12-PAGE MOVIES

by
Don Lloyd

Taking Jack Krammer's hint in "New Improved Target Game with Jumping Man" (Questdata Vol. 1 Issue 9), I started building this combination draw–and–animate program for the 4K Super Elf. A blinking cursor can be made to write or erase; the user then chooses from a set of options including: move cursor one bit in any direction, change address, change page, save page, patch to user subroutine(s), and show movie. Once you've used the option codes a few times interaction with this program becomes quite fast.

The DMA address is Page 03, and saved pages are 04 through 0F. One note on terminology: to avoid double use of terms, I will refer to the seven–segment hex readout as the 'display' and to the image on the video monitor as the 'screen'. The paragraphs below are keyed both to the function diagram and to sections of the listing.

I. One–page video interrupt routine.

II. As soon as you press and release 'R' and 'G', the Q LED comes on and you input your choice of background: '00' for black, 'FF' for white (reverse video) or any other entry for patterned background for your drawings.
This style of writing an input:

ADDR	CODE
00	3F 00 6C 64 22
05	37 02

allows you to double check the entry as you are making it by holding the 'I' key down, a handy feature especially if the input is about to become a program counter as it often does in this program. Because the input is for immediate use and need not be saved on the stack beyond the next input, the X register is decremented right away.

III. Q prompts for a second input: '62' to go to the animation program, or 'AA' to go to the drawing program.

IV. Animation program. Q prompts for two inputs: the first is the number of the last page to be screened as a loop, and the second is rate at which successive pages are to be brought to the screen. Try 0A to start with. Each of the save pages are brought (loaded) to the screen in turn, held while the delay timer is counted down, then replaced by the next page; when the limit is reached, the first save page is brought back and the process is repeated until 'I' is pressed whereupon program returns to (III). Page numbers are displayed during animation.

V. Entry to drawing program. Q prompts for a page number and first time through enter '04'; this will initialize the save area to start saving at the beginning. Subsequent returns to this subroutine bypass the initializations and allow you to call up any page for editing.

VI. This is the heart of the drawing program. The cursor (first time through) is located near the center of the page. Subsequent times through this subroutine the cursor will be located wherever you left it. To write using black background, press 'I' while the blinker is on; to erase, press 'I' while the blinker is off (for reverse video, white background, do the opposite) and DO NOT YET RELEASE: input your subroutine option before releasing 'I'. The reason for having a single input routine for both write/erase and where to? is to allow repeated calls to the same subroutine with single 'I' strokes. For example, entering '20' will move the cursor up one space; since you don't need to enter an—

other instruction to write or erase, the '20' stays there and need not be re-entered. By timing yourself with the blinker you can draw a straight line, one bit at a time, quite rapidly. To change the blinker rate, alter the byte at M(00 D8).

VII. 'A2' will fill the current page with whatever you entered in (II.) If you entered '00' then this is a 'clear' subroutine. Returns to (VI.)

VIII. There are eight movement subroutines called from (VI.).

'20' – up '28' – up and right
'3C' – right '4D' – down and right
'61' – down '69' – down and left
'7D' – left '8E' – up and left

You could spread these subroutines out over another page so that each one could begin with an address easier to remember (e.g. 20, 40, 60, 80 for square movements, etc.) but you may be surprised how quickly you will pick these up. All movement subroutines display the cursor's new address and return to (VI.).

IX. 'B3' loads what you see on the screen into the current save page and initializes the next save page, the number of which is displayed, then returns to (VI.) While you are drawing with the cursor, the display shows the cursor address, alternating with option selection. To remind yourself of the current save page number without initializing to the next save page, call 'CB' instead of 'B3'.

If the page you have just saved turns out to be the last one, 'EE' is displayed and the program returns to (III.) instead of (VI.) assuming that you are ready to animate. A shortcut back to (III.) without filling up your memory is to call 'CA' from (VI.).

X. 'D0' will issue a prompt for new cursor address.

XI. 'DB' will issue a prompt for calling any save page. This .will be used mainly after you have watched the animation and have decided to do a little editing.

XII. This is the patch to your own subroutines on Page 02. 'DF' as listed, '00' is the only allowed response to the prompt – this will draw a border around the perimeter of the current page. The border subroutine, given here without notation, is worth investigating for a way to add elements to a

drawing without disturbing drawing that may have already been done on that page. Possible subroutines here include pattern generators, alphanumerics, figures, any set element to be reused. (This is where Jack Krammer left you, too.) Complex routines may require the storage of registers for which you may want to initialize another stack. As a rule, your routines should end with 'DB' to return to (VI.).

Register Usage:

R(0) – DMA
R(1) – Interrupt routine
R(2) – Stack Pointer
R(3) – Program Select
R(4) – Utility
R(5) – P.C. for Animation or Drawing Entry
R(6) – Utility
R(7) – Utility
R(8) – Save page Pointer
R(9) – Cursor
R(A) – Cursor Address
R(B) – P.C. for Drawing Main
R(C) – P.C. for Drawing Subs
R(D) – Blinker Sub
R(E) – P.C. for Page 02 Subs
R(F) – .1 Minus Cursor
      .0 OR Cursor

```
0000 90B1 B2B3 B4B5 F816 A1F8 FFA2 F832 A3F8
0010 4EA4 E2D3 7270 C4C4 C422 7822 52F8 03B0
0020 F800 A080 E2E2 20A0 E220 A0E2 20A0 3C23
0030 3014 7B3F 336C 6422 3735 A67A 54F8 01B4
0040 F8A8 A486 5469 38D5 F803 B493 A4F8 6954
0050 1494 FB04 3A4D 7B3F 576C 6422 3759 A57A
0060 3047 93B4 B6F8 A0A4 F895 A67B 3F6C 6C64
0070 2237 6EFC 0154 3F76 6C64 2237 7856 7AF8
0080 04B8 F803 B793 A7A8 9852 6422 4857 1797
0090 FB04 3A8C F862 B4A4 2494 3A98 37A5 98FB
00A0 633A 8230 7F37 A53F A7D3 F801 BBBC BDA9
00B0 ABF8 02BE F87B AAF8 D7AD F803 B7BA 93BD
00C0 A7A8 7B3F C36C 6422 37C5 B87A 4857 1797
00D0 FB04 3ACC 6CB8 DBF8 33B6 A626 963A DB30
00E0 D600 0000 0000 0000 0000 0000 0000 0000
00F0 0000 0000 0000 0000 0000 0000 0000 25AA
0100 DC0A 5289 F1AF 5289 F5BF 7B8F 5ADD 3716
0110 7A9F 5ADD 3F07 7B6C 6422 3717 AC7A 3000
0120 8AFF 08AA 5264 22DB 89F6 A93B 348A FC01
0130 AAF8 80A9 8AFF 08AA 5264 22DB 89F6 A93B
0140 488A FC01 AAF8 80A9 8A52 6422 DB89 F6A9
0150 3B59 8AFC 01AA F880 A98A FC08 AA52 6422
0160 DB8A FC08 AA52 6422 DB89 FEA9 3B75 8AFF
0170 01AA F801 A98A FC08 AA52 6422 DB89 FEA9
0180 3B89 8AFF 01AA F801 A98A 5264 22DB 89FE
0190 A93B 9A8A FF01 AAF8 01A9 8AFF 08AA 5264
01A0 22DB F803 B493 A4F8 69A6 8654 1494 FB04
01B0 3AAA DBF8 03B4 93A4 4458 1894 FB04 3AB8
01C0 98FB 103A CBF8 EE52 6422 D398 5264 22DB
01D0 7B3F D16C 6422 37D3 AA7A DBF8 BAA5 D57B
01E0 3FE0 6C64 2237 E2AE 7ADE 0000 0000 0000
01F0 0000 0000 0000 0000 0000 0000 0000 0000
0200 F803 B693 A6F8 FFA4 8456 1686 FB08 3A08
0210 0652 F880 F156 86FC 07A6 0652 F801 F156
0220 1686 FBF8 3A10 8456 1686 3A26 DB
```

ADDR CODE	COMMENTS
I	
0000 90 B1 B2 B3 B4 B5	Initialize registers to point to:
0006 F8 16 A1	interrupt routine,
0009 F8 FF A2	stack,
000C F8 32 A3	start of program
000F F8 4E A4	location to store first input.
0012 E2 D3	Set X and go to program
0014 72 70	Interrupt exit: restore D, X, P
0016 C4 C4 C4	Interrupt entry: nop's
0019 22 78 22 52	store X, P, D on stack
001D F8 03 B0	Initialize DMA pointer
0020 F8 00 A0	
0023 80 E2	Reset DMA pointer
0025 E2 20 A0	to repeat each line
0028 E2 20 A0	4 times
002B E2 20 A0	
002E 3C 23	Test for end of window
0030 30 14	Branch to exit
II	
0032 7B	Turn on Q
0033 3F 33 6C 64 22	Load and display Switch Byte;
0038 37 35 A6	Store S.B. in R(6).0
003B 7A 54	Turn off Q; store S.B. via R(4)
003D F8 01 B4	Reset R(4) to another location
0040 F8 A8 A4	
0043 86 54	and store S.B. there as well.
III	
0045 69 38	Turn on TV; skip next instruction
0047 D5	Exit from program select
0048 F8 03 B4	Point R(4) to screen page
004B 93 A4	
004D F8 xx 54 14	Load screen with input from M(00 3C)
0051 94 FB 04 3A 4D	one byte at a time until completed
0056 7B	Turn on Q
0057 3F 57 6C 64 22	Load and display S.B.;
005C 37 59 A5	Store S.B. in R(5).0 (to become P.C.)
IV	
005F 7A 30 47	turn off Q; branch to exit
0062 93 B4 B6	point registers to locations
0065 F8 A0 A4	to store two inputs
0068 F8 95 A6	
006B 7B	turn on Q
006C 3F 6C 6C 64 22	load and display S.B.;
0071 37 6E FC 01 54	add 01 to S.B.; store via R(4)
0076 3F 76 6C 64 22	Load and display S.B.;
007B 37 78 56	store S.B. via R(6)
007E 7A	Turn off Q
007F F8 04 B8	Point R(8) to first save page
0082 F8 03 B7	Point R(7) to screen page
0085 93 A7 A8	
0088 98 52 64 22	Display save page number
008C 48 57 17	Load save page onto screen
008F 97 FB 04 3A 8C	until completed
0094 F8 yy B4 A4	Load delay timer with input from M(00 7D)
0098 24 94 3A 98	Countdown delay timer
009C 37 A5	Continue if 'I' not pressed

ADDR CODE	COMMENTS
009E 98 FB zz	Test save page number for limit from M(00 75)
00A1 3A 82	If limit not reached, load next page;
00A3 30 7F	If limit reached, branch to reset
00A5 37 A5 3F A7	If 'I' pressed at M(00 9C) wait for
00A9 D3	'I' released and branch to (III.)
V	
00AA F8 01 BB BC	Initialize registers to point to:
00AE BD A9 AB	page 02,
00B1 F8 02 BE	starting address of cursor,
00B4 F8 7B AA	
00B7 F8 D7 AD	blinker timing subroutine,
00BA F8 03 B7 BA	screen page
00BE 93 BD A7 A8	
00C2 7B	Turn on Q
00C3 3F C3 6C 64 22	Load and display S.B.;
00C8 37 C5 B8	Store S.B. in R(8).1 (select save page)
00CB 7A	Turn off Q
00CC 48 57 17	Load save page onto screen
00CF 97 FB 04 3A CC	until completed
00D4 6C B8	Reset R(8) to start of save page
VI	
00D6 DB	and go to cursor routine
00D7 F8 33 B6 A6	Load blinker timer
00DB 26 96 3A DB	Countdown timer
00DF 30 D6	Branch to exit
00FD FE FF	Stack
0100 00 DC	Cursor routine exit
0101 0A 52	Load via cursor address, put on stack,
0103 89 F1 AF	OR with cursor, put result in R(F).0,
0106 52	Replace byte on stack
0107 89 F5 BF	Subtract cursor, put result in R(F).1
010A 7B 8F 5A	Turn on Q; put cursor on screen
010D DD	Delay subroutine
010E 37 16	Continue if 'I' NOT pressed
0110 7A 9F 5A	Turn off Q; put cursor on screen
0113 DD	Delay subroutine
0114 3F 07	Repeat if 'I' not pressed
0116 7B	Turn on Q (if not already on)
0117 6C 64 22	Load and display S.B.;
011A 37 17 AC	Store S.B. in R(C).0 (to become P.C.)
011D 7A 30 00	Turn off Q and branch to exit
U	
0120 8A FF 08 AA	Subtract 08 from cursor address,
UR	
0124 52 64 22 DB	display new address, return
0128 89 F6 A9	right shift cursor
012B 3B 34	Jump ahead if no carry occurred,
012D 8A FC 01 AA	if carry occurred, adjust address
0131 F8 80 A9	and cursor

ADDR	CODE	COMMENTS
0134	8A FF 08 AA	Subtract 08 from cursor address,
0138	52 64 22 DB	Display new address, return
R		
013C	89 F6 A9	right shift cursor
013F	3B 48	Jump if no carry
0141	8A FC 01 AA	Adjust if carry
0145	F8 80 A9	
0148	8A 52 64 22 DB	Display, return
RD		
014D	89 F6 A9	right shift cursor
0150	3B 59	Jump if no carry
0152	8A FC 01 AA	Adjust if carry
0156	F8 80 A9	
0159	8A FC 08 AA	Add 08 to cursor address
015D	52 64 22 DB	Display, return
D		
0161	8A FC 08 AA	Add 08 to cursor address
0165	52 64 22 DB	Display, return
DL		
0169	89 FE A9	Left shift cursor
016C	3B 75	Jump if no carry
016E	8A FF 01 AA	Adjust if carry
0172	F8 01 A9	
0175	8A FC 08 AA	Add 08 to cursor address
0179	52 64 22 DB	Display, return
L		
017D	89 FE A9	Left shift cursor
0180	3B 89	Jump if no carry
0182	8A FF 01 AA	Adjust if carry
0186	F8 01 A9	
0189	8A 52 64 22 DB	Display, return
LU		
018E	89 FE A9	Left shift cursor
0191	3B 9A	Jump if no carry
0193	8A FF 01 AA	Adjust if carry
0197	F8 01 A9	
019A	8A FF 08 AA	Subtract 08 from cursor address
019E	52 64 22 DB	Display, return
VII		
01A2	F8 03 B4	Point R(4) to screen
01A5	93 A4	
01A7	F8 xx A6	Load R(6) with Input from M(00 43)
01AA	86 54 14	Load screen with R(6)
01AD	94 FB 04 3A AA	until completed
01B2	DB	Return
VIII		
01B3	F8 03 B4	point R(4) to screen
01B6	93 A4	
01B8	44 58 18	Load save page from screen via R(8)
01BB	94 FB 04 3A B8	until completed
01C0	98 FB 10 3A CB	Test save page number for limit
01C5	F8 EE 52 64 22	If limit reached, display EE
01CA	D3	and return to program select
01CB	98 52 64 22	If limit not reached, display page number
01CF	DB	and return to cursor routine

ADDR	CODE	COMMENTS
IX		
01D0	7B	Turn on Q
01D1	3F D1 6C 64 22	Load and display S.B.;
01D6	37 D3 AA	store S.B. in R(A).0 (cursor address)
01D9	7A DB	Turn off Q and return
X		
01DB	F8 8A A5	point R(5) to middle of (V.)
01DE	D5	and go there
XI		
01DF	7B	Turn on Q
01E0	3F E0 6C 64 22	Load and display S.B.;
01E5	37 E2 AE	Store S.B. in R(E).0 (to become P.C.)
01E8	7A DE	Turn off Q; go to somewhere on page 02
XII		
0200	F8 03 B6	Non overwriting border subroutine
0203	93 A6	
0205	F8 FF A4	
0208	84 56 16	
020B	86 FB 08 3A 08	
0210	06 52	
0212	F8 80 F1 56	
0216	86 FC 07 A6	
021A	06 52	
021C	F8 01 F1 56 16	
0221	86 FB F8 3A 10	
0226	84 56 16	
0229	86 3A 26	
022C	DB	

Q*BUG

Here are the details for eliminating more of the Super startup routines.

The nominal startup point for Super is address 0100. Here, a long branch is made to address 1800 where Super determines the end of your memory and stuffs it and other static data to work page 0000.

I already have suggested that you save work page 0000 as part of your Super master program. If you did this, the end of memory location and other data are included in your program and you do not need to go through the routine each time you start up Super.

The routine at address 1800 runs through address 183F and you can simply fill this portion of memory with "C4"s and hold it for use in the future.

The long branch at location 0100 is also no longer required and can be changed to "C4 C4 C4". Execution will now merely fall through to address 0103 which contains the long branch "C0 0F CD". This is the same address that the routine at address 1800 would branch to when it finished. Thus, you end up at the same point but have freed up 64 bytes of program space.

Incidentally, although the manual for Super states that address 0100 is the cold start point and address 0103 is the warm start point, this is not correct. As just explained, the only difference between starting at 0100 or 0103 is the bypassing of the routine at 1800. The actual cold or warm start address is contained in a routine on page 0100 which branches to the proper address in response to your answer to the "C/W?" prompt. I sincerely suggest that you DO NOT make any changes to the C/W? routine unless you are quite certain of what you are doing. A mistake here can really mess up Super.

Now, lets clear up an annoying little error in the way the command word "THEN" is printed. At present, THEN is treated as an operator and is coded to print a space before the word but WITHOUT a space after the word. To me, this looks rather odd when an IF/THEN line is printed.

A very simple correction to the first byte in the THEN command table entry will place the desired space after the word. The command table entry for THEN is presently:

Address: 06B4   25   54   48   45   CE   C5
                (T) (H) (E) (N)

Referring to the manual, you will find that the first byte in the entry table controls two things. The lower five bits determine the length of the word. Bit five (by the way, the bits are numbered from RIGHT to LEFT and start with bit #0), if a 1, prints a space before the word and bit six, if a 1, prints a space after the word.

In binary (or bit by bit), 25 looks like this:

BIT#	7	6	5	4	3	2	1	0
	0	0	1	0	0	1	0	1

As you can see, bit five is set to a 1 but bit six is a 0.

To get the trailing space, you merely need to change bit six to a 1. In binary, this would look like this:

BIT#	7	6	5	4	3	2	1	0
	0	1	1	0	0	1	0	1

Changing the leading byte in the command table at address 06B4 from 25 to 65 is all that is needed to remove this "BUG".

Now, a useful (I hope) program.

In many financial type programs written for the TRS 80 and other machines, a nifty command statement called "PRINT USING" is available. This statement formats the columnar output of strings so that the decimal points, for instance, all line up beneath each other when printed.

Unfortunately, Super does not offer such a command. We can, however, do this type of formatting with a simple Basic routine called by a GOSUB from the main program.

We will start with line #65000 in order to avoid any possible conflict with the regular program line numbers. First, at the point at which you wish to invoke the subroutine, you must insert a line such as:

(?)$ = X$(255): GOSUB 65000

This line will make X$(255) equal your original string (?)$. I suggest X$(255) to avoid any overlap with your regular program strings. Also, X$(255) takes the same memory space as X$(1). Of course, you can use any string you wish.

Line 65000 will establish the format of your string printout. For instance, if you want your printout to line up dollar amounts up to $1 million, you would enter:

65000   N$(255)="0,000,000.00"

Again, you are using a work string name which is your choice. Line 65010 will measure the length of the $1 million dollar amount:

65010   N255=LEN(N$(255))

Line 65020 will provide the space to be inserted in the final string to bring it up to the $1 million dollar length and line 65030 provides the "$":

65020   N$(253)=" "
65030   N$(254)="$"

Line 65040 will measure the length of X$(255), which was copied from the regular program string:

65040   N254=LEN(X$(255))

Line 65050 will test the length of both strings to check for the possibility that the regular program string (and thus X$(255)) is equal to or greater than $1 million already. If so, it will put the $ sign in front and return to the main program:

65050 IF N254>=N255 THEN
          X$(255)=N$(253)+X$(255):RETURN

Line 65060 will test the lengths of both strings and add a space in front of X$(255) if it is less than $1 million dollars and then return to line 65040 to reset the length of string X$(255) in the counter N254:

65060 IF N254<N255
       THEN X$(255)=N$(254)+X$(255):GOTO
       65040

The program will loop between line 65060 and 65040 until the length of both strings X$(255) and N$(255) are equal. At that point, line 65050 will pop on the dollar sign and return to the main program.

In the main program, immediately following the first inserted line, insert:

(?)$ = X$(255)

This will replace the original main program string with the new $1 million dollar formatted string. As an example, $ 98.76 should be reformatted and print as:

$ 98.76

This routine will also work with non-dollar amounts or just plain words. Remove the decimal point in N$(255) and do not input lines 65030 and 65050.

Will this routine work with variables? Not as it is now but it is possible to convert variables to strings in Basic. HOW??? Well, this is a good puzzler for you, the reader, to solve. Lets get your brains and pencils busy and have some reader input. All those who submit working solutions will be mentioned in a future QBUG column.

A recap of the reformatting program is:

MAIN PROGRAM – X$(255)=(?)$:GOSUB 65000

65000 N$(255)="0,000,000.00"
65010 N255=LEN(N$(255)
65020 N$(254)=" "
65030 N$(253)="$"
65040 N254=LEN(X$(255))
65050 IF N254>=N255 THEN
          X$(255)=N$(253)+X$(255):RETURN
65060 IF N254<N255 THEN
          X$(255)=N$(254)+X$(255):GOTO 65040

MAIN PROGRAM – (?)$=X$(255)

LOST·A·BYTE?

Jess Hillman

Ever hand-assembled a long program—by long I mean one that fills two or more pages of memory—and found that after a length of time "B's" start to look like "8's" or "3's"? Even quality reproductions of program listings tend to blur after a while.

Here's a comparatively short program that will enable your Elf to search through any length segment of memory, or all 65K bytes if you wish, for a specific byte. As written the program runs in page 00 of memory and is set up to examine the first 4K of memory.

The program will enable the user to double check all the memory locations where some troublesome code occurs. When entering the program in memory these are the key locations: 01, 04—high and low address bytes of the memory location where the byte search begins; 07, 0A—high and low address bytes of the ending memory location; 0D—this is the specific byte for which you are searching. The program uses register three as the current data pointer, register four to count the number of bytes checked and register six as a pointer to tell the program it's time to stop at a specific location. Register five contains the search byte, which is also stored on the stack beginning at location 90.

When you load the program in page 00 and run it, the data displays will almost immediately show "00" – the high address byte of the code for which it is searching. Press the input key and the display will change to show the low order address of the search byte. Press the input button once more and the program resumes its search, stepping through the specified memory area showing you first the high byte, then the low byte of the addresses where the search byte occurs.

Once the last byte of the memory segment has been examined (refer to program locations 1A-2D) the program offers you the option of searching for a different byte, turning on the Q LED and then looping (locations 40-52) until you key in the data and press the input key. Press input again, after the new search byte has been stored in location 000D and echoed to the data displays, and the process starts all over again.

I have successfully used this program to debug a Pilot interpreter I hand loaded into my system. Perhaps you will find as good a use for it, also.

ADDR	CODE	OPCODE	OPERAND	COMMENT
00	F8 00	LDI	#00	Set R3 as starting
02	B3	PHI	R3	address pointer
03	F8 00	LDI	#00	
05	A3	PLO	R3	
06	F8 0F	LDI	#0F	Set up R3 as end-
08	B4	PHI	R4	ing addr. pointer
09	F8 FF	LDI	#FF	
0B	A4	PLO	R4	
0C	F8 XX	LDI	#XX	Put sought after
0E	A5	PLO	R5	byte in loc 0D and register R5.0
0F	93	GHI	R3	Copy starting
10	B6	PHI	R6	address pointer
11	83	GLO	R3	into R6.
12	A6	PLO	R6	
13	F8 90	LDI	#90	Set stack at loc.
15	A2	PLO	R2	90 of current page
16	E2	SEX	R2	R2=X
17	85	GLO	R5	Start search by
18	52	STR	R2	getting the lost byte and putting it on the stack.
19	43	LDA	R3	Load byte into D
1A	F3	XOR		Comp needed byte
1B	32 2E	BZ	#2E	Jump if a match
1D	24	DEC	R4	Else count down
1E	94	GHI	R4	one location and
1F	52	STR	R2	see if it's time
20	96	GHI	R6	to stop.
21	F3	XOR		Checking last mem page.
22	CE	LSZ		If so, skip.
23	30 13	BR	#13	Else get next byte
25	84	GLO	R4	On last page, chk
26	52	STR	R2	for last mem loc.
27	86	GLO	R6	
28	F3	XOR		Last byte checked?
29	CE	LSZ		If so, skip.
2A	30 13	BR	#13	Else get next byte
2C	30 40	BR	#40	Go get next byte
2E	93	GHI	R3	to look for match!
2F	52	STR	R2	Get high address and store it
30	64	OUT	4	and show it
31	3F 31	BN4	#31	Wait
33	37 33	B4	#33	Until input press
35	83	GLO	R3	Get low address
36	FF 01	SMI	#01	Adjust for proper
38	52	STR	R2	address and store
39	64	OUT	4	it and show it
3A	3F 3A	BN4	#3A	Wait for input key
3C	37 3C	B4	#3C	to be pressed
3E	30 1D	BR	#1D	Continue searching
40	7B	SEQ		Turn on Q, get rdy
41	F8 00	LDI	#00	for new byte to
43	BF	PHI	RF	search for by
44	F8 0D	LDI	#0D	loading address of
46	AF	PLO	RF	loc in main prog into RF and making it the stack ptr.
47	EF	SEX	RF	X=F
48	3F 48	BN4	#48	Wait for input key
4A	37 4A	B4	#4A	pressed
4C	6C	INP	4	Get new byte and
4D	64	OUT	4	echo it
4E	3F 4E	BN4	#4E	Wait for input key
50	37 50	B4	#50	pressed
52	30 00	BR	#00	Start looking for the new byte.

INDEX TO VOLUME 2

BASIC, SUPER

"Hamurabi Game"
Fred Hannan   #10 pg. 10

"Membership"
E.H. Sandelin   #11 pg. 9-10

"Q Bug"
Fred Hannan   #12 pg. 5-6

"Rotating Christmas Display"
Fred Hannan   #11 pg. 11-12

"Using Super Basic"
Ron Cenker   #2 pg. 6-7

BASIC, TINY

"Biorhythm"
Gary Gehlohoff   #4 pg. 8-9

"A Cool Display"
William Carnes   #2 pg. 3

"Elf II Patches for TB TTY IF"
Chuck Reid   #2 pg. 11

"Patch Sort Function for Elf II
Tiny Basic"
Chuck Reid   #1 pg. 11

"Tiny Basic Music"
Richard Warner   #3 pg. 12

"Tiny Basic Screen Clear"
Paul W. Morris   #9 pg. 5

"Tiny Basic Strings"
David Taylor   #8 pg. 14

GAMES

"Beat the Machine"
Mike O'Rourke   #5 pg. 7-8

"Biorhythm"
Gary Gehlhoff   #4 pg. 8-9

"Mods to Biorhythm"
Gary Gehlhoff   #9 pg. 9

"Blockade"
Kevin Cutts   #1 pg. 6-9

"Cosmac Cowboy"
Gary Gehlhoff   #5 pg. 9-11

"Elf Die"
Ron Zoscak   #3 pg. 1-2

"ESP"
Jess Hillman   #7 pg. 5-6

"Fifteen Puzzle, The"
Ray Tully   #1 pg. 1-5

"Fifteen Puzzle for Elf II"
Robert V. Dipippo   #2 pg. 11

"Hamurabi Game"
Fred Hannan   #10 pg. 10

"Nim"
Richard Moffie   #3 pg. 3-4

"One-Armed Bandit"   #7 pg. 1-5
L. Owen

"Put Life Into Your Computer"
Ray Tully   #4 pg. 1-5

"Saucer Squasher"
Mark Bitting   #11 pg. 1-3

"STIX"
Vic Worthington   #10 pg. 1-4

MATHEMATICS

"BCD to Binary Converter"
Al Williams   #2 pg. 10

"Hex to Decimal"
Paul J. Grech   #3 pg. 7

"PNR Calculator"
Ron Zoscak   #1 pg. 12-13

"The Sieve of Eratosthenes"
Phillip B. Liescheski III   #2 pg. 4-5

MUSIC

"Advanced Organ"
Nick Williams   #8 pg. 13

"Auld Lang Syne"
Paul Thompson   #6 pg. 9

"God Rest Ye Merry Gentlemen"
Gerald VanHorn   #6 pg. 8

"Good King Wenceslaus"   #6 pg. 9
Paul Thompson

"Harmonious Sequencer"
Don Stevens   #2 pg. 8-10

"Jolly Old St. Nicholas"
Paul Thompson   #6 pg. 9

"Joy to the World"
Gerald VanHorn   #6 pg. 8

"Music for Expanded Memories"
Allan Armstrong   #1 pg. 10-11

"Music to March By"
Dan Van Dyke   #7 pg. 2

"O Christmas Tree"   #6 pg. 9
Paul Thompson

"Silent Night"
Jan Beckman   #6 pg. 8

"Sonatina, Albert Biehl"
Ian Beer   #3 pg. 11

MISCELLANEOUS

CONTRIBUTING AUTHORS

Allan Armstrong	San Francisco, CA
Van C. Baker	St. Petersburg, FLA
Jan Beckman	Ashtead, Surrey ENGLAND
Ian Beer	Seattle, WA
Mark Bitting	Lemoyne, PA
William Carnes	Marlboro, MA
David Cartier	Lanham, MD
Ron Cenker	Coplay, PA
Stephen P. Clark	Tallahassee, FL
David Crawford	North Attleboro, MA
Kevin Cutts	Steamwood, ILL
Hugh Dagg	Waterloo, Ontario CANADA
Robert DiPippo	W. Warwick, RI
Ivan Dzombak	Latrobe, PA
Gary Gehlhoff	Oswego, NY
Paul Grech	Brampton, Ontario CANADA
John Guarini	Ocean, NJ
Lester Hands	Sylvania, OH
Fred Hannan	E. Lyme, CT
Jess Hillman	Columbia, MS
Richard Johnson	Shorewood, ILL
Enos Jones	Dallas, TX
Jeff Jones	Trafford, PA
Jack Krammer	N. Massapequa, NY
Bobby Lewis	Waterford, CT
Phillip Liescheski III	Houston, TX
Ken Mantei	San Bernardino, CA
Richard Moffie	Canoga Park, CA
Paul W. Morris	Winter Park, FL
L. Owen	Trenton, Ontario CANADA
Gary Price	Sunnyvale, CA
Stephen Rarick	N. Adams, MA
Chuck Reid	Sarina, Ontario CANADA
E.H. Sandelin	Portland, ME
Don Stevens	New York, NY
David Taylor	APO San Francisco, CA
Paul Thomson	Lombard, ILL
Ray Tully	Baton Rouge, LA
Dan Van Dyke	Healdsburg, CA
Gerald Van Horn	Junction City, OR
Richard Warner	W. Covina, CA
Mark Wendell	Los Angeles, CA
Al Williams,	Bay St. Louis, MS
Nick Williams	San Francisco, CA
Vic Worthington	Sundance, WY
Ron Zoscak	Pittsburgh, PA

ROTATE & FLIP

by
Bill Carnes

One of the nice features of the COSMAC microprocessor is the low cost graphics made possible by the 1861 video chip. Graphic patterns are not difficult to write and are easily manipulated by software. The following program is an aid in creating interesting graphic effects. The program takes a 256 byte page of memory and either rotates it showing the "other side" or flips the pattern over diagonally depending upon the state of Q.

The program is ROMable and page relocatable, and needs at least 3/4K of memory to work. The Rotate routine starts at Hex address 08 and the Flip routine starts at 09. This allows room for branches and a stack. The program will accept input of a source page and then a destination page. When finished, the routine displays the result on your screen or displays any page entered on the Hex keypad. Before running the program, a pattern must already be entered into a page of memory. For example, see QUESTDATA No. 8 for Jay Mallin's DOODLE PROGRAM that makes loading a pattern easy.

Mirror images can be easily made by making the destination page the same as the source page. With appropriate combinations of the two routines, many images can be made. A picture may be quartered, halved, or folded along a diagonal. Another application of this program is to encode messages or programs. As long as the destination is different from the source, these routines are non-destructive and reversible.

The program works as follows. The Q line is set if the entry point is 08. Next, the program counter is set to r(3). (For some systems, location 09 may be changed to "93"). The stack is set to r(2), and the subroutines are set next. The first subroutine inputs a Hex byte. The second subroutine takes a source byte, reverses the order of the bits in the byte and stores the byte at the destination. The program now obtains the source and destination, clears the low order byte of the source register, and if Q is set, it branches to the Rotate routine.

The Flip routine begins by taking the byte at the beginning of the source page, reverses the bits, and stores the result at the end of the destination page. The destination register is decremented and the source register is incremented until the routine is done.

The Rotate routine takes a byte at the beginning of the source page, reverses the bits, and stores the result at the right side position of the TV screen on the destination page. The source register is incremented, and the destination register is decremented until the row is done. The routine then does the next row. When completed with all the rows, the routine branches to the video display routine.

It is not necessary for you to count the machine cycles between an interrupt and DMA, since the 00-Idle instruction takes up any slack. The same instruction slows down the Hex keypad sampling where high speed is not needed. If your machine states are decoded and shown on LEDs, you will notice that the LED for Fetch is dimmer, but you will also notice that your graphics capability is brighter!

Registers Used:

```
        0 = PC entry and DMA
        1 = Interrupt
        2 = Stack
        3 = Main PC
        4 = Bit count
        5 = Row count
        7 = Source pointer
        8 = Destination pointer
      9.0 = Original byte holding register
      9.1 = Reversed byte holding register
        A = Subroutine Get character
        B = Subroutine Reverse byte
```

ADDR	CODE	COMMENT
0000	30 08/09	Branch to program
0002	-- -- --	Free space
0005	00 00 00	Stack area
0008	7B	SEQ, ROTATE entry
0009	90 B3 BA BB	Set registers, FLIP entry
000B	F8 00 B2	Set stack
0010	F8 07 A2	to 0007
0013	E2	SEX X=2
0014	F8 18 A3 D3	Set PC to r(3)
0018	F8 49 AA	Set "Get char." sub.
001B	F8 53 AB	Set "Reverse byte" sub.
001E	DA B7	Get source page
0020	DA B8	Get destination page
0022	F8 00 A7	Clear r(7.0)-source
0025	31 31	BQ to ROTATE
0027	F8 FF A8	Set r(8.0)-destination
002A	DB	Reverse byte
002B	28	DEC destination
002C	87 3A 2A	Branch if not done
002F	30 68	Branch to video display
0031	7A	ROTATE entry, REQ
0032	F8 08 A8	Set r(8.0)-destination
0035	F8 08 A5	Set row count
0038	28	DEC destination
0039	DB	Reverse byte
003A	25	DEC row count
003B	85 3A 38	Branch if row not done

ADDR	CODE	COMMENT
003E	88 FC OF A8	Add 10 Hex
0042	18	and allow carry-over
0043	87 3A 35	Branch if not done
0046	30 68	Branch to video display
0048	D3	Return
0049	6C 64 22	Input and display char.
004C	3F 49	Loop while in is high
004E	37 4E	Loop while in is low
0050	30 48	Go to Return
0052	D3	Return
0053	47 A9	Get byte, r(7)+1, save
0055	F8 08 A4	Set count for bits
0058	89 FE A9	Left shift byte
005B	99 76 B9	Right shift carry
005E	24	DEC bit count
005F	84 3A 58	Branch if not done

ADDR	CODE	COMMENT
0062	99 58	Store byte at destination
0064	30 52	Go to Return
0068	F8 77 A1	Set interrupt
006B	93 B1 69	Set int. turn on
006E	6C 64 22 B0	Display keyboard
0072	00 30 6E	Wait, do it again
0075	72 70	Return from interrupt
0077	22 78	Save "T"
0079	22 52	Save "D"
007B	F8 00 A0	Clear R(0.0)
007E	80 00	Wait
0080	E2 20 A0	DMA
0083	E2 20 A0	DMA
0086	E2 20 A0	DMA
0089	3C 7E	Done?
008B	30 75	Go to Return

TAKE TWO ASPIRIN

by
Schramke

"Take two aspirin and be sure not to call me in the morning!"

"Darn! If they print a great program like that in Questdata one more time and I'll kill myself! How could it be that they write one program starting at page 01, which I have, and then turn around and write another at page 04, which I don't have?"

"Well, what can be done?"

Actually we have two choices: we can write a letter to ANSI and ask them to put standards on programs for the ELF, much as they did for the major languages like COBOL; or we could, voluntarily, "police our own". The latter is probably the best, and easiest way.

Take for instance the great program which appeared in the 7th issue of Questdata (Vol. 1) written by Edgar Garcia. I, for one, was very excited when I saw this program, however, I quickly realized that, where Mr. Garcia used pages 00–06, I only had 04–0B. No problem. I got out a pencil and began to weave in and out of machine code, trying to find all occurences of page initilization. Once I felt I had found them all I began to load the program into memory, and quickly saved it on tape, lest it just disappear. You don't have to watch too much TV to realize that the unexpected is usually the most expected occurences. Well, as I'm sure you've guessed, the program blew up and — well two more times and each time the same thing happened. At that, you can imagine the headache I had.

I did as they said, and took a couple aspirin, and quickly retired for the night. In the morning I had a brand new perspective on the matter. With all respect to Mr. Garcia, instead of "weeding" through the program again, I wanted to find out where Mr. Garcia resided and spray paint the word "ELF" across his windows. Once I had grasped hold of myself I realized that there was one more thing I could do. I went to my junk box, pulled out a few parts and re-addressed my memory to Mr. Garcia's specifications. I loaded the program, and, sure enough, the program worked very well.

I began to feel a bit upset that I would have to re-address memory every time I wanted to use a program which just didn't conform to my hardware specifications. I knew that there had to be another way. I sat down and made a list of all the things I felt that could cure this problem that I'm sure more than myself have encountered. I've taken the liberty to submit ths list, partially condensed and simplified, to Questdata in hope that at least one person would take heart and give us stubborn hobbyists a break. Such could be used as, well, the outline for a standard. Here it is:

1. Omit cute little tricks when addressing the high-order byte of a register whose purpose is to point to a memory location. Take for instance the person who needs to clear a register (as a counter or whatever) and does it on page 00 using a sequence like — 90 A8. You know what happens when a person then wants to locate the program at page 01; the counter, instead of being cleared, is initialized to 01.

2. Always document programs, no matter how small or how large. If you want people to use your ideas, first they have to know what you did. Along the same lines, never list just machine code. Going through machine code is a great pain. Use standard RCA assembly language along with, or instead of machine language. This makes the program not only easier to add to and read, but it also allows another person to go through your program quickly without having to contemplate machine code.

3. Make a seperate table, along with the program, which states at what location a page number is initialized, and give the page number a symbolic name, such as DISP for a display page, TAB for a table page, etc. This allows a person to go through and change the required page numbers without being restricted to your absolute addressing.

4. Not all ELF's use the same IO ports or EF flags, so, the same should be done for IO statements and branches on EF condition codes. Make a table, along with symbolic names, for all instructions which deal with the outside world. Remember, what may seem as a standard (like 6C for the keyboard or toggles), may not be a standard to another person.

5. As Mr. Garcia did, separate all subroutines and data tables. This allows another person to take certain subroutines you have written, and possibly use them in a program of their own.

These 5 suggestions could not possibly cover all cases and make up a standard, but it is my hope that it would serve as the beginning of an interaction between users. Such an interaction could not hurt and may save you quite an aspirin bill. I would encourage more people to write in and offer suggestions, along the lines of mine, which may help to make it easier to relocate programs. I totally agree with the statement made by Mr. Larry R. Baysinger on page 6 of the 5th issue of this newsletter. Conventions need to be established now, before things get too far out of hand, if they haven't already.

Before closing, I would like to apologize to Mr. Garcia for the "picking-apart" I have done to his article. I have done so for two reasons. First of all, it is the first of the really big programs to appear in Questdata, and therefor was a large candidate to be "unruly". Secondly, because it is one of the best programs I have seen to date and I felt that the documentation did not do justice to the hard job he had. I feel that this lack of documentation may have scared some people away from writing larger programs due to the way M. Garcia's looked so complex without documentation. I feel that, with the proper documentation, nearly any program, no matter how long, can be made simple.

ADDRESS CORRECTION REQUESTED