Questdata

Volume 1 Issue # 4 ©

SOFTWARE–WHICH PATH LEADS TO GOLD?

THE race is on. Which race, where? The great software race is on, and while the race is still close to the starting gate; it is important to look at the major horses to find out who is betting on which horse, and why.

We have national microcomputer magazines and manufacturers taking the lead. *Byte* Magazine is betting on something it has devised and calls "Paper Bytes®." In this *Byte* system, the code to be read into the microcomputer will be coded on paper, and will be read into the computer with an optical wand. The code in this system resembles the coded markings on the cans of your local supermarket. This is certainly an economical system, given the cost of printing and the distribution by mails, and if only because *Byte* is the largest circulation microcomputer publication, is sure to have a following.

There are some frustrations to the *Byte* system, however. The system of reading optical code, suposes that you own a scanner (probably a $30 item). It also assumes that you own some other type of storage device such as cassette tape; since wanding in codes for ten pages is apt to get tiresome. Another drawback of this approach is the inability of the user to produce his own wand readable code without very elaborate equipment.

Radio Shacks TRS-80 is producing cassette tapes with prerecorded programs already on them. The cost of production of such a program media is bound to be more, but the convenience of buying cassette tape directly is an advantage over the *Byte* optical wand system. The Radio Shack cassette idea assumes that you own a cassette recorder and own a Radio Shack TRS-80 computer. The cost of copying magnetic tape will always be higher than printing or pressing a record, but apparently this does not worry Radio Shack at the moment. If you compare cassette tape to magnetic disc, the cassette tape comes out behind in "speed of getting information into a computer."

Floppy disc computing systems have real advantages over cassette tape in both speed and convenience. A disc will load in 10 seconds what a cassette tape will take 5 minutes to load. Also, you can store discs in record like books that take up very little space. To get to the program you want never means having to rewind or play through a cassette tape–

access is immediate.

The catch with disc storage of software is the cost. The cost is tumbling downward, but a good disc will still cost you $600. Disc has some hidden advantages in the cost and convenience of mailing, and the cost of stamping out discs for mass sales. However, because of the cost, a mass market for discs for the hobbiest has not yet emerged.

Interface Age Magazine has an interesting idea called the Floppy Rom®. This media is nothing more than a plastic phonograph record and it has some interesting advantages over other systems. It is economical to produce records, easy to store them, you can jump to the track of your choice just by moving the phonograph needle, and almost everyone already owns a record player.

Paper tape could be a sleeper in this software race. This overlooked horse is extremely reliable, reasonably priced, and easy to store. If you have a pull thru paper tape reader–RAECO tape readers cost $32.50 –you can grab your program tape and pull it thru like grease lightening. Generating your own paper tape calls for owning a teletype or punch system such

(Continued on Page 2)

(Continued from Page 1)

compartments for fast access to programs. Generating your own paper tape calls for owning a teletype or punch system such as the Heathkit punch-reader at $370. A hand punch for creating an original program is not out of the question since it has been estimated to take about a half-hour and up to generate a line of debugged machine code. Reading paper tape through a standard teletype machine is a slow process, and if it weren't for the reliability factor of paper tape, it would probably fall even further behind the leaders in the software competition.

Under the heading of exotic future systems we find the following: TV cassette recorders, TV disc, and Jim Warren's (of *Dr. Dobb's Journal* fame) SCA radio broadcasting. TV cassette and disc recording will allow even denser program storage than now exists. Since TV disc (using laser technology, etc.) has not been marketed as yet, it is diffieult to talk about. The SCA idea is to set up special frequency radio stations and actually transmit software over the air. Such a system would have to have sponsors, since software is not "free." Alas, the crystal ball for the future grows cloudy and the crystal resonates unclearly about the future.

There are still other horses in the software media race. These would be publications in human readable object code form. Magazines are likely to buy and publish software and documentation for some time to come. Such code is easy to alter, and you only have to put the code into the computer once and then transfer it to your preferred media. Books of software programs are starting to appear on the market. The publications approach offers a real opportunity to the freelance writer and the self-publisher. About such an approach, one QUEST-DATA reader, James Stephens, writes: "Programming in Hex with this 1802 chip is easy and sensible and one doesn't have to go completely off the deep-end to get a complete sequence into the processor. I think the 1802 chip is 'Super.'"

The publications approach offers a real opportunity to the freelance writer and the self-publisher.

So there you have some of the major contenders in the software media race. So what do you think; who has the "best approach" and why? The industry is still in the "learning "phase and your vote for best software media counts. Let QUESTDATA know your thoughts on this great issue of our time. Must one software approach win and the others lose? Can they all win? Stay tuned, this is shaping up to be one-heck-of-a-fascinating race.

NEW GRAPHICS PROGRAM BOOKLET IS PUBLISHED BY PAUL C. MOEWS

There are a lot of interesting things you can do with graphics with your basic 256 byte Super Elf or Elf II. This booklet shows you how to make your Elf into kaleidoscope pattern generator and tickertape-like display system. You can also have a horse race on your video screen. The final two programs are a TV stop watch and a TV clock; the clock shows hours, minutes, and seconds and is completely setable. The booklet contains a discussion of graphic interrupt routines along with the eight programs.

This exciting new booklet can be yours for only $3.00 and 50 cents extra for postage and handling. Please make checks payable to Quest Electornics.

VORTEX II

By Bob Richie

This music sounds like its title. You can almost see the vortex doing its dance. This new type of sound can be called indigenous music because it is uniquely suited to the computer. This is a new frontier, and it must be admitted that it takes some adjustment of the musical senses to get used to it. The actual aesthetic value of this kind of music has yet to be established. It used to be that music was limited by how fast the human hand could span a keyboard, but with the advent of the computer the question seems to b —what will the human mind conceive of next.

Soon you may be requesting to hear "the sound of the square-root of pi" from your local radio station. Anyhow, Vortex II is an example—a mere beginning—of sounds of the future. We will have more on this kind of indigenous music in future issues of QUEST-DATA.

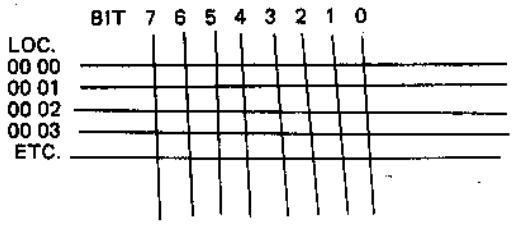
LOC.	CODE	MNEM.	COMMENT
00	F8	LDI	Load delay
01	77		
02	A1	PLO R1	Into R(1).0
03	21	DEC R1	Decrement delay
04	81	GLO R1	Move delay to D Reg.
05	32	BZ	Branch if delay finished
06	18		
07	A1	PLO R1	Load 00 to Reg. 1
08	A2	PLO R2	Load 00 to Reg. 2
09	7A	REQ	Reset Q (Turn off Q)
0A	22	DEC R2	Decrement Reg. 2
0B	82	GLO R2	Test R(2).0
0C	32	BZ	Test delay
0D	03		
0E	A3	PLO R3	Load R(3).0
0F	23	DEC R3	
10	83	GLO R3	
11	3A	BNZ	Test another delay
12	0F		
13	31	BQ	If Q is on turn it off
14	09		
15	7B	SEQ	Turn Q on
16	30	BR	Branch for next cycle
17	0B		

18 30

19 00

MEMORY ORGANIZATION

A mathematician a long time ago is said to have come up with the concept of the matrix while gazing at a busy spider making a web. Ahhha, the great math personage is said to have mumbled, "If one devises a system of rows and columns in a rectangular arrangement, you get some interesting properties." And thus it was that the usefulness of the matrix arrangement was first contemplated. Modern computer memory is arranged in a big matrix. By identifying a particular column (address), you can reveal its contents (op code or data). Thus, computer memory can be thought of to look something like this:



Note that we call the hex numbers addresses or locations, and use them to get to the data contents. Now what a computer does is to automatically step or sequence itself through the memory, starting at location 00 and marching upward through memory. In the 1802 microcomputer, we have the potential of addressing 65K bytes (a byte being eight of the points in the matrix or 8 bits, as they are called). So the computer marches through locations 00 00 to FF FF. Memory can be further divided into pages. The first page of memory is called page 0, and it reaches from location 00 00 to location 00 FF. When the memory steps into location 01 00 it has stepped into page 1. The short branches (30 the unconditional branch being an obvious example), stay within a page of memory. Since the instruction will only let you state two hex digits for the jump to location, you can see how it is that the computer can only branch within the memory page you are currently programming.

One of the hidden benefits of page addressing is that if you use nothing but this type of addressing you can easily relocate your program. A program written using nothing but short branches can be moved to *any* page in memory and still run perfectly. So how would you get to a new page of memory using nothing but short branches. One possible way to do this would be to jump to location FF on that page and ride over the page boundary using NOPs.

A NOP (machine code C4) is an instruction which does absolutely nothing but execute 3 machine cycles of time. This NOP when placed in location FF will increment your computer into the next page of memory.

Another way you might wish to use a NOP (C4) is to pad your machine code so that you can come back and add things to the middle of your code without changing all of the jumps. We will show you an example of this technique later on in this machine language section.

THE PROGRAM COUNTER REGISTER

So what is it that steps your microcomputer sequentially through memory? With most computers you have no control over and are almost unaware of the existence of the program counter (PC). The program counter is a register which very quietly increments itself after each instruction in a program is executed. You are never really conscious of this PC Register when you are programming other (non-1802) computers. The fact is that the 1802 is unique in giving the programmer a choice of program counters. You can think of the PC as a register which executes a very simple program. This program is: Increment the PC, execute the next instruction, increment the PC, etc.

The ability to name your PC allows some very interesting programming operations. Any one of the 16 general purpose scratch-pad registers can be made the program counter. What would happen if we put address 04 00 into Register B, and then switched to Register B as our program counter.

Correct, the next instruction we would execute would be in location 04 00. So what we have here is a very fast CALL to that part of memory we set the PC Register to designate. With a one byte instruction and 2 machine cycles of time, we have CALLED a subroutine. A subroutine is usually a small section of code which you wish to use repeatedly in your machine code. A good example of a subroutine would be a delay (time killing loop) subroutine. So let's see what this looks like, since a listing with comments is worth a thousand words:

```
LOC.  CODE  MNEM.  COMMENTS
 00    F8    LDI    Load R(4).0 with starting
 01    0C           location of subroutine
 02    A4    PLO4   Put D Reg. into R(4).0
 03    7B    SEQ    Turn on Q LED
 04    D4    SEP4   Switch PC thus delay routine
 05    7A    REQ    Turn off Q LED
 06    D4    SEP4   Delay routine again
 07    30    BR     Branch back to do again
 08    03
 09    C4    NOP    This is an example of a
 0A    C4    NOP    NOP—to take up space
 0B    D0    SEP0   This returns to main prog.
 0C    C4    NOP    Not necessary—an example
 0D    F8    LDI    Load a delay into high Reg.
 0E    FF
 0F    BE    PHIE   Put delay into R(E).1
 10    2E    DECE   Decrement Reg. E
 11    9E    GHIE   R(E).1 to D
 12    3A    BNZ    Test to see if delay up
 13    10           loop to 10 if not 00
 14    30    BR     Branch to reset subroutine
 15    0B           starting location which is 0C
```

The effect of executing this program is to turn the Q-LED on, delay, turn the Q-LED off, delay, etc. It is the SEP 4 (D4) instruction which calls the delay subroutine. Using the SEP 4 (D4—Set the PC to Reg. 4) instruction takes some setting-up to accomplish the Register switch. The first instruction F8 loads the starting location of the CALL routine into the D Register; the D Register is then transferred into the low part of Register 4. If you are using an extended memory system, you will also have to put 00 into the high memory of Register 4. The 7B turns on the Q LED, and the D4 sets the program counter to Register 4. When the computer begins running a program the PC is always Register 0 and the starting location is always 00. One of the RCA conventions is to use R3 for the MAIN routine program counter. In a short program such as this, it is not really essential to do this, but in long programs which use Standard Call and Return Technique (SCRT), it is necessary in order to avoid confusion in register assignments. So, the PC starts out as Register 0 and is switched to Register 4. This means that whatever is in location 0C is going to be the next instruction executed.

In other words, since the contents of Register 4 are 0C, the next location to be considered by the microprocessing unit (MPU) is 0C. The contents of 0C form the start of the delay routine. The delay is the same time killing tactic we used in QUESTDATA No. 3 to kill time in the stoplight problem.

At the end of the delay we come to an interesting unconditional branch to location 0B. The 30 0B branch is for the purpose of resetting the program counter back to 0 and to leave the contents of Reg. 4 pointing to the start of the delay routine. The last contents of program counter Reg. 4 before it passes control back to our main program in Reg. 0, will be 0C. Now that everything is set back to the way it was initially; the control passes back to where we left Reg. 0—namely location 05. When the contents of 05 are executed, the Q LED is turned off—7A.

The next insturction D4 repeats the delay routine as we did before, and leaves it to go to where Reg. 0 is pointing--location 07. The contents of 07 reveal a short branch to location 03. This passing of control will continue indefinitely—or until you decide you have seen enough and turn off your computer.

Keep the memory you have put into your computer loaded, and proceed to the next section.

PROBLEM IN SEARCH OF A SOLUTION

So you think you understand what is involved in the clever switching PC technique? OK! Now, for extra credit, try this one: Get the computer to alternately display on the LED display the letters AA, delay, BB, delay, AA, etc. Hint—review the sneaky dice problem in QUESTDATA No. 2 for use of the display immediate instruction. After you have figured it out, you can turn to page 7 just to check the solution.

No peeking before you figure it out—you should be able to get this one in under three minutes. One more thing: If you should peek your microcomputer will turn to dust and you will turn into a frog . . . just kidding . . .

COMING ATTRACTIONS:

- COSMAC Combination Lock
- TV Graphics—Shoot the Moving Target Game
- Indigenous Music—Witch Doctor, Songbird
- An Interesting Way to Handle Arithmetic
- And Much, Much, More . . .

LIGHT CONTROLS DARKROOM TIMER

Terry Owen Permenter

Communication with your COSMAC Super Elf doesn't have to be restricted to the INPUT switch—for less than a dollar, you can communicate using light. The COSMAC has four I/O flag input lines; the INPUT is one (they are EF1-EF4; the INPUT is EF4).

We can substitute a photocell for a switch as is used for INPUT. The cell must have a "high" dark resistance, and a very "low" light resistance to simulate a switch. A readily available, inexpensive cell with the proper characteristics is the Radio Shack part #276-116 cadmium sulfide photocell ($.99). This is con- and GND-ground. (Use a 47K ohm resistor between +5 and EF2). The Instructions 35 and 3D monitor the status of this line.

Once the photocell is connected, we can test the setup with a simple program: 35, 00, 7B, 3D, 03, 7A, 30, 00. Load this program starting at LOC# 00 and run it. Turn off the room lights, and the Q LED will come on. When you turn the lights on, the LED will go out. An automatic night light. This has rather limited uses, but let's examine what's happening before we go on. The first two bytes—35, 00 are a tight loop, with COSMAC repeatedly testing EF2. With the light on, EF2=1, and causes a branch back to 00, for another check. When the light is out, EF2=0, and the branch is ignored. 7B turns on the Q LED. Another tight loop, 3D, 03 has COSMAC waiting for the light to come back on, then 7A turns Q off. 30, 00 sends us back to the beginning.

To make the photocell more sensitive to an individual light, rather than any light, tape a cardboard tube around the photocell.

Now, the room lights can be on, and only a light shining down the tube will trigger the flag. this can be seen in stores, with a buzzer sounding when you walk between the light and the cell. That's one application you may want to use. Another is a darkroom print timer. The accompanying program can be used to accurately time exposures for enlargements in a darkroom. Simply aim the photocell-tube arrangement at your enlarging easel, then turn on the enlarger. When COSMAC displays the desired time, turn off the enlarger. The Q LED will come on, and the display will freeze the time. the program takes only 46 bytes, and combines the fun of two hobbies.

The meat of the program is the loop at LOC#0C-14. This loop times one second before going on. The loop is executed over 27,000 times for each second. In order to avoid having to think in HEX, the display counts seconds in base 10 (up to 99 seconds). This is done by adding 07 HEX every ninth second, thus skipping A, B, C, D, E, F. This requires the use of a counter to keep track of how long its been since the last big jump. To compare the counter with 09, I used the "Exclusive Or Immediate" command. Since the rules for Exclusive Oring are:

1 XOR 1 = 0
1 XOR 1 = 1
0 XOR 0 = 0
0 XOR 1 = 1

when the counter is 09 (1001_2), it will XOR to leave 00.

The next step in my book will be to connect a relay, for direct control of my enlarger, stereo, etc. I would be very interested in hearing from other COSMAC users with problems and/or suggestions.

Terry Owen Permenter
P.O. Box 1338
Bellingham, WA 98225

"ON BOARD" POWER CONNECTION FOR LOGIC PROBES

James C. Nicholson

In order to use a logic probe (such as the Continental Specialties Corp. LP-2) when checking out the Super-Elf circuitry, +5vDC and GND are needed. A simple modification to the Super-Elf PC board consists of soldering a short piece of resistor lead through one of the PC side-to-side connectors where the +5vDC is available. Clip the wire about ¼" over the top of the circuit board. The two power clips of the LP-2 can then be attached as follows: red (positive) to the short wire stub, and black (negative) to the ground side of capacitor C2. Power drain on the Super-Elf supply is negligible.

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

Publisher . . . . . **Quest Electronics**
Editor . . . . . **Bill Haslacher**
Technical Coordinator . . . . . **Bill Thompson**
Programming Assistance . . . . . **Pam Gazlay**
Proofreading . . . . . **Ken Brown**

1802 phototimer listing

LOC.#	OP CODE	MNEM.	COMMENTS
00	F8	LDI	Define work area; M(2D) is used as a scratchpad
01	2D		
02	A1	PLO	We'll use R(1).0 to point to our scratch pad
03	F8	LDI	"Zero" the scratchpad by storing '00' over
04	00		the previous contents
05	51	STR	
06	A3	PLO	"Zero" R(3).0 for use later as a counter
07	E1	SEX	We'll use X to refer to R(3)
08	64	OUT 4	Display beginning "time" (00) (and increment R(X))
09	21	DEC	OUT 4 increments R(X), so we decrement
0A	3D	BN2	Now we'll wait for the enlarger to come on
0B	0A		
0C	F8	LDI	This loop times the seconds . . .
0D	6D		
0E	B2	PHI	. . . we put HEX '6D' in R(2).1, and decrement R(2)
0F	22	DEC	until R(2).1=0 . . .
10	3D	B2	. . . during the loop, we continue to monitor
11	2B		the enlarger, and branch to the end when it's done
12	92	GHI	
13	3A	BNZ	. . . if D (thus R(2).1 ≠ 0 then keep decrementing
14	0F		
15	83	GLO	Load our counter into D to check it
16	FB	XRI	"Exclusive Or'ing" it with 09 tells us if they
17	09		are equal (see text)
18	3A	BNZ	If counter ≠ 9, we increment the "ones" digit
19	22		of our scratchpad . . . Otherwise . . .
1A	F8	LDI	. . . we have to reset the counter . . .
1B	FF		(FF+01=00, see text)
1C	A3	PLO	
1D	F0	LDX	. . . move the seconds from scratch pad to D . . .
1E	FC	ADI	. . . and add 07 to simulate base 10 (09+07=10) . . .
1F	07		
20	30	BR	. . . now we branch around the "ones" increment
21	25		
22	F0	LDX	Move our seconds from memory to D . . .
23	FC	ADI	. . . add 1 to D . . .
24	01		
25	51	STR	Restore updated seconds to memory
26	64	OUT 4	Display updated time
27	21	DEC	Once again we have to decrement
28	13	INC	Add one to our counter
29	30	BR	Branch back to repeat for another second
2A	0C		
2B	7B	SEQ	Q on indicates to user that program is stopped
2C	00	IDL	STOP!
2D	00		Scratchpad location

MORE SOFTWARE MAGIC—
TV TYPEWRITER JR.

By Richard Moffie

This program lets you display and type a message of 2 lines of 8 characters per line using only the basic 256 byte memory. Any combinations of 16 preloaded characters can be typed. They are in a 4 x 5 format so that all 16 will fit in 40 byte area. After the 2nd line is filled, the next line begins over the 1st line. Patterns 0–F hex characters are given as well as an alternate with common alphabetic characters. If you have an extra 256 bytes, you can display 5 lines this way, using the entire alphabet.

Note: You must press the INPUT switch after each letter.

HOW TVT JR. WORKS

While this program is limited with only 256 bytes, its use could easily be expanded with minor modifications to provide a complete alphanumeric display without spending a cent for hardware. All that would be necessary is a little more memory—certainly 1K would be more than enough. As it is, a minimum 256 byte system can still type a choice of any 16 characters or patterns at the bottom of the screen as 2 lines of 8 characters each. Each character is in a 4 x 5 grid, so that 5 bytes is enough to store 2 complete patterns.

This is done by putting two pattern lines in a single byte and masking out the half that is not to be used. Bit 0 chooses the left half if zero and the right half if 1. Since this is done by shifting right, the remainder of the input chooses one of eight rather than 16 bytes. this sounds confusing, but examine the codes below:

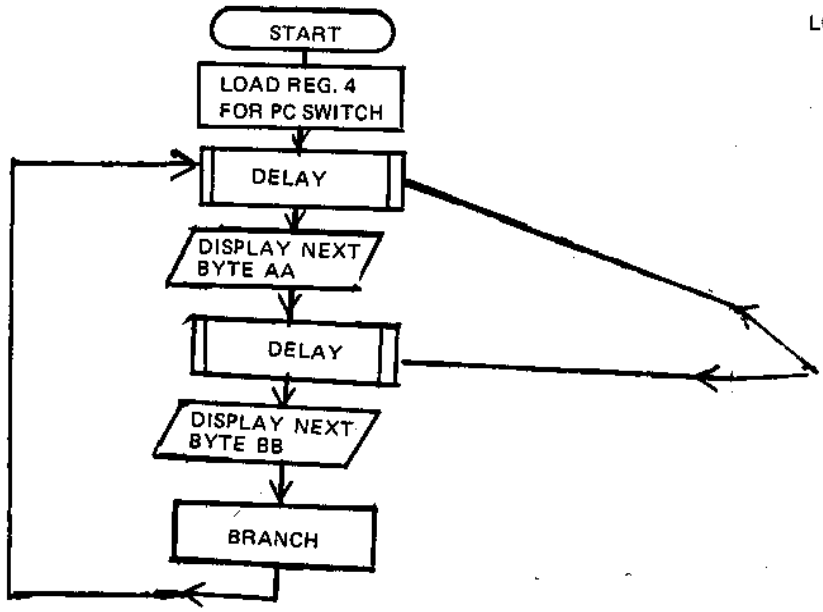
0 = 0000		0 = x000	DF = 0
1 = 0001		1 = x000	DF = 1
2 = 0010	when shifted	2 = x001	DF = 0
3 = 0011		3 = x001	DF = 1
4 = 0100		4 = x010	DF = 0
5 = 0101		5 = x010	DF = 1

Thus DF determines whether to mask the left or the right half of the byte when transferring a pattern to the display area and the remainder chooses which of the 8 bytes in the pattern area is desired.

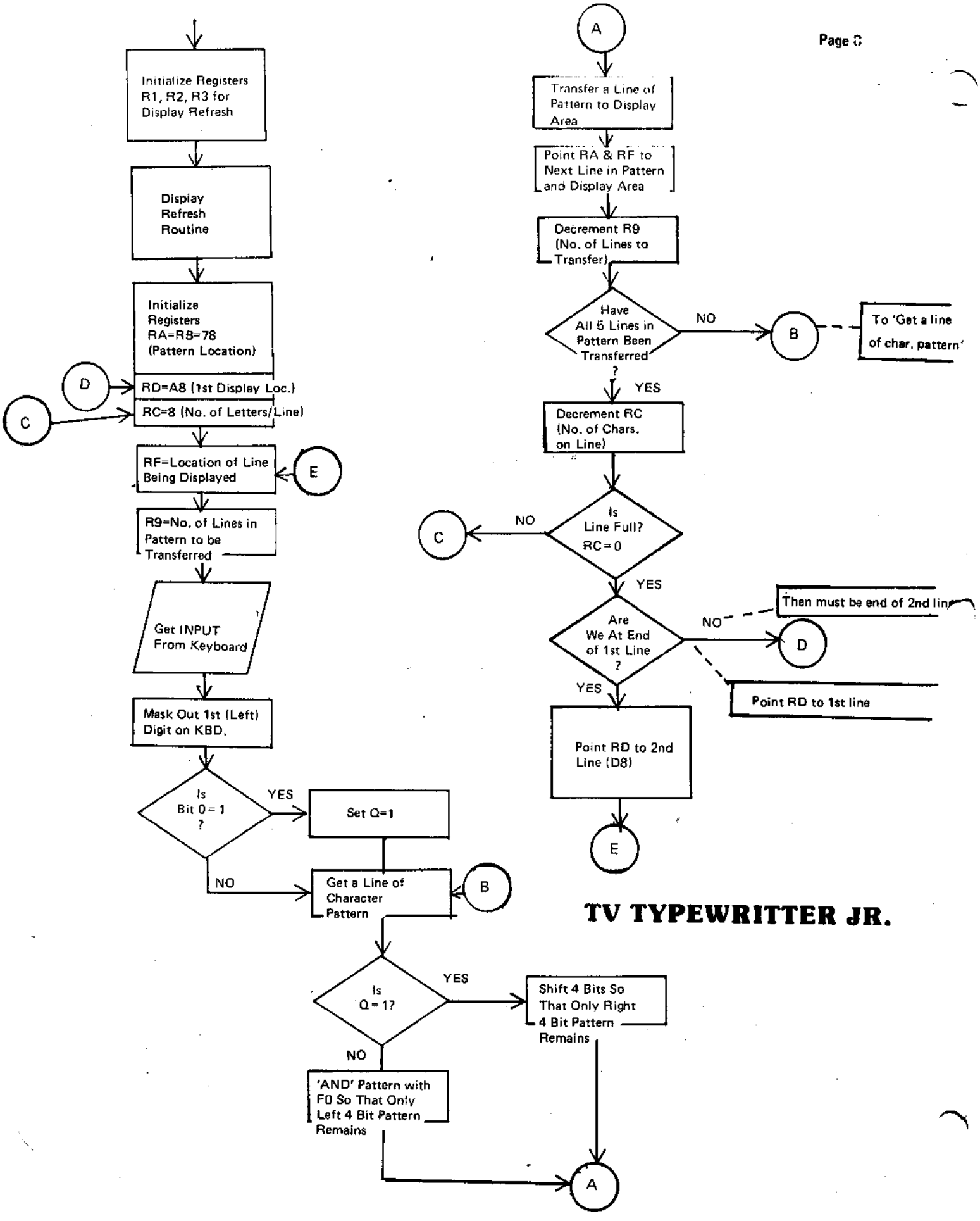
Once the pattern has been selected, the program transfers a byte at a time from the pattern to the next position in the display area, until all 5 lines of the pattern have been displayed. If there is still room on the current line, the program loops back and gets ready for the next input. If the line is full, the program points to the beginning of the other line and begins to type into it, replacing what was already there from a previous typing. By putting desired patterns into bytes 78–9F, any 16 patterns can be displayed in this manner, as illustrated by the totally different examples in the listing.

ANSWER TO EXTRA CREDIT PROBLEM ON PAGE 4

If your answer looks something like the flowchart and listing, you get a gold star in use of the SEP instruction. Notice that the delay in this answer has been put before the display of AA and that there is a branch back to this delay. What you have is a main program (Reg. 0) calling a delay subroutine (Reg. 4). This is accomplished by switching the program counter to Reg. 4. The DELAY routine begins at loc. 0C and has a branch to reset the number in the Reg. program counter so that the routine can be re-exectued.



LOC.	CODE	MNEM.	COMMENTS
00	F8	LDI	Load R(4).0 with starting
01	0C		location of subroutine
02	A4	PLO4	Put D Reg. into R(4).0
03	D4	SEP4	Delay
04	64	OUT 4	Display Immediate next byte
05	AA		
06	D4	SEP4	Delay
07	64	OUT 4	Display Immediate next byte
08	BB		
09	30	BR	Branch to repeat
0A	03		
0B	D0	SEP0	This returns to main prog.
0C	C4	NOP	Not necessary—an example
0D	F8	LDI	Load a delay into high Reg.
0E	FF		
0F	BE	PHIE	Put delay into R(E).1
10	2E	DECE	Decrement Reg. E
11	9E	GHIE	R(E).1 to D
12	3A	BNZ	Test to see if delay up
13	10		loop to 10 if not 00
14	30	BR	Branch to reset subroutine
15	0B		starting location which is 0C



TV TYPEWRITTER JR.

TV Typewriter Jr.

LOC.#	OP CODE/DATA	COMMENTS
00	F8 28 A3	Initialize registers
03	F8 76 A2	
06	F8 0C A1	
09	D3	
0A	72 70	Display refresh
0C	22 78	
0E	22 52	
10	C4 C4 C4	
13	F8 00 B0	
16	F8 00 A0	
19	80 E2	
1B	E2 20 A0	
1E	E2 20 A0	
21	E2 20 A0	
24	3C 19	
26	30 0A	
28	E2 69	
2A	F8 78 AA AB	Initialize registers
2E	F8 A8 AD	
31	F8 08 AC	
34	8D AF	
36	F8 05 A9	
39	3F 39	Get input
3B	37 3B 6C	
3E	FA 0F	Mask out 1st digit
40	F6 52	Select left or right half of display
42	3B 47	pattern
44	7B 30 48	
47	7A	
48	8B F4 AA	
4B	0A 31 52	
4E	FA F0	
50	30 56	
52	FE FE FE FE	
56	5F	
57	8F FC 08 AF	Transfer a line of pattern
5B	8A FC 08 AA	
5F	29 89	
61	3A 4B	Has entire character been transferred?
63	2C 8C	
65	1D 3A 34	Test for end of line
68	8D FF B0	
6B	3A 2E	If not end of 1st line, point to 1st line
6D	F8 D8 AD	Point to 2nd line & get next input
70	30 31	
72	00 00 00 00 00 00	Stack
78–9F	(see upper right)	Character patterns
A0–FF	00	Display Area

Character Patterns for 16 hex characters

78	F6 FF AF FF FF FF FF FF
80	92 11 A8 81 99 95 85 88
88	92 FF FF F1 FF F7 85 FF
90	92 81 21 91 91 95 85 88
98	F7 FF 2F F1 FF 9F FF F8

Sample Character Patterns for Alphabetic characters

These patterns will display the letters: blank, A, B, C, D, E, F, H, I, K, M, O, R, S, T, Y when the keys for 0–F respectively are pressed.

78	0F FF EF F9 49 BF FF F9
80	09 98 98 89 4A F9 98 49
88	0F E8 9E EF 4C D9 EF 46
90	09 98 98 89 4A 99 91 46
98	09 FF EF 89 49 9F 9F 46

Register Assignments

R1, R2, R3 – Display refresh & Stack
R9 – No. of lines in character
RA – Current line in character displayed
RB – First line in character displayed
RC – No. of characters in line
RD – First line in display area
RF – Current line in display area

PRACTICE YOUR MORSE CODE WITH THIS PROGRAM

By Edgar Garcia

This program converts alphanumerics and punctuation into its equivalent in Morse Code so that with a speaker attached to the Q output, you can hear a programmed sequence of Morse Code characters played serially and then repeated. This program is useful in learning the Morse Code which is necessary to receive an amateur radio license. Someone else could load a sequence of characters into memory and then upon execution of the program one could learn the code by listening to it the way it actually should sound rather than by trying to associate the letter with its written "dot-dash" representation.

Register R5 is used as the address pointer for fetching characters located in memory starting immediately after the program. If 00 is fetched from memory, the program starts over so that loading 00 after the end of the message will cause the message to repeat.

Next, the last three bits of the memory byte are examined to see how many code elements are involved in making up each character. For example, A (•–) has two code elements and its byte representation (42) has 010 as its last three bits indicating that the program has to generate only 2 code elements. The period (•–•–•–) has 6 code elements and its byte representation is 56 with the last three bits being 110.

A special case had to be considered for the 4 punctuation marks ?, :, and ". These characters all have six code elements but they all end with a "dot" whereas the period and comma both end in a "dash." The program looks at the first bit of the byte representation and plays a "dot" if it is 0 or a "dash" if it is 1. The byte representation is shifted left and the next bit is examined, etc. until the proper number of code elements have been played. In the case of the above 4 punctuation marks, the sixth code element should be a "dot" but making the last three bits 110 would cause the program to generate a "dash." The problem occurs because the first bit of the last three bits overlaps with the sixth code element in the punctuation code. This is taken care of by making the last three bits 111 for the 4 special cases and then changing the third to last bit to a 0 after the program is told to play only 6 code elements. This trick is accomplished with the last 7 bytes of the program.

Next, the number of code elements is put in R6.0 which counts the number of code elements played. If the last three bits placed in R6.0 are 000, a long pause is generated. This is useful for spacing words. I use a 10 in memory to indicate a space. After the required number of code elements are played for a particular character, a short pause is generated to space between characters.

The tone generated by the program can be changed by changing location 2B, the length of the very small pause between code elements is controlled by location 41, and the length of the long pause between words is controlled by location 45. The length of the "dot" is controlled by location 28 and the length of the "dash" is controlled by location 24.

```
          MORSE CODE
          PROGRAM

    00    F8 00 B5
          F8 57 A5
          45
          32 00
          FA 07
          A6
          32 44
          FB 07
    10    32 50
          25 45
          FE
          B6
          3B 27
          33 23
          26 86
          32 21
          96
          30 14
    21    30 40
          F8 08
          30 29
          F8 02 B4
          F8 04 A3
          23 83
          3A 2D
    31    31 36
          7B
          30 37
          7A
          24 94
          3A 2A
          7A
          F8 10
          30 46
    40    F8 88
          30 46
          F8 FF B3
          23 93
          3A 47
          86
          32 06
          30 1A
    50    25 45
          FA FB
          26
          30 14
```

LEARNING MORSE CODE IS EASY WITH A COSMAC

LETTER	MORSE CODE	BYTE REPRESENTATION
A	• –	42
B	– •••	84
C	– • – •	A4
D	– ••	83
E	•	01
F	•• – •	24
G	– – •	C3
H	••••	04
I	••	02
J	• – – –	74
K	– • –	A3
L	• – ••	44
M	– –	C2
N	– •	82
O	– – –	E3
P	• – – •	64
Q	– – • –	D4
R	• – •	43
S	•••	03
T	–	81
U	•• –	23
V	••• –	14
W	• – –	63
X	– •• –	94
Y	– • – –	B4
Z	– – ••	C4

NUMBER	MORSE CODE	BYTE REPRESENTATION
1	• – – – –	7D
2	•• – – –	3D
3	••• – –	1D
4	•••• –	0D
5	•••••	05
6	– ••••	85
7	– – •••	C5
8	– – – ••	E5
9	– – – – •	F5
0	– – – – –	FD

PUNCTUATION	MORSE CODE	BYTE REPRESENTATION
.	• – • – • –	56
,	– – •• – –	CE
?	•• – – ••	37
:	– – – •••	E7
;	– • – • – •	AF
-	– •••• –	86
/	– •• – •	95
"	• – •• – •	4F
SPACE		10
REPEAT		00

EXAMPLE:

W H A T H A T H G O D W R O U G H T ?

10 63 04 42 81 10 04 42 81 04 10 C3 E3 83 10 63 43 E3 23 C3 04 81 37 10 00

EXPERIMENTING WITH TINY BASIC

This is a brief review of the contents of a document called the TINY BASIC EXPERIMENTER'S KIT; the benefits of using Tiny BASIC are briefly touched upon for those not directly familiar with the language.

Tiny BASIC was not designed to be as powerful and fast as Full BASIC; nevertheless it does have many hidden and interesting features which make it worth further study. Tom Pittman, the creator of the 1802 version of Tiny BASIC, shows how you can use arrays, strings, and floating point numbers with this compact language in his 40 page TINY BASIC EXPERIMENTER'S KIT.

The Tiny BASIC language was designed to take up only 2K of either RAM or ROM memory and yet still be a powerful and effective language. The enterprising programmer can find a way to solve most tasks in this language with only an occasional need to drop back into machine language. When you need a machine language routine for speed or an esoteric function, Tiny BASIC offers a USR function which is a machine-language subroutine call. This USR routine will let you get at nearly every hardware feature in your computer from a Tiny BASIC program, including input and output directly to peripherals.

The resemblance of Tiny BASIC to Full Dartmouth BASIC makes it an easy language for those with any BASIC training to learn. The Tiny BASIC user manual gives a description of all the language features and functions, and includes examples. The LET, IF . . . THEN, GOTO, RETURN, CLEAR, RND and other Tiny functions are easily grasped by those with BASIC experience. Beginners should also find Tiny a good first language; for while it lacks many of the sophisticated special purpose instructions of Full BASIC, it has not sacrificed any of the ease with which a person can learn BASIC.

THE EXPERIMENTER'S KIT is not for beginners since it contains information that adds new capabilities and possibilities to Tiny BASIC.

Tiny BASIC does not have a "peek" or "poke" instruction. Pittman in his EXPERIMENTER'S KIT shows how you can easily perform this feature using the USR function. His examples of how to extend all of the Tiny BASIC features are clear and to the point. There are examples of how to use the REM statement for a character string; and arrays can also be handled in Tiny without too much difficulty in writing but there is some sacrafice in speed with the the use of an array written in Tiny. There is a little known trick to store data temporarily in the GOSUB stack.

The EXPERIMENTER'S KIT takes the reader inside the design and implimentation of the IL (Interpretive Language) itself. This is a fascinating look at how it is all accomplished, and yeilds ideas for more efficiently using the language.

There are also sections in the EXPERIMENTER'S KIT for constructing an ASSEMBLER using Tiny BASIC, and implementation notes. Although the EXPERIMENTER'S KIT appears to be written primarily for the 6800 and 6502 user, its contents are also applicable to the 1802 user. If one wishes to more fully understand Tiny BASIC and utilize hidden capabilities which extend the power of this language; the possession of this 40 page document is recommended. The EXPERIMENTER'S KIT can be purchased for $10 plus 50 cents to cover postage and handling. Checks should be made out to Quest Electronics. California residents should include sales tax with their order.

CHESS+COSMAC=FUN

QUESTDATA would like to hear from those of you who have written chess or chess problem programs. RCA has written a chess like game, hexpawn, and QUESTDATA would like to hear from those who know the whereabouts of this interesting game or who have implemented similar games themselves.

4
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

44807
Steve Brune
408 E. Wadsworth Hill MTU
Houghton, MI. 49931