280";
6A |  140  FOR X = 0 TO 39
97 |  150  CALL 672,X * 7,Y * 8,A$: REM  GET SCREEN B
      YTES
49 |  160  CALL 848,A$: REM  TRANSPARENT PRINT
6D |  170  NEXT X: PRINT  CHR$ (10)
0C |  180  NEXT Y
05 |  190  PRINT  CHR$ (4);"PR#0"
```

TOTAL: 0F31

END OF LISTING 3

LISTING 4: ALT.PRINT Source Code

```
 1 *
 2 * ALT.PRINT Source Code
 3 * by Roger Wagner
 4 * Copyright(c) 1988
 5 * MicroSPARC, Inc.
 6 * Concord, MA 01742
 7 * Merlin 8/16
 8 *
 9
10 COUT     EQU   $FDED       ; CHARACTER OUTPUT ROUTINE
11 LEN      EQU   $9D         ; TEMP LOCATION FOR STR$ LENGTH
12                            ; (NORMALLY FAC)
13 FRMEVL   EQU   $DD7B
14 FRESTR   EQU   $E5FD
15 INDEX    EQU   $5E         ; $5E,5F
16 ILDIR    EQU   $E306       ; CHECK FOR DIRECT MODE
17
18 CHRGOT   EQU   $B7
19 CHKCOM   EQU   $DEBE
20 FPERR    EQU   $D412
21
22 START    JSR   ILDIR       ; MAKE SURE WE'RE NOT IN IMMED MODE
23          JSR   CHRGOT
24          CMP   #','        ; CHECK FOR COMMA
25          BNE   NAME        ; NO COMMA
26          JSR   CHKCOM      ; ADVANCE TXTPTR
27
28 NAME     JSR   FRMEVL      ; EVALUATE STRING EXPRESSION
29          JSR   FRESTR      ; MAKE SURE IT'S A STRING AND
30                            ; SET UP POINTERS
31                            ; (A=LEN, INDEX=LOC)
32
33 CHECK    CMP   #$00        ; CHECK FOR LEN = 0
34          BNE   PRINT
35 ERR      LDX   #$3         ; ILL QTY ERR
36          JMP   FPERR
37
38 PRINT    STA   LEN         ; SAVE LENGTH
39          LDY   #$00        ; START AT BEG. OF STRING
40 LOOP     LDA   (INDEX),Y   ; GET CHAR OF NAME STRING
41          JSR   COUT        ; PRINT IT
42          INY
43          CPY   LEN         ; DONE YET?
44          BCC   LOOP        ; NOPE!
45
46 DONE     RTS               ; ALL DONE!
47
48 CHKSUM   CHK               ; CHECKSUM = 6B (OMIT FOR OTHER ASSEMBLERS)
```

END OF LISTING 4

LISTING 5: ALT.PRINT

Start: 8000 Length: 2C

```
5E | 8000:20 06 E3 20 B7 00 C9 2C
D7 | 8008:D0 03 20 BE DE 20 7B DD
79 | 8010:20 FD E5 C9 00 D0 05 A2
AE | 8018:35 4C 12 D4 85 9D A0 00
23 | 8020:B1 5E 20 ED FD C8 C4 9D
B6 | 8028:90 F6 60 6B
```

TOTAL: B4B1

END OF LISTING 5