Questdata

Volume 1 Issue #8 ©

WHAT THE MACHINE IS THINKING

"I know what you're thinking about," said Tweedledum; "but it isn't so, nohow."
"Contrariwise," continued Tweedledee, "if it was so, it might be; and if it were so, it would be; but as it isn't, it ain't. That's logic."
–ALICE IN WONDERLAND

WILD electrons tamed and commanded to jump through hoops and perform all manner of tricks. Fun just to think about the goings on within the COSMAC microprocessor. What mad pursuit? What struggle to escape? What pipes and timbrels? What wild electron ecstasy? What paraphrasing of Keats? Electrons herded about through logic gates and corralled into little pens (registers) and put in little boxes with addresses on them (memory).

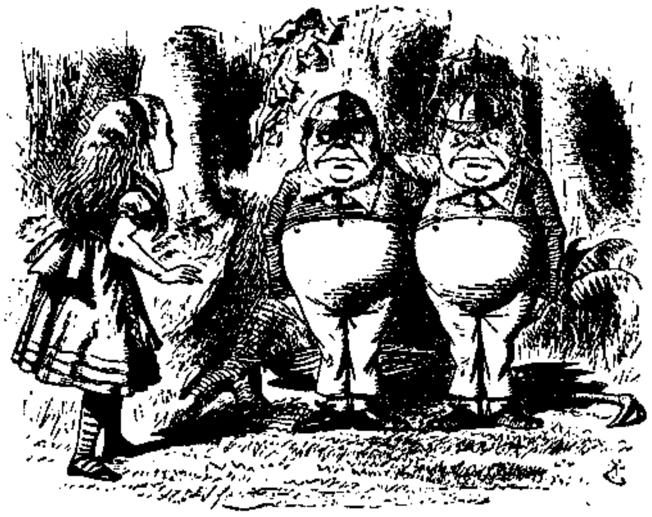
Wiring that can configure and reconfigure. Microprocessors have to be one of mankind's and technology's finest wonders.

Knowledge about our little computers does not come all at once. Programming creeps up on you until something inside your head clicks and you say aha! If things have not clicked together for you yet, they will; it is a matter of reading everything about microcomputers in sight, talking to other hobbyists, reading COSMAC code listing, and experimenting a lot.

Machine and assembly language are rewarding and are the only languages where you will feel the full power of the computer. If you learn a language like BASIC, you will always feel that there is this "something" you don't know. Indeed, the language of BASIC itself is written with a lot of creative machine code. Machine language will always take up the least amount of memory. If you know machine code, you are among the elite of the microcomputer hobbyists.

Today we are going to discover how a language like BASIC knows how to recognize words like LET. Before we proceed, let's review the AND and OR instructions, and add two new ones XRI and SHIFT:

- The AND instruction can turn a bit or bits to zero without affecting the other bits in a byte. It can test an individual bit or turn many bits to zero. This is done by constructing a MASK (programmer word for the immediate byte), the 0's in the MASK will block out (turn to 0) the byte in the D(starting) register.
- The inclusive OR instruction will add a bit or bits to a byte without touching any of the other bits in the byte. The MASK in the immediate byte consists of 1's which turn the bit positions in which they are entered into 1's. A 0 in the OR MASK will not affect the bit or bits in the same bit position in the D(s) byte.
- The eXclusive OR instruction can be used to turn a byte into its complement: 1100 0101 becomes 0011 1010 when the XRI byte is FF (1111 1111). The XRI instruction can also be used to compare logical and make decisions based upon the outcome. This is done by using as a MASK for the value you wish to test. If you want to test whether 4C has entered your programming net, then 4C or 0100 1100 is your MASK.
- SHIFT (Right, Left—with and without ring shifting) is able to bounce things off to oblivion (to the right is divide by 2 and to the left is multiply by 2). Or the bits can make a complete circle with a bit first being brought into the DF (data flag). By entering a 0 in the highest bit position and executing a SHLC, the DF is cleared to 0 and whatever was in the DF becomes the lowest bit position.


From left to right: Alice, Tweedledee and Tweedledum or is it vice versa?

More details on these last two items and examples will be given on the next three pages.

In the last issue of QUESTDATA, recall that the logic instructions perform their operations upon the individual bits of a byte. In the case of an Inclusive OR (RCA just calls this a regular OR), we learned that if one or the other of the bits in the ORing operation is a one then the result is a one. The only time the result D(f) is a zero (0), is when the immediate (data mask) and the D(s) starting value are 0's. The Inclusive OR is consistent with our semantic expectation of the word OR. When A OR B is true then C is true. If C represents "is a chess player" and C is true statement, then we know that either Andy or Bill or both are chess players.

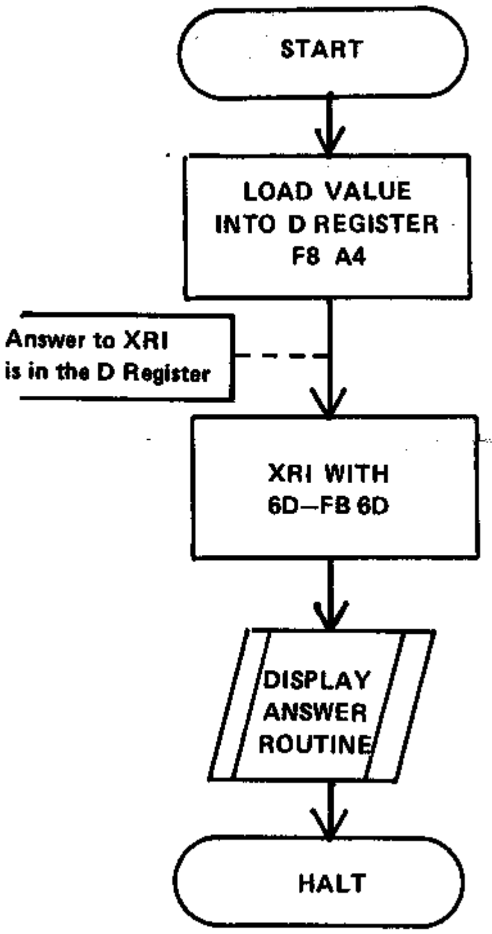
The eXclusive OR (XRI in this case) puts a condition upon the Inclusive OR. It says, "the BOTH ON (1's) condition will cause a ZERO (0) in the final D(f) result." Repeating the "chess player" example with the eXclusive OR, we find: If we know that C (is a chess player is true) then we know Andy OR Bill, BUT NOT BOTH, is a chess player.

XRI LOGIC TABLE

Immediate	D(s)	D(f)
0	0	0
0	1	1
1	0	1
1	1	0

This semantic handle on the logic situation of ORI (OR Immediate) and XRI (eXclusive Or Immediate) are to give you a hand in memorizing the logic tables of the bit values. The usefulness of the XRI instruction becomes apparent when you want to COMPARE and MATCH numeric or alphabetic values. It is the XRI concept which allows the computer to "know" and display the word "LET" in the language BASIC. Try to figure out what the result of XRIing A4 and 6D will be. First translate these two hex numbers into binary 1's and O's using the handy and speedy table on page 3. Then translate these values back into hex once you have found the answer. Hopefully, your results match the COSMAC machines.

What would be the result of XRIing 0011 0110 and 1100 1100? What about 0011 0010 XRIed with 0011 0010? Right, what you have found by putting these last two values into the laboratory experimenter is exactly the property which lets you discover when the number 2 in ASCII code has been "found." You got it, a match between two like binary codes will yield 0000 0000 and we have an easy way to test for this condition—good old BNZ (3A hex) and BZ (32 hex). Last lesson we touched upon the fact that XRIing with FF will give the complement (or in other words—turn your zero's to one's and vice versa). Indeed, the XRI is a nice instruction to have around.


START
LOAD VALUE INTO D REGISTER F8 A4
Answer to XRI is in the D Register
XRI WITH 6D—FB 6D
DISPLAY ANSWER ROUTINE
HALT

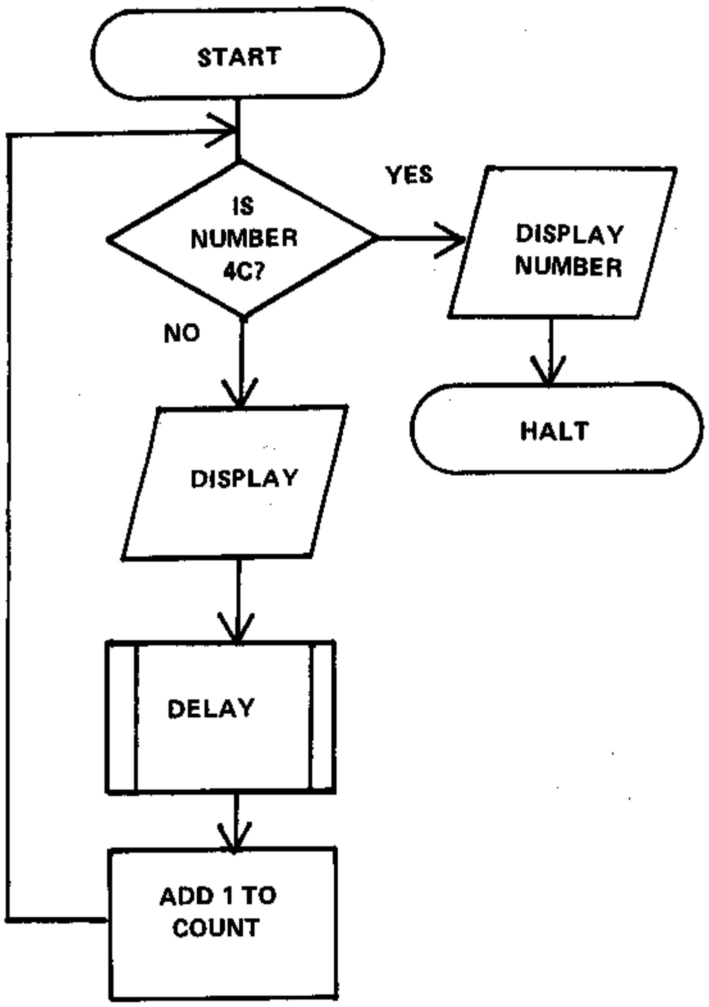
XRI EXPERIMENT

LOC.	CODE	MNEM.	ACTION
00	F8	LDI	Load Starting
→ 01	A4		D(s)
02	FB	XRI	functions: [FB], [F9], [FA]
→ 03	6D		Immediate MASK data
04	C4	NOP	NOP's to be used for
05	C4	NOP	sequential logic if you wish
06	C4	NOP	
07	C4	NOP	
08	A4	PLO4	Save answer in R4.0
09	90	GHI0	Initialize R5.1
0A	B5	PHI5	R5.1=00 if page 0
0B	F8	LDI	Initialize pointer to ans.
0C	13		
0D	A5	PLO5	thus ans. to appear loc. 13
0E	E5	SEX5	Set the X Register to 5
0F	84	GLO4	put answer back in D reg.
10	55	STR5	Store ans. via 5 pointer
11	64	OUT 4	Display what X is pointing
12	00	IDL	this 00 tells the 1802 to HALT
13	00		ANSWER FOR VIP LOOKUP

The experiments have been arranged so that the first part is the only thing you need to be concerned with. The other half is for display. In the case of the "L" sieve (hex 4C is "L") there is a delay and hex number generator section which you don't really need to understand completely. The NUMBER SIEVE lets you see what is happening while you find "4C." Put "57" in location 0003. Now you have found the letter "W." The delay section of the code is familiar from other earlier experiments. Try speeding up the delay to a count of only 10 (hex) instead of 1000 as it is now. By putting AE or PLO in location 000B and 8E in location 000D, you see just how fast the sieve can find something—the blink of an eye is pretty fast. If you could point to a hex number as fast as the COSMAC, you could get a job as a computer.

Sequential logic lets you perform more than one

SIEVE FLOWCHART



XRI SIEVE FOR 4C HEX

```
  LOC.   CODE   MNEM.     COMMENTS
  0000    30     BR        Branch around logic
  0001    12               for Initialization
  0002    FB     XRI       Exclusive OR Immediate to
→ 0003    4C               Check for HEX value 4C
  0004    C6     LSNZ      If D≠00 then  Display & cont.
  0005    64     OUT 4     otherwise Display and HALT
  0006    00     IDL       HALT finish of 4C sieve
  0007    64     OUT 4     Display & continue
  0008    27     DEC R7    Restore X pointer to Display
  0009    F8     LDI       Load DELAY routine
  000A    10               Amount of DELAY
  000B    BE     PHI RE    is put in R(E).1
  000C    2E     DEC RE    decrement until 00
  000D    9E     GHI RE    R(E).1 into D
  000E    3A     BNZ       Proceed if DELAY is up
  000F    0C               repeat if DELAY not up
  0010    30     BR        Branch around Intiialization
  0011    18
  0012    90     GHI R0    Initialization is done here so
  0013    A4     PLO R4    that it is easier to get to XRI
  0014    B7     PHI R7    Laboratory LOGIC
  0015    F8     LDI       Load pointer to work area
  0016    1E               Loc. 1E HEX
  0017    A7     PLO R7    The 1E goes into R(7).0
  0018    E7     SEX R7    Set output pointer
  0019    84     GLO R4    Put count in D
  001A    57     STR R7    Store count in work area
  001B    14     INC R4    Increment count
  001C    30     BR        GOTO–LOGIC TEST
  001D    02
  001E    ANY              This is DISPLAY work area
```

BINARY		HEX
0 0 0 0	=	0
0 0 0 1	=	1
0 0 1 0	=	2
0 0 1 1	=	3
0 1 0 0	=	4
0 1 0 1	=	5
0 1 1 0	=	6
0 1 1 1	=	7
1 0 0 0	=	8
1 0 0 1	=	9
1 0 1 0	=	A
1 0 1 1	=	B
1 1 0 0	=	C
1 1 0 1	=	D
1 1 1 0	=	E
1 1 1 1	=	F

Binary and Hexadecimal

operation on a number. Often you will want to add a bit and then COMPARE it. Try to figure out what value of XRI (FB is the code for XRI) will turn the D register to zero when it comes to locations 06 and 07. If you can figure it out before you run the program you get 3 extra points.

SEQUENTIAL LOGIC

LOC.	CODE	MNEM.	ACTION
00	F8	LDI	Load Starting
01	32	D(s)	
02	F9	ORI	functions: [F9], [FB], [FA]
03	41		Immediate MASK data
04	FB	XRI	Complement value using FF
05	FF		with the eXclusive OR Immed.
06	C4	NOP	What XRI Sieve value will turn
→ 07	C4	NOP	D Register to 00 at this point?
08	A4	PLO4	Save answer in R4.0
09	90	GHI0	Initialize R5.1
0A	B5	PHI5	R5.1=00 if page 0
0B	F8	LDI	Initialize pointer to ans.
0C	13		
0D	A5	PLO5	thus ans. to appear loc. 13
0E	E5	SEX5	Set the X Register to 5
0F	84	GLO4	put answer back in D reg.
10	55	STR5	Store ans. via 5 pointer
11	64	OUT 4	Display what X is pointing to
12	00	IDL	this 00 tells 1802 to HALT
13	00		ANSWER FOR VIP LOOKUP

The instruction F8 puts the very next thing in the D register and F6 shifts the good old D register one bit to the right. The SHIFT instructions are pretty straightforward. They bounce the bits to the right (or left) one place each time they are executed. Thus, 1111 1111 becomes 0111 1111 after one shift right. Notice how things bounce off to oblivion—shifting to the right is the equivalent of dividing by two. Shifting to the left is multiplying by two. This is used and mentioned by Mike Tyborski in his NEW PATTERNS article in this issue.

The SHIFT with CARRY is very similar to the shifts without this' feature EXCEPT that there is a CARRY LINK [ ]. Thus, if you shift 1111 1111 by executing one 76, you will get ?111 111 [1]. The brackets represent the CARRY LINK. The question mark indicates that unless you know the value of the link (which can be either a 0 or 1 upon startup) you can not fill in this value. If you don't believe in this carry link, try loading F8 00 76 C4 C4 C4, etc. and runing it after you have run the 1111 1111 shift you tried. Surprise 80 is your result. The link was loaded by the 1111 1111 and has circled around and come up in the high order bit position. In the Shift left with CARRY this bit will be found in the lowest order bit position. You have also discovered that the CARRY LINK is not cleared to zero upon startup! Only I, N, X, P, Q, and R(0) are initialized to zero on startup. How can you make sure the link is 0? F8 00 76, out to do it.

SHIFT RIGHT

LOC.	CODE	MNEM.	ACTION
00	F8	LDI	Load Starting
→ 01	0F		VALUE
02	F6	SHR	Shift Right One Bit
03	F6	SHR	Another SHR
04	C4	NOP	NOP's so same logic
05	C4	NOP	can be used for
06	C4	NOP	all experiments
07	C4	NOP	
08	A4	PLO4	Save answer in R4.0
09	90	GHI0	Initialize R5.1
0A	B5	PHI5	R5.1=00 if page 0
0B	F8	LDI	Initialize pointer to ans.
0C	13		
0D	A5	PLO5	thus ans. to appear loc. 13
0E	E5	SEX5	Set the X Register to 5
0F	84	GLO4	put answer back in D reg.
10	55	STR5	Store ans. via 5 pointer
11	64	OUT 4	Display what X is pointing to
12	00	IDL	this 00 tells the 1802 to HALT
13	00		ANSWER FOR VIP LOOKUP

SHIFT RIGHT with CARRY

LOC.	CODE	MNEM.	ACTION
00	F8	LDI	Load Starting
→ 01	0F		VALUE
02	76	SHRC	Shift Right One Bit with CARRY
03	76	SHRC	Another SHRC
04	C4	NOP	NOP's so same logic
05	C4	NOP	can be used for
06	C4	NOP	all experiments
07	C4	NOP	
08	A4	PLO4	Save answer in R4.0
09	90	GHI0	Initialize R5.1
0A	B5	PHI5	R5.1=00 if page 0
0B	F8	LDI	Initialize pointer to ans.
0C	13		
0D	A5	PLO5	thus ans. to appear loc. 13
0E	E5	SEX5	Set the X Register to 5
0F	84	GLO4	put answer back in D reg.
10	55	STR5	Store ans. via 5 pointer
11	64	OUT 4	Display what X is pointing to
12	00	IDL	this 00 tells the 1802 to HALT
13	00		ANSWER FOR VIP LOOKUP

Back issues of QUESTDATA are available at $1.50 each from Quest Electronics. Programming skills are cumulative.

MULTIPLE PRECISION MULTIPLICATION

By Ivan Dzombak

This program provides multiplication of two 2-byte (16 bit) numbers to get a 32 bit result. Note the problem below:

```
                11011010
               x10111001
               ---------
                11011010
               00000000
              00000000
             11011010
            11011010
           11011010
          11011010
         00000000
        11011010
        ---------
      1001110110001010
```

A few observations can be made:

- The product is the sum of the partial products.
- Each digit in the multiplier has a corresponding partial product. This is equal to the operand if the multiplier bit is 1, equal to zero if the multiplier bit is zero.
- From bottom to top, the partial products are shifted one place to the right with respect to the one below it.
- The number of bits in the product is equal to the sum of the number of bits in the operand (Ex. Multiplying two 8 bit numbers yields a 16 bit result).

In a microprocessor, it is easier to add the sums of the partial products as they are formed from bottom to top. In my program, I have set up three "software registers" as follows:

00F8	ACCUMULATOR	00FB
00F0	OPERAND	00F3
00E8	MULTIPLIER	00EB

We will use ACCUMULATOR to hold the sum of the partial products, OPERAND to hold the number to operate on, and MULTIPLIER to contain the number that OPERAND is to be multiplied by. Before adding a partial product to ACCUMULATOR, we must shift ACCUMULATOR left one bit; this gives the same effect as shifting each partial product right before adding it to ACCUMULATOR. The procedure for integer multiplication is:

- Clear ACCUMULATOR.
- Shift MULTIPLIER and ACCUMULATOR one place left; if bit shifted out of MULTIPLIER is 1 (DF=1), then add OPERAND to ACCUMULATOR.
- LOOP 32 times.

In this program, you must first enter the low order byte (LSB) of the operand, then the high order byte (MSB). After this, the Q LED will turn on to signify that the computer is waiting for a multiplier. Enter it in the same fashion as the operand. Repeated depressions of INPUT will cause the product to be displayed, from the low order byte to the high order byte. For a more practical application, decimal to hex and hex to decimal subroutines could be added, but I did not include them in order to retain the 256 byte limit. This program should run in expanded systems, due to the initialization of the high order pointer bytes.

[Nore: This program starts at memory location 003A to facilitate the use of the Super Elf monitor when loading.]

[Use Monitor then enter 30 in location 0000 and 3A in location 0001 to jump to the start of program]

```
LOC.   DATA   MNEMONIC   REMARKS
003A   F8     LDI        Initialization of pointers–
003B   F0                load F0 into R(1).0  (OPERAND)
003C   A1     PLO R1
003D   F8     LDI        load E8 into R(2).0  (MULTIPLIER)
003E   E8
003F   A2     PLO R2
0040   F8     LDI        load F8 into R(3).0  (ACCUMULATOR)
0041   F8
0042   A3     PLO R3
0043   F8     LDI        Clear ACCUMULATOR BY WRITING
0044   00                00 into M(00F8)-M(00FB)
0045   53     STR R3
0046   13     INC R3
0047   53     STR R3
0048   13     INC R3
0049   53     STR R3
004A   13     INC R3
004B   53     STR R3
```

```
LOC.   DATA         MNEMONIC        REMARKS
004C   B1 B2 B3     PHI R1, R2, R3  Initialization of High order bytes
004F   B4 B5                        of registers
0051   F8           LDI             reset ACCUMULATOR pointer
0052   F8
0053   A3           PLO R3
0054   E1           set X=1
0055   C8           LSKP            skip next two control bytes (for input control)
0056   7B           SEQ             This SEQ is a flag for both OPER. & MULT.
0057   E2           SEX R2          Points the input bytes to MULTIPLIER
0058   F8           LDI             loop twice on each operand
0059   02
005A   A5           PLO R5
005B   3F           BN4             Wait for INPUT pressed
005C   5B
005D   37           B4              Wait for INPUT released
005E   5D
005F   6C           INP 4           input byte from keyboard
0060   64           OUT 4           display and increment pointer
0061   25           DEC R5          decrement register 5 & load into D; if
0062   85           GLO R5          D≠0 then loop back for more input
0063   3A           BNZ
0064   5B
0065   F8           LDI             load D with 00
0066   00
0067   31           BQ              GOTO M(6E) if Q=1
0068   6E
0069   51           STR R1          Write 00 into two high order bytes of
006A   11           INC R1          of OPERAND
006B   51           STR R1
006C   30           BR              go back to input loop at M(56) for
006D   56                           MULTIPLIER
006E   52           STR R2          Write 00 into two high order bytes
006F   12           INC R2          of MULTIPLIER
0070   52           STR R2
0071   39           BNQ             GOTO M(56) if Q=0
0072   56           STR R6
0073   F8           LDI             Reset pointers
0074   F0
0075   A1           PLO R1
0076   F8           LDI
0077   E8
0078   A2           PLO R2
0079   7A           REQ             Turn Q off
007A   F8           LDI             Set counter for the 32 loops
007B   20
007C   A4           PLO R4
007D   03 FE 53 13                  Shift ACCUMULATOR one place left
0081   03 7E 53 13
0085   03 7E 53 13
0089   03 7E 53 13
008D   F8           LDI             Reset pointer
008E   F8
008F   A3           PLO R3
0090   02 FE 52 12                  shift MULTIPLIER one bit left
0094   02 7E 52 12
0098   02 7E 52 12
009C   02 7E 52
009F   C7           LSNF            Set Q if carry=1 (DF=1)
```

```
LOC.    DATA   MNEMONIC   REMARKS
00A0    7B     SEQ
00A1    C4     NOP
00A2    F8     LDI        Reset pointer
00A3    E8
00A4    A2     PLO R2
00A5    39     BNQ        Skip addition routine if carry=0
00A6    BF
00A7    E1     SEX R1     *****ADDITION ROUTINE*****
                          In this section of the program, I think
                          it will be clearest if I simply describe what
                          register operation is performed
00A8    03     LDN R3     M(R(3)) to D
00A9    F4     ADD        M(R(1))+D to D, DF
00AA    53     STR R3     D to M(R(3)), increment
00AB    13     INC R3
00AC    11     INC R1
00AD    F8     LDI        Set up counter to loop three times
00AE    03
00AF    A5     PLO R5
00B0    03     LDN R3     M(R(3)) to D, add with carry
00B1    74     ADC
00B2    53     STR R3     D to M(R(3)), increment
00B3    13     INC R3
00B4    11
00B5    25     DEC R5     Counter=Counter-1, R(5) to D
00B6    85     GLO R5
00B7    3A     BNZ        GOTO M(B0) if D≠00
00B8    B0
00B9    F8     LDI        Reset pointers
00BA    F0
00BB    A1     PLO R1
00BC    F8     LDI
00BD    F8
00BE    A3     PLO R3
00BF    7A     REQ        Turn Q off
                          *****END ADDITION ROUTINE*****
00C0    24     DEC R4     decrement main loop counter
00C1    84     GLO R4
00C2    3A     BNZ        GOTO M(7D) if counter≠00
00C3    7D
00C4    E3     SEX R3     set X=3 to display ACCUMULATOR
00C5    F8     LDI        Set up counter to loop four times
00C6    04
00C7    A5     PLO R5
00C8    25     DEC R5     Counter=Counter-1
00C9    3F     BN4        Wait for INPUT pressed
00CA    C9
00CB    37     B4         Wait for INPUT released
00CC    CB
00CD    64     OUT 4      Display and increment pointer
00CE    85     GLO R5     counter to D
00CF    3A     BNZ        GOTO M(C8) if D≠00
00D0    C8
00D1    00     IDL        HALT
```

NOTE: The basic structure of this program is an adaptation of a general idea in *Microprocessor Programming for Computer Hobbyists* by Neil Graham, Tab Books is the publisher.

DOODLE PROGRAM

By Jay Mallin

Perhaps the best feature of your Super Elf—and possibly the one that persuaded you to buy it—is its video graphics ability. No other micro in the same price range can do video graphics.

However, displaying pictures and designs can take some work. First you might find yourself coloring in little squares on graph paper to make the design, then coding all those little squares into hex.

This program, a doodler, simplifies the process to the point of fun by allowing you to draw on your screen as you watch by controlling a blinking cursor. Using the keyboard you can move the cursor in any direction and both write and erase with it. And while it might look best on an expanded Super Elf with a full page or more for the display, it was specifically designed to fit into a half page of memory, for use in an unexpanded Super Elf.

The program works via a blinking cursor, a single bit in size. The cursor bit is stored as a byte with 7 zeros and a one in RA.0, with its address in the display in R4. The cursor is moved by manipulating the information in these two registers.

For example, to move the cursor on the screen left one place, the byte in RA.0 is shifted left once. To move the cursor up one, eight is subtracted from R4, since 8 bytes less in memory is displayed on the screen as one line up. Moving diagonally is done with a combination of these two.

The program then writes RA.0 into M(R4) so the cursor shows up on the screen. The information in R4 and RA.0 is also used to write or erase in the cursor bit's location, under instructions from the keyboard.

The computer reads instructions from the keyboard whenever the INPUT switch is depressed. Each bit in the control byte read tells it whether or not to perform a separate operation (see figure 1). As an example, the first bit, bit 0, tells it whether to merely move the cursor or if it should perform a write or erase. Other bits in the control byte tell the program what combinations of directions to move the cursor.

The program has a few interesting internal features. One is the use of the 00 instruction in a loop to produce a delay. It causes the computer to idle until a display interrupt occurs. The 00 instruction is also used in the interrupt display routine between each line of the display, instead of the two E2s that are normally used to waste four cycles. Here the 00 causes the computer to wait for DMA, and saves space.

To use the program (this is the part you were looking for, right?), first clear the memory. If you don't already have a program to do this, a small one is provided that will leave just two bytes in the memory. (It's possible to write a program that will clear all but one byte—try it sometime.) Simply put in the clear program and press GO. The 0 state light will flash for an instant and the program will have done its work.

Now enter the doodle program, checking for mistakes. When it is entered correctly, press GO. A blinking square will appear in the upper left hand corner of the screen. The bottom half is filled with the program, the top half of the picture is blank except for the blinking cursor.

To use the program, use the diagram of your keyboard (figure 1). The nine keys with arrows will move the cursor in the direction of the arrow. The 7, B, and F are the mode keys. First you push a mode button, then a direction button, then hold down the INPUT switch.

Try 7A, or "move only, down and to the right." The blinking cursor will move in that direction as long as you hold the INPUT switch down. Then try B4, or "write upward." To erase that, use F8, or "erase downward." Play a while, just remember to always push a mode key first, then a direction. Other combinations can have unpredictable effects. If something goes wrong, you probably hit the wrong keys. Just push reset and restart the program. Also, you are welcome to try to move the cursor out of the picture area.

After a while, you may notice you have a screen filled with strange designs and no fast way to clear it. Try doing the following:

- Stop the program and put 01 in address 59 without changing any other bytes (remember the monitor changes address 20).
- Now start the program again.

The cursor will appear as a gray square. You've shortened the delay, and so sped it up. Push F2 and hold the INPUT switch down. The cursor will streak through each bit, erasing everything. Now try writing with the speeded up cursor. You can send it in one diagonal direction for a while and then the other for unusual patterns. Use it to turn the whole screen white, then put 0C back in address 59 to slow the cursor back down. You can then draw in the negative, with the write as erase and vice versa.

For one last trick, use the cursor at normal speed to draw a figure like a square. Then put the 01 in 59 to speed everything up, and an F3 (Exclusive-OR) into 26. Start the program, push B2, and hold down INPUT. The cursor will turn the picture into a negative of itself. Keep holding it down, and it will change back into the original. The Exclusive-OR inverts each bit, changing it into what it wasn't before. The program uses the same idea to flash the cursor, inverting a bit, pausing, then inverting again.

Using more memory there are a number of ways to go further. You could have the program take the

(Continued on next page)

cursor speed as input each time it starts. Perhaps you would like to only have to enter the mode when you want to change it. You could even make the 3 key into another mode key, as an Exclusive-OR or even something else.

If you want to use a whole page or more for the display, simply change what the upper half of the address registers are set to, and modify the four bits which limit the picture to half a page and begin at byte 50. Displaying more than a page will mean the interrupt routine must be changed, but be sure to move the location of the stack if necessary so it does not write over the end of the routine.

You could also write an animation program in which the program writes and erases the picture itself. It would just take control bytes from a memory table instead of the keyboard.

As usual with your Super Elf, it is simply up to your imagination.

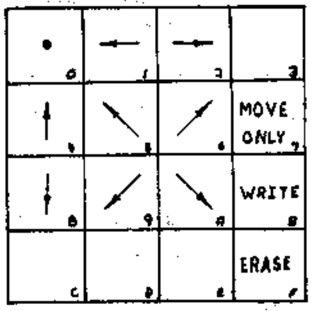


FIGURE 1

KEYBOARD CONTROL
HOW INPUT BYTE IS ANALYZED

	INPUT	MEANING
	1	If zero, neither write nor erase
F	1	If one, write; if zero, erase
	1	Doesn't matter
	1	Doesn't matter
	0	If one, move cursor down
5	1	If one, move cursor up
	0	If one, move cursor right
	1	If one, move cursor left

So, F5 erases where the cursor is, then moves it up one and to the left one.

CLEAR PROGRAM

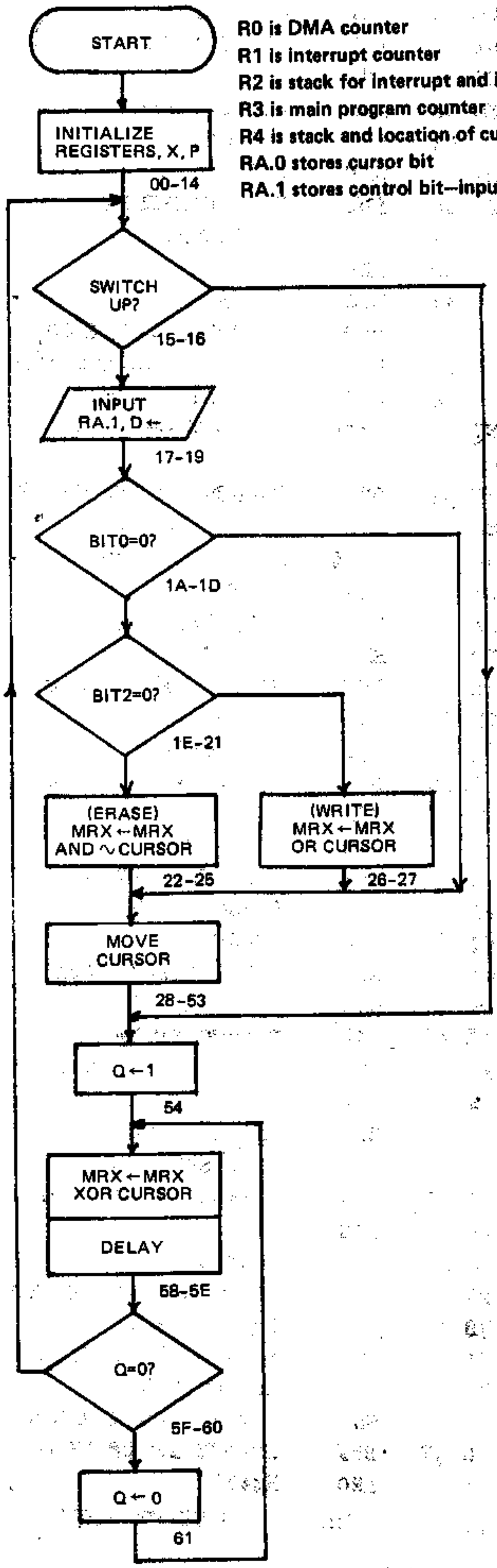
```
LOC.  CODE   MNEM.
0000  F8 05  LDI
0002    AA
0003  EA     SEX
0004  F8 00  LDI
0006  73     STXD
0007  30 06  BR
```

To clear memory with before entering Doodle Program

DOODLE FLOWCHART

REGISTERS

R0 is DMA counter
R1 is interrupt counter
R2 is stack for interrupt and input
R3 is main program counter
R4 is stack and location of cursor
RA.0 stores cursor bit
RA.1 stores control bit—input



RUN CLEAR PROGRAM THEN ENTER DOODLE LISTING

LOC.	CODE	MNEM.	ACTION
00	F8 00	LDI	SET UPPER
02	B1 B2	PHI	REGISTERS
04	B3 B4	PHI	
06	F8 0A	LDI	SET PROGRAM
08	A3	PLO	COUNTER
09	D3	SEP	SET P
0A	F8 66	LDI	SET INTERRUPT
0C	A1	PLO	COUNTER
0D	F8 80	LDI	SET
0F	AA	PLO	CURSOR
10	A4	PLO	CURSOR LOCATION.
11	A2	PLO	INTERRUPT STACK AND
12	22	DEC	INPUT ADDRESS
13	69	OUT1	DISPLAY ON
14	E4	SEX	SET X
15	3F 54	BN4	BRANCH IF SWITCH UP
17	E2	SEX	SET X FOR INPUT
18	6C	INP4	READ KEYBOARD
19	E4	SEX	RESET X
1A	BA	PHI	MOVE INTO RA.1
1B	FE	SHL	D LEFT ONCE
1C	3B 28	BNF	BRANCH IF DF IS 0
1E	FE	SHL	D LEFT ONCE
1F	8A	GLO	GET CURSOR
20	3B 26	BNF	BRANCH IF DF IS 0
22	FB FF	XRI	INVERT CURSOR
24	F2	AND	D AND MRX
25	38	SKP	SKIP NEXT BYTE
26	F1	OR	CURSOR OR MRX
27	54	STR	STORE
28	9A	GHI	GET CONTROL BYTE
29	F6	SHR	SHIFT RIGHT
2A	3B 35	BNF	BRANCH IF DF IS 0
2C	F6	SHR	SHIFT RIGHT AGAIN
2D	BA	PHI	SAVE CONTROL BYTE
2E	8A	GLO	CURSOR INTO D
2F	7E	RSHL	RING SHIFT LEFT
30	3B 3F	BNF	BRANCH IF DF IS 0
32	24	DEC	R4-1
33	30 2F	BR	BRANCH
35	F6	SHR	SHIFT CONTROL RIGHT
36	3B 41	BNR	BRANCH IF DF IS 0
38	BA	PHI	SAVE CONTROL BYTE
39	8A	GLO	GET CURSOR
3A	F6	SHR	SHIFT CURSOR RIGHT
3B	3B 3F	BNF	BRANCH IF DF IS 0
3D	14	INC	R4+1
3E	76	RSHR	RING SHIFT RIGHT

LOC.	CODE	MNEM.	ACTION
3F	AA	PLO	SAVE CURSOR BYTE
40	9A	GHI	GET CONTROL BYTE
41	F6	SHR	SHIFT RIGHT
42	3B 49	BNF	BRANCH IF DF IS 0
44	84	GLO	GET CURSOR ADDRESS
45	FF 08	SMI	D-08
47	30 4F	BR	BRANCH
49	F6	SHR	SHIFT CONTROL RIGHT
4A	3B 50	BNF	BRANCH IF DF IS 0
4C	84	GLO	GET CURSOR ADDRESS
4D	FC 08	ADI	D+08
4F	A4	PLO	SAVE CURSOR ADDRESS
50	84	GLO	GET CURSOR ADDRESS
51	F9 80	ORI	OR 80; TO KEEP IN
53	A4	PLO	BOUNDARIES
54	7B	SEQ	SET Q
55	8A	GLO	GET CURSOR
56	F3	XOR	D XOR MRX
57	54	STR	STORE D
58	F8 0C	LDI	SET D FOR DELAY
5A	00	IDL	WAIT FOR INTERRUPT
5B	FF 01	SMI	D-01
5D	3A 5A	BNZ	BRANCH IF D NOT 0
5F	39 15	BNQ	BRANCH IF Q NOT 0
61	7A	REQ	RESET Q
62	30 55	BR	BRANCH

DISPLAY ROUTINE

LOC.	CODE	MNEM.	ACTION
64	72	LDXA	LOAD X, RX+1
65	70	RET	RETURN
66	C4	NOP	NO-OP FOR TIMING
67	22	DEC	R2-1
68	78	SAV	SAVE T
69	22	DEC	R2-1
6A	52	STR	STORE D
6B	F8 00	LDI	SET UPPER
6D	B0	PHI	REGISTER
6E	F8 80	LDI	SET LOWER
70	A0	PLO	REGISTER
71	80 00	GLO	SET D; WAIT
73	A0 00	PLO	RESET R0; WAIT
75	A0 00	PLO	"
77	A0 00	PLO	"
79	3C 71	BN1	BRANCH IF NOT DONE
7B	30 64	BR	BRANCH

7D to 7F is stack for interrupt and input

NEW PATTERNS

By Mike Tyborski

Graphics is an extremely interesting and enjoyable application for personal computers. It is through its use that games, art and animated effects are created.

The Quest Electronics Super Elf provides bit-mapped graphics capabilities on a budget, using the CDP1861 video IC. Through proper programming, display resolutions up to 64Hx182V dots may be obtained and complex animated effects are possible.

PATTERNS, a semi-animated pattern drawing program, took advantage of this feature as shown in QUESTDATA #6. It plotted points in a 64Hx128V array as calculated by the simple mathematical formula; X:=X+Y/2 and Y:=Y+X/2. In addition, ten more formulas were given for user experimentation. In their basic form, various combinations of circular patterns were produced.

PATTERNS, however, contained one typographical error that prohibited execution. This error is located in the INTERRUPT routine at the address 0019H. It is simply corrected by changing the CA to C4 (NOP). The program will now run as described.

Additional patterns may be created as desired, by rewriting the NEW POINT CALCULATION subroutine. This subroutine is located at 00A7H and is responsible for computing new point values according to a specific formula.

The original article did not fully explain how to write new routines. This can lead to boredom after the initial effects of PATTERNS wears off. The following points should help you create new patterns:

- The (X,Y) coordinates for plotting are stored sequentially in memory, and are accessed using R6 as a data pointer.
- R6 is pointing to X upon entering NEW POINT.
- Intermediate results must be placed in the stack pointed to by R2. This should be accomplished with a SEX R2 instruction at 00A8H. This should then be changed to SEX R6 before returning to the MAIN program.
- Multiplying by a power of 2 is accomplished using the left shift (SHL—FE) instruction as required. For example, to multiply by 8 just execute three SHL instructions. This is because $2^3=8$.
- Division by a power of 2 is accomplished using the right shift (SHR—F6) instruction.

The user should now be able to write routines for most of the formulas that were given in the original article. As an aid, four additional NEW POINT subroutines are included here.

I hope this update has ironed out any problems that may have been encountered and that graphics is no longer a mystery.

Coming soon: Graph Pac—a graphics subroutine package that includes rotation and scaling functions.

```
EQUATION  X Old=X  X:=X-Y/2   X:=Y+(X Old/2)
LOC    CODE     MNEMONIC      COMMENTS
00A7   D3 EXITN:SEP R3         RETURN
00A8   E2        SEX R2        Use (R2) as stack
00A9   06        LDN R6        Store X/2 in stack
00AA   F6        SHR
00AB   73        STXD
00AC   46        LDA R6        Save X in stack
00AD   52        STR R2
00AE   06        LDN R6        Compute Y/2
00AF   F6        SHR
00B0   F5        SD            Compute X-Y/2
00B1   26        DEC R6        Save as X New
00B2   56        STR R6
00B3   12        INC R2        Point to X Old/2
00B4   16        INC R6        Compute Y+(X Old/2)
00B5   06        LDN R6
00B6   F4        ADD
00B7   56        STR R6        Save as Y New
00B8   E6        SEX R6        Restore X
00B9   30 A7     BR EXITN      GOTO RETURN
```

```
          EQUATION X:=X-(Y/2) Y:=Y+(X/4)
LOC    CODE     MNEMONIC      COMMENTS
00A7   D3 EXITN:SEPR3          RETURN
00A8   E2        SEX R2        Use (R2) as stack
00A9   46        LDA R6        Save X in stack
00AA   52        STR R2
00AB   06        LDN R6        Compute Y/2
00AC   F6        SHR
00AD   F5        SD            Compute X-(Y/2)
00AE   52        STR R2        Save result in stack
00AF   26        DEC R6        Save as X New
00B0   56        STR R6
00B1   F0        LDX           Compute X New/4
00B2   F6        SHR
00B3   F6        SHR
00B4   52        STR R2
00B5   16        INC R6          Point to Y
00B6   06        LDN R6
00B7   F4        ADD
00B8   56        STR R6        Save as Y New
00B9   E6        SEX R6        Restore X to R6
00BA   30 A7     BR EXITN      GOTO RETURN
```

LISTING FOR PATTERNS

```
LOC.  CODE         MNEM.  COMMENT
0000  90           GHI R0   Initialize registers
0001  B1           PHI R1
0002  B2           PHI R2
0003  B3           PHI R3
0004  B4           PHI R4
0005  B5           PHI R5
0006  B6           PHI R6
0007  F8 19        LDI      Interrupt routine
0009  A1           PLO R1
000A  F8 FF        LDI      Stack pointer
000C  A2           PLO R2
000D  F8 28        LDI      Main
000F  A3           PLO R3
0010  F8 95        LDI      Random number subroutine
0012  A4           PLO R4
0013  F8 A8        LDI      New Point Subroutine
0015  A5           PLO R5
0016  D3           SEP R3   Go to Main
0017  72   INTRET: LDXA     Return to main
0018  70           RET
0019  C4      INT: NOP      4 Block display format
001A  22           DEC R2
001B  78           SAVE
001C  22           DEC R2
001D  52           STR R2
001E  E2 E2        SEX R2; SEX R2
0020  F8 04        LDI 04H  Display block
0022  B0           PHI R0
0023  F8 00        LDI 00H
0025  A0           PLO R0
0026  30 17        BR INTRET
0028  F8 04START:LDI 04H   Clear display memory
002A  B9           PHI R9
002B  93           GHI R3
002C  A9           PLO R9
002D  93     LOOP: GHI R3   Zero a byte
002E  59           STR R9
002F  19           INC R9   Done with blocks?
0030  99           GHI R9
0031  F8 08        XRI 08H
0033  3A 2D        BNZ LOOP; no
0035  E6           SEX R6   Activate video IC
0036  69           INP 9
0037  F8 EE        LDI Y    Set X, Y to random
0039  A6           PLO R6   Initial values
003A  D4           SEP R4   Call RND
003B  FC 0B        ADI 0BH
003D  FA 7F        ANI 7FH  Get Y in display range
003F  73           STXD
0040  D4           SEP R4   Get random X
0041  FA 3F        ANI 3FH
0043  56           STR R6

LOC.  CODE         MNEM.  COMMENT
0044  F8 02        LDI 02H  Set point counter 0200H
0046  B7           PHI R7
0047  F8 04  PLOT: LDI 04H  Set R9 to starting
0049  A9           PLO R9   Block of display
004A  F8 ED        LDI X    Is X out of range?
004C  A6           PLO R6
004D  F0           LDX
004E  FF 41        SDI 41H
0050  33 8D        BPZ NPOINT; yes
0052  72           LDXA     Compute X/8
0053  F6 F6 F6     SHR; SHR; SHR
0056  52           STR R2; Save in stack
0057  F0           LDX      Is Y out of range?
0058  FF 81        SDI 81H
005A  33 8D        BPZ NPOINT; yes
005C  F0           LDX      Compute Y • X
005D  FE           SHL      Times 2
005C  3B 61        BNF TIME4; Overflow?
0060  19           INC R9   Yes
0061  FE    TIME4: SHL      Times 4
0062  3B 65        BNF TIME 8; Overflow?
0064  19           INC R9   Yes
0065  FE    TIME8: SHL      Times 8
0066  3B 69        BNF ADD; Overflow now?
0068  19           INC R9   Yes
0069  E2      ADD: SEX R2   Add X/8+Y • 8
006A  F4           ADD
006B  3B 6E        BNF NC   Any carry
006D  19           INC R9   Yes
006E  A8       NC: PLO R8   Set R8 to point addr.
006F  89           GLO R9
0070  B8           PHI R8
0071  F8 80        LDI 80H  Set bit position
0073  52           STR R2
0074  26           DEC R6   Compute X • 7 and
0075  06           LDN R6   put in counter R9
0076  FA 07        ANI 07H
0078  A9           PLO R9
0079  32 82        BZ DISP
007B  F0      BIT: LDX      Shift 80H required
007C  F6           SHR      Number of times
007D  52           STR R2
007E  29           DEC R9   Done?
007F  89           GLO R9
0080  3A 7B        BNZ BIT; no
0082  08      DISP LDN R8   Activate desired bit
0083  F1           OR
0084  58           STR R8
0085  F8 02        LDI 02H  Delay
0087  B9           PHI R9
0088  29      DLY: DEC R9
0089  99           GHI R9   Time up?
008A  3A 88        BNZ DLY; no

008C  27           DEC R7   Decrement point counter
008D  E6   NPOINT: SEX R6   Restore X to R6
008E  D5           SEP R5   Calculate new X, Y
008F  97           GHI R7   Required points plotted
0090  3A 47        BNZ PLOT; no
0092  30 28        BR START; Begin again
* * * * RANDOM NUMBER SUBROUTINE * * * *
0094  D3     EXITRSEP R3    Return
0095  86      RND: GLO R6   Save data pointer
0096  52           STR R2
0097  F8 EF        LDI rndnum; compute 5 times
0099  A6           PLO R6   Old random number
009A  F0           LDX
009B  FE FE        SHL; SHL
009D  F4           ADD
009E  FC 02        ADI 02H
00A0  56           STR R6   Set as new rndnum
00A1  A9           PLO R9
00A2  02           LDN R2   Restore data pointer
00A3  A6           PLO R6
00A4  89           Glo R9   Return to Main
00A5  30 94        BR EXITR; with rndnum in D
* * * * NEW POINT CALCULATION SUBROUTINE * * * *
00A7  D3    EXITN: SEP R3   Return
00A8  16      NEW: INC R6   Compute Y/2
00A9  06           LDN R6
00AA  F6           SHR
00AB  FB FF        XRI FFH  Negate result
00AD  26           DEC R6
00AE  F4           ADD        XNew=X+(-Y/2)
00AF  56           STR R6
00B0  16           INC R6     YNew=Y+XNew/2
00B1  F6           SHR
00B2  F4           ADD
00B3  56           STR R6
00B4  30 A7        BR EXITN
```

NEW PATTERNS

(Continued)

```
               EQUATION X:=X+1   Y:=Y+X
LOC.   CODE    MNEMONIC      COMMENTS
00A7   D3 EXITN:SEP R3        RETURN
00A8   E2        SEX R2       Use (R2) as stack
00A9   06        LDN R6       Compute-X+1
00AA   FC 01     ADI #01
00AC   56        STR R6       Save as X New
00AD   52        STR R2       Save in stack
00AE   16        INC R6       Point to Y
00AF   06        LDN R6       Compute Y+X
00B0   F4        ADD
00B1   56        STR R6       Save as Y New
00B2   E6        SEX R6       Restore X
00B3   30 A7     BR EXITN     GOTO RETURN
```

```
                 EQUATION X:=X-Y   Y:=Y+(X/2)
LOC.   CODE    MNEMONIC      COMMENTS
00A7   D3 EXITN:SEP R3        RETURN
00A8   E2        SEX R2       Use (R2) as stack
00A9   46        LDA R6       Save X in stack
00AA   52        STR R2
00AB   06        LDN R6       Compute X-Y
00AC   F5        SD
00AD   52        STR R2       Save result in stack
00AE   26        DEC R6       Save X New
00AF   56        STR R6
00B0   02        LDN R2       Compute X/2
00B1   F6        SHR
00B2   52        STR R2
00B3   16        INC R6       Add result to Old Y
00B4   06        LDN R6
00B5   F4        ADD
00B6   56        STR R6       Save as Y New
00B7   E6        SEX R6       Restore X
00B8   30 A7     BR EXITN     GOTO RETURN
```

NEWS FLASH!
CHIP-8 INTERPRETER FOR ELF
A new booklet by Paul Moews

At last Elf users and VIP users can speak in one common language—CHIP-8 INTERPRETER. The CHIP-8 interpreter for the Elf is identical to the RCA VIP version in every way. It is fully relocatable and includes many original additions to the language such as new skip instructions, multiply, divide and 16 bit display. Paul Moews has designed the interpreter to work with 4K Elf systems with 1861 video. He has, however, included a demonstration of a limited subset of CHIP-8 in his booklet—to whet the appetite of 256 byte Elf owners. The full language will allow you to run all of the VIP programs contained in VIP-320 VIP User Guide (contains 20 interesting games and costs $5), VP-710 VIP Games Manual (contains "more exciting games. . including Blackjack, Biorythm, Pinball, Bowling and 10 others—price is $10).

RCA also has a new booklet which tells how to use the VIP interpreter language to create your own programs. This booklet, VIP Instruction Manual VIP-311 is priced at $5. The new Moews booklet does an excellent job of taking you through worked out examples of creating a program.The booklet also tells the differences between the various Elf systems, gives the register usages, and is very well documented throughout. This welcome Moews booklet, PROGRAMS FOR THE COSMAC ELF-INTERPRETERS, will help unite all 1802 owners.

The booklet includes a number of demonstration programs, an addition game, and an ASCII code display program. Paul Moews shows you the entire CHIP-8 interpreter with extensions and tells how you can alter it to suit your needs.

The Moews INTERPRETERS booklet is available from Quest Electronics, P.O. Box 4430, Santa Clara, CA 95054 for $5.50 plus .50 cents postage and handling. RCA VIP booklets can also be ordered from Quest at RCA's listed prices.

MORE MUSICAL MADNESS

By Bobby R. Lewis

Here is how to turn your Elf keyboard into a tone generator. If you are going to run this program in an expanded memory system be sure and set R4.1. To use the program, simply load memory and switch to run. At that time, tones will be heard according to what keys are pressed on the keyboard. You can change the range by changing the byte at address 0A. A 00 or 01 entry will generate a pause. An FF will give a low note. This program is written for Elf systems and because of the way the keyboard is latched up, there will be no pause between tones.

An easy way of listening to this program is to tune a small AM radio between stations around 1400 KC and set it near the Q LED. Surprisingly good results can be obtained with this method. Another way of obtaining output if you don't have a Super Elf system is to feed the cassette output line into a phono input of a stereo if you have a Netronics giant board.

MUSICAL KEYBOARD PROGRAM

```
LOC.   CODE      COMMENTS
0000   90 B4     Initialize Register 4.0
0002   F8 12 A4  Address of wk. area
0005   E4        Set X=4
0006   6C        INPUT from keyboard
0007   64        Output to display
0008   24        DEC so X=same loc.
0009   F0        Put contents into D
000A   7B        Turn Q on
000B   FF 01     Subtract (1) from D
000D   3A 0A     Branch until D=00
000F   7A        Turn Q off
0010   30 06     Get another entry
0012   00        Work Area
```

BUG SQUASHER

Dear Mr. Haslacher,

Your articles are very interesting and quite informative, I would like to take this opportunity to question the examples made on page 2 and 3 of Issue Number 7. On page 2, paragraph 3 of the left hand column, the statement is made "It turns out that 80 hex will do the job for us." I believe that 40 hex should do the job!

On page 3 in the left hand column for the conversion of the hex code to ASCII code example, I believe the correct conversion should show 00110000 (30 hex) with the resulting D(f) of 00111000 (38 hex). I do not mean to be critical except in a constructive way.

Thank you for taking the time to read my letter and please keep turning out these interesting articles.

Sincerely yours,
Richard E. Warner

[There is no hiding behind my data mask on that one. Thank you my sharp eyed and gentle reader.
—Bill Haslacher]

QUESTDATA
P.O. Box 4430
Santa Clara, CA 95054

Publisher . . . . Quest Electronics
Editor . . . . Bill Haslacher
Technical Coordinator . . . . . . Jane McKennon
Circulation . . . . Laurie Buzzell
Proofreading . . . . Ken Brown

COSMAC CLUB COSMAC CLUB COSMAC CLUB COSMAC CLUB COSMAC CLUB COSMAC

8

QUESTDATA
P.O. Box 4430
Santa Clara, CA 95054

ADDRESS CORRECTION REQUESTED

BULK RATE
U.S. Postage Paid
QUEST
Electronics
Permit No. 549
Santa Clara, CA