Questdata

Issue #2 August 1978

WHAT YOUR COSMAC CAN DO – PRESENT AND FUTURE

OUR world is changing. Much of the change we see about us today can be related to the computer. The computer is a tool which can serve almost limitless purposes.

All computers operate in basically the same way; they execute one instruction at a time. It is hard to believe, but nevertheless true that the Elf computer works in the same manner as the big IBM machine and the computer that monitors the flight of 747 airplanes. As Science Fiction author Poul Anderson has said, "We live in Science Fiction times." This is true and it is profound if you stop and think about it.

The computer is no ordinary tool.

To see how the computer differs from one of man's everyday tools, let's use the hammer as an example. A hammer can be used to hammer nails into wood and pry boards and nails out of wood. That is about all a hammer can do. A computer, however, has a whole bag of magical tricks—it can change its function as fast as a new program can be loaded into it. As we have said earlier, big computers and microcomputers have this ability in common.

The present capabilities of the COSMAC system include: TV games and animation; guess the number games using your display LEDs; hexadecimal conversion decimal; timer for telephone calls, darkroom, and chess games; music programs; Morse code keyer; Tiny BASIC; hexadecimal addition programs; and A/D and D/A applications. All of this ability belongs to the COSMAC SUPER ELF, ELF and VIP right now. And as an added bonus, the COSMACS make learning and using machine language fun.

Here are some COSMAC programs that are being worked on and should be here in the near future [**you can be a real part of working on these projects if you so wish**]:

- You can make your COSMAC into a full fledged clock that will turn on your coffee in the morning. You can program an automatic snooze alarm into the program—so that a buzzer goes off, and then the COSMAC waits another five minutes before going off again. The present clock oscillator will be accurate to within 5 to 10 minutes a day. To achieve more accuracy you might choose to use a crystal time base or the 60 cycle wall current to keep time. The power company keeps its cycles per day to clock accuracy.

- A Television Typewriter (TVT) done completely in software. You could link a program to the powerful Chip-8 language which has already been written.
- You can link your COSMAC to A/D and D/A to get really interesting musical tones.

These are only a few ideas, if you let your imagination wander, you are sure to come up with more. When you do, we hope you choose to share your ideas and programs with QUESTDATA readers.

Here are a few more ideas of future and present computer abilities: stop lights, cars, washing machines, microwave ovens, coke machines, typesetting machines, control systems in industry, robots, television sets and studios, digital stereo systems, airplanes, artificial limbs, telephones, radios, computer games, refrigerators, accounting systems, income tax systems, insurance systems, banking systems, satellite systems,

[Continued on Page 2]

railroad switching systems, teaching machines, missile control and defense systems, automatic change-making machines, Cable TV systems that actually communicate with the viewer (What tape would you like to see from the tape library?–Push A, B, or C), automatic field harvesting machines, supermarket checkout machines, automatic inventory machines, automatic vacuum machines (clean your whole house with the push of a button), automatic street-cleaning machines, automatic sprinklers that consider the amount of moisture and how much an individual plant needs for water, daily computerized reminder systems, systems that fix coffee when you get up and warm up the car, automatic control systems so that you cannot possibly get into a wreck, automatic grocery-ordering machines, automatic grocery-stocking and gathering machines, credit card rating systems, instant money credit systems–insert card and based on credit rating–instant cash, automatic control of boats, automatic computerized unloading of boats, medical diagnosis systems, automated X-ray machines which scan for certain types of cells, automated library checkout and inventory systems, automatic translating machines, computerized dictionary, desktop Library of Congress

These are only a few of the possible computer applications that the future will bring.

knowledge computers, computer music–rearrange the notes of a symphony to suit your mood (less violins and more drums, please), automatic identifying machines (if your thumbprint matches a certain digital configuration, a safe opens or you are granted credit or the door to your house opens), automated bartenders–mixes the drinks and keeps track of how much of each beverage has been used, stock market analysis computers, slot machine computers, computerized light shows, individualized manufacturing–custom tooling a particular part for your car (out of a list of thousands of possible tool machining programs in its memory), computerized lie detector systems, computerized cryptography machines, computerized textile and weaving machines, and computerized underwater mining machines, crime prevention computers, and computer ground lenses. These are only a few of the possible computer applications that the future will bring.

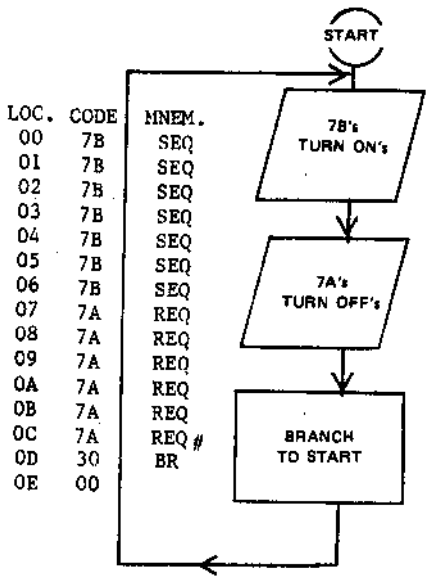
Back issues of QUESTDATA are available (beginning with the July issue) at $1.00 each. Customers in foreign countries, other than Canada and Mexico, please send $.50 additional for airmail with each issue ordered.

The flowchart is not only a way to show documentation of a program, so that others may follow the program; it is also a good tool to use when designing a program or learning to program. With a flow of the program in visual form the programmer can see what he has done and where he wants the program to go. The branch points are made clear by following the arrowed lines. The diamond shapes represent DECISIONS. The computer takes one route or the other at these points. Input and Output are represented by a parallelogram. You can block out large blocks of instructions or individual instructions using flowcharting to give a visual idea of what is going on inside the Super Elf in the MACHINE LANGUAGE section. The next issue of QUESTDATA will feature an article on flowcharting and its importance. Flowcharting is a good documentation, learning and planning tool.

WHAT THE MACHINE IS THINKING

D-REGISTER, INITIALIZATION & COUNTERS

In the last QUESTDATA we examined a simple loop which turned on the Q-LED and made a tone. The frequency of the tone depended upon the number of 7B's (turn on Q-LED) and 7A's (turn off Q-LED) that the loop contains. When we used seven 7B's and six 7A's, the resulting frequency was 8,000 Hz. Here is another look at that loop:



LOOP WITH UNCONDITIONAL BRANCH
EXPERIMENT 5

```
#USING SEVEN 7B's AND SIX
 7A's WILL GIVE A HIGH
 NOTE--8,000 HZ. USE 14 7B's
 AND 13 7A's TO GET WELL
 WITHIN HEARING RANGE--
 4,000 HZ.
```

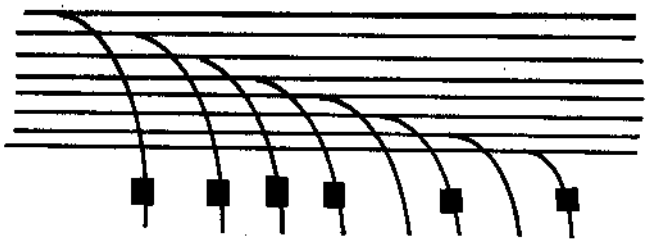
Such a loop is called a closed loop since it will loop forever; or at least until you turn off your COSMAC. Often when professional programmers write a program they will unintentionally create a closed loop. This is an error in logic or a programming bug. The program created in this manner will loop forever in one particular part of the program and never get to execute the remainder of the program. To gain some control over the loop we need to set up a device which counts the number of times that the computer goes through the loop. This is done by setting up a counter.

We are going to pick the **D-Register** (the D stands for data) to be our counter. The **D-Register** is located inside the 1802 microprocessor itself and it looks architecturally like this:

7	6	5	4	3	2	1	0

AN INSIDE LOOK AT THE D-REGISTER

As you can see the **D-Register** is made up of 8 bits or a byte. For a long time computer programmers have used the analogy of a railway switchyard in teaching the concept of registers. Using this comparison we can take the following look at the **D-Register**:



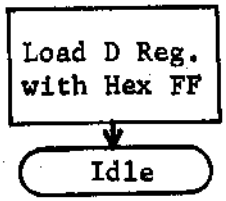
The train configuration in the **D-Register** represents the storage of the hexadecimal number F5. All of the digits in the left four tracks contain a one or boxcar and the right hand four contain—off, on, off, on (the binary coding for 5). The comparison of the 1802's **D-Register** is only an analogy—don't tell your friends that there are little boxcars and switches inside your microprocessor chip . . . little men with nets might come and get you for saying such things.

The concept of what a register is may not come to you right away, it is something you have to work with for a while before you comprehend its exact meaning. Fortunately, you can work with registers while you are learning about them and their capabilities. The way in which you choose to use registers will determine the effeciency and amount of memory you use in your programs.

Many of the early computers had only the **D-Register** or **Accumulator** to work with for temporary storage. The early PDP-8 also had an **MQ (Memory Quotient Register)** but essentially everything was done in the **Accumulator**. The **D-Register** is a very important part of the COSMAC functioning. If you look at a typical listing for a program, you will see that the **D-Register** is used a lot.

How do you get a byte inside the **D-Register**? One way to do it is with the instruction LDI or Load Immediate. What load immediate means is that you put the next byte into the computer's **D-Register**. The following program will load the **D-Register**:

```
LOC.  CODE  MNEM.        ACTION
 00    F8    LDI         LOAD NEXT BYTE
 01    FF                HEX FF INTO D
 02    00    IDL         HALT
```



The **D-Register** is now loaded with the hexadecimal number FF. You cannot see this process because it is not displayed on the display LED's but nevertheless something has happened—the **D-Register** is loaded.

INITIALIZING A COUNTER

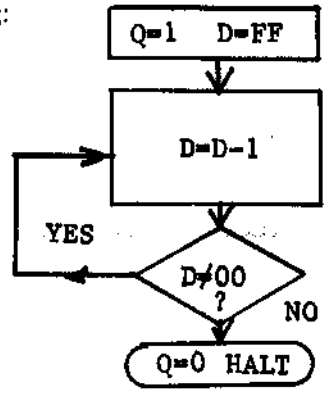
If we wish to use the **D-Register** as our counter to keep track of the number of times a loop is executed, we must first give the counter an initial or starting value. This is the reason we enter the instruction F8 and the data FF to be loaded into the **D-Register**. The **D-Register** now contains the starting value FF in hex or 256 in decimal. The value 00 is usually picked as the stopping value of the counter because there are two COSMAC instructions which can test the **D-Register** for this value (3A and 32). We have a starting value of FF and an ending value of 00 for our loop but how does the **D-Register** change values to get to its final ending value of 00?

DECREMENTING A COUNTER

One way to change the value of the **D-Register** would be to add or subtract something from it. If we subtract one from the value of the **D-Register** each time we go through the loop, the value of the **D-Register** will decrement until it reaches the value 00. When it reaches the value 00 the program will stop looping and continue executing the rest of the program. Programmers would say that the loop is terminated and the flow of the program falls through. We have created an educated loop or DO LOOP with the number of times the loop is done determined by the initial value we put in the **D-Register**.

If we use the initial value of FF for the loop we will go through the loop 256 times. So let's turn the Q-LED on for 256 loops using initialization, counters, and decrementing:

```
LOC.  CODE  MNEM.        ACTION
 00    7B    SEQ         Turn on Q
 01    F8    LDI         load D
 02    FF                with hex FF
 03    FF    SMI         D-1
 04    3A    BNZ         BRANCH NOT 0
 05    03                LOC. 03
 06    7A    REQ         Turn off Q
 07    00    IDL         HALT
```



Let's take a closer look at the device we are using to decrement the **D-Register**. The instruction FF (SMI—Subtract Memory Immediate) on page 30 of the RCA Microprocessor User Manual MPM-201 says that the two operands are subtracted. The D byte represents the minuend (the quantity from which another quantity is subtracted) and the memory byte immediately following the FF instruction represents the subtrahend (the number subtracted from the minuend) in this subtraction operation. So when we issue the code sequence FF, 01, the number 01 is subtracted from the **D-Register**. The results of the subtraction are placed back into the **D-Register**. The operation looks like this after passing through the loop one time:

D-Register	FF
–subtrahend	01
D-Register	FE

The process of decrementing the **D-Register** continues until the contents of this counter are equal to zero. Each time the loop is made counts for one iteration. The word iteration means to do again as you no doubt know from the word reiterate which means to repeat again.

The flow of the machines counting and looping ability is structured as follows:

- Set up or initialize counter register
- Decrement the counter
- Test the counter
- If the counter is not equal to zero then loop again
  If the counter is equal to zero then continue with the flow.

We use a conditional branch to see if the condition we impose (**D-Register** must be equal to zero) is met. When the condition is not true, we continue looping. The flow passes to the next instruction on the computer's agenda when the condition is met (**D-Register** is equal to zero).

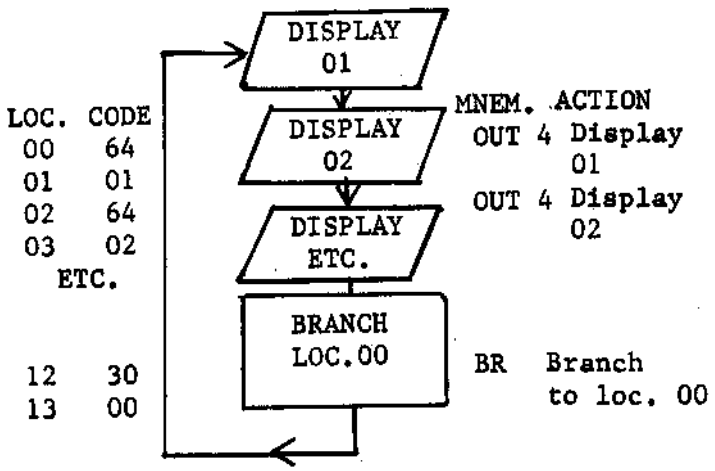
At this point you should be able to program a tone sounder or buzzer. You can make either one—the choice is yours and depends on how many times you want to loop on and off. With the number of iterations up to you—go forth and make a tone . . . You are free to use all knowledge gleaned from QUESTDATA machine language sections. Try it. (The answer is on the next page).

[No fair peeking beyond this point without trying to make a tone on your own first.]

SNEAKY LOADED DICE PROGRAM

There is an interesting way to quickly program dice into your computer. It is easy to remember. Its only drawback is it is not statistically accurate—it will show a slight preference for the number six. Well, you can't have everything; in a board game—if everyone is made aware of this curious fact—you are all off to an even start. Try it just for fun: (Starting at location 00) 64 01 64 02 64 03 64 04 64 05 64 06 30 00.

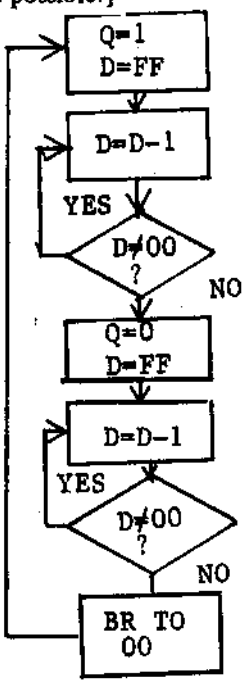
When you want to stop the roll just press reset. The number you have rolled will appear on the right hand LED. To simulate two rolls of the dice—run the program twice. Obviously this is not a fancy dice-roller. How loaded are the dice? You will get 6 two-sevenths of the time; the rest of the numbers will come up one seventh of the time.



The dice work by using the display immediate or 64 instruction. When you turn the run cycle off by resetting the microcomputer, the last number on the display is your dice roll. You don't stand a chance of picking out the number using your reaction time—unless your reaction time is in thousandths of a second.

Your answer should look something like this:
{Note: Other solutions are possible.}

```
LOC.  CODE  MNEM.
 00    7B    SEQ
 01    F8    LDI
 02    FF
 03    FF    SMI
 04    3A    BNZ
 05    03
 06    7A    REQ
 07    F8    LDI
 08    FF
 09    FF    SMI
 0A    3A    BNZ
 0B    09
 0C    30    BR
 0D    00
```




```
ACTION
 Turn on Q
 load D
 with hex FF
 D-1
 BRANCH NOT 00
 LOC. 03
 Turn off Q
 load D
 with hex FF
 D-1
 BRANCH NOT 00
 LOC. 09
 BRANCH
 REPEAT;GOTO 00
```

What you have done is to loop with a count for the Q-LED on for 256 times and loop for 266 times with the Q-LED off. You then branch unconditionally back to the starting location 00 and start the whole process all over again. If you have successfully completed this assignment without peeking, you get a gold star and can call yourself a programmer. If you peeked, well, you must go directly to jail, not pass go, and you do not get to collect $200.

DECIMAL TO HEX CONVERTER

A program for the SUPER ELF which computes the hexadecimal equivalent of any decimal number from 0 to 65535. The programming techniques used illustrate:

- Input/Output using the hex key pad and the display
- Setting-up and indexing through data buffers
- Double byte arithmetic
- Subroutine construction and call
- Logical AND and shifting
- Controlling the program by loops

OPERATION:

The decimal number to be converted must be entered via three successive inputs from the keyboard. Each input consists of two digits followed by the depression of the I (Input) key. The digits are concurrently displayed. The order of the inputs is from the most significant two digits to the least significant two digits of the number to be converted, requiring a total entry of six digits. Therefore, leading zeros are required to pad the decimal number to six digits. The capability is provided to convert only from 0 to 65535 decimal; any other entries will give erroneous results. This range was chosen because the answer is displayed two digits for four total digits of answer. Four digits can encompass a hexadecimal range of 0000 to FFFF which is, of course, 0 to 65535 in decimal, thus giving a limitation. After entering the last two digits of the decimal number and depressing the I key, the most significant two digits of the answer will display and the Q LED will also glow. Depressing the I key then will cause the least significant two digits of the answer to display, and the Q LED will extinguish. Subsequent depressions of the I key will display the answer over and over in the same way as explained above.

To enter another decimal number for conversion one simply depresses the reset key and then the GO key and the program is ready for another entry.

This procedure illustrates the operation of the program:

CONVERT 41155 TO HEXADECIMAL

1. Depress R key
2. Depress G key
3. Enter 04, Display: 04. Depress I key.
4. Enter 11, Display: 11. Depress I key.
5. Enter 55, Display: 55
6. Depress I key, Q LED glows, Display: A0
7. Depress I key, Q LED goes out, Display: C3

ANSWER: A0C3

You may repeat steps 6 and 7 over again, as you wish, to view the answer.

PROGRAM LOGIC:

The program flow chart is given in Figure 1. Two calls are made to a subroutine which is flowcharted in Figure 2. The organization of the program is as follows: 1) entry and branch to main routine; 2) Data Tables; 3) Subroutine; 4) Main routine.

PROGRAM LISTING:

The program listing is organized for analysis as well as entry into your SUPER ELF. The first column on the left contains a number in a circle which can be used to find where the particular location is being referenced somewhere else in the program. The matching reference(s) will appear in the MNEMONIC/REFERENCE column. The next column is the memory location in which is loaded the code, address or data given in the next two columns respectively.

The next column is the MNEMONIC/REFERENCE column previously mentioned. The MNEMONICS used are the set defined in the RCA 1800 Microprocessors User Manual. The last column supplies a description of, or what is accomplished in, the current code or group of codes.

	LOCATION	CODE	ADDR/ DATA	MNEMONIC/ REFERENCE	DESCRIPTION
	00	30	22	BR(6)	Branch around data table and subroutine to start program.
(1)	02	00			Input digit buffer
	03	00			Initial zero value
	04	00			LSD of answer
	05	00			MSD of answer
(2)	06	00			CONVERSION TABLE, used to convert
	07	10			the input to hexadecimal. The
	08	E8			codes in pairs given here are the
	09	64			hexadecimal equivalents of powers
	0A	0A			of ten.
	0B	01			
(3)	0C	00			LOC 0C & 06: 0000: 10^5_4 (Decimal)
	0D	27			0D & 07: 2710: 10^4_3
	0E	03			0E & 08: 03E8: 10^3_2
	0F	00			0F & 09: 0064: 10^2_1
	10	00			10 & DA: 000A: 10^1_0
	11	00			11 & 0B: 0001: 10^0
(4)	12	D0		SEP	SUBROUTINE which computes a two byte temporary answer for each digit of the input. The digit is in the D register right justified.
(5)	13	32	12	BZ(4)	See if the digit is zero, return to main program if it is.
	15	AA		PLO	Store the digit in a scratch register to control repeated addition of current conversion code bytes.
	16	E8		SEX	Perform addition of cummulative
	17	06		LDN	answer with current conversion
	18	F4		ADD	code (least significant byte).
	19	56		STR	
	1A	E9		SEX	Perform the most significant
	1B	07		LDN	byte addition.
	1C	74		ADC	
	1D	57		STR	
	1E	2A		DEC	Decrement the digit

	1F	8A		GLO	by one and go back to the test
	20	30	13	BR(5)	for zero at (5).
(3)	22	F8	00	LDI	MAIN PROGRAM. First, initialize
	24	B4		PHI	the high order byte of the
	25	B5		PHI	registers to point to page 0.
	26	B6		PHI	
	27	B7		PHI	
	28	B8		PHI	
	29	B9		PHI	
	2A	BA		PHI	
	2B	BF		PHI	
	2C	F8	03	LDI	Next, set up register 4 to keep
	2E	A4		PLØ	track of inputs. Only three sets of input digits allowed.
	2F	F8	02	LDI(1)	Now point register 5 to the input
	31	A5		PLØ	digit buffer.
	32	A6		PLØ	Point registers 6 and 7 to the
	33	16		INC	zero initial value and store
	34	A7		PLØ	(initialize) zero in the answer
	35	17		INC	least significant digits and
	36	06		LDN	most significant digits. This
	37	16		INC	code leaves register 6 pointing
	38	56		STR	to the LSD of the answer and
	39	17		INC	register 7 pointing to the MSD
	3A	17		INC	of the answer.
	3B	57		STR	
	3C	F8	06	LDI(2)	Point registers 8 and 9 to the
	3E	A8		PLØ	Conversion Table. Register 8
	3F	F8	0C	LDI(3)	points to the beginning, Register
	41	A9		PLØ	9 to the middle. The initial data contain zeroes for conversion of the first digit.
	42	F8	13	LDI(5)	Load register F with the address
	44	AF		PLØ	of the subroutine.
(6)	45	E5		SEX	This begins the main working loop of the program.
(7)	46	6C		INP4	Wait for I key depression but
	47	64		OUT4	display any input. When I key
	48	25		DEC	is finally depressed, last
	49	3F	46	BN4(7)	data is used for conversion.
	4B	37	4B	B4(8)	
	4D	FA	F0	ANI	Select upper digit of input and
	4F	F6		SHR	shift it to the right hand side
	50	F6		SHR	of the D register.
	51	F6		SHR	
	52	F6		SHR	
	53	DF		SEP	Call the SUBROUTINE to convert the upper digit and store results in the cummulative answer.

	54	18		INC	Point register 8 and 9 to the
	55	19		INC	next conversion data in the CONVERSION TABLE.
	56	05		LDN	Select the lower digit of the
	57	FA	0F	ANI	input.
	59	DF		SEP	Call the SUBROUTINE to convert the lower digit.
	5A	18		INC	Point register 8 and 9 to the
	5B	19		INC	next conversion data.
	5C	24		DEC	Decrement input count by one,
	5D	84		GLØ	when we arrive here for the
	5E	3A	45	BNZ (6)	third time go on to display answer.
(9)	60	7B		SEQ	Display most significant two
	61	E7		SEX	digits of hexadecimal answer.
	62	64		OUT4	Turn on Q LED.
	63	27		DEC	
(10)	64	3F	64	BN4 (10)	Next I key depression causes the
	66	37	66	B4 (11)	display of the least significant
	68	7A		REQ	two digits of the answer. Turn
	69	E6		SEX	off the Q LED.
	6A	64		OUT4	
	6B	26		DEC	
	6C	3F	6C	BN4	Wait for I key again and go back
	6E	37	6E	B4	to (9) to display the MSD of answer
	70	30	60	BR	again.

The program now remains in a loop so that the MSD and LSD of the answer can be displayed alternately with I key depressions.

OTHER APPLICATIONS:

Changing the contents of the conversion table to the values indicated below, causes the program to convert octal inputs to hexadecimals. The input octal range would therefore be 0 to 177777. The conversion values are the hex equivalents of powers of 7:

LOC 0C & 06: 8000: 10^5 (octal)
LOC 0D & 07: 1000: 10^4 (octal)
LOC 0E & 08: 0200: 10^3 (octal)
LOC 0F & 09: 0040: 10^2 (octal)
LOC 10 & 0A: 0008: 10^1 (octal)
LOC 11 & 0B: 0001: 10^0 (octal)

Given the appropriate conversion values, the program also can be used to convert numbers to hexadecimal in other number systems.

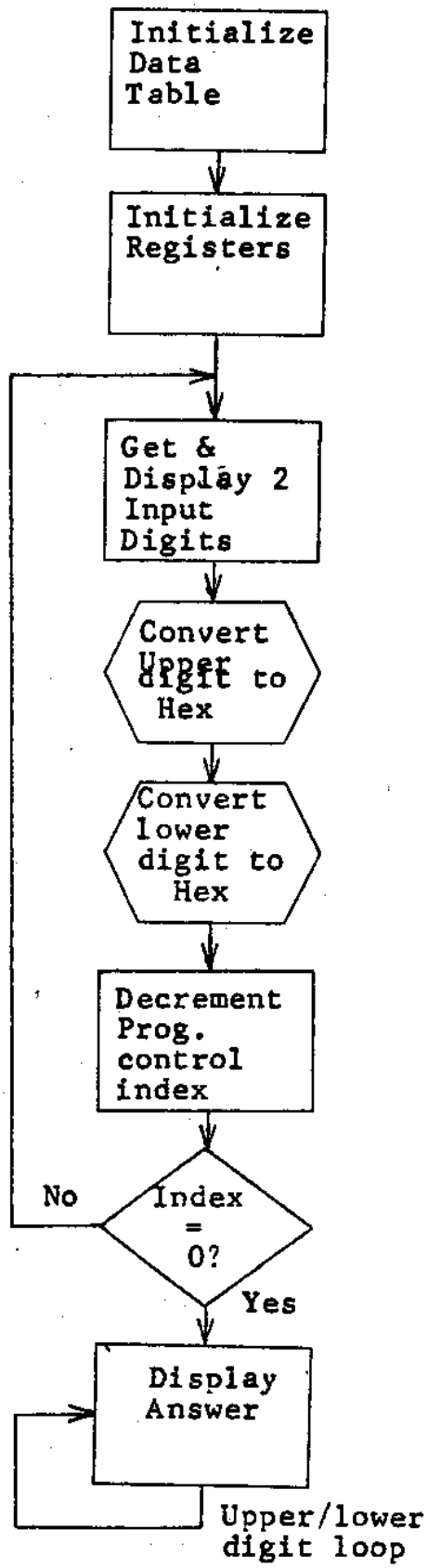


FIGURE 1 - MAIN PROGRAM LOGIC

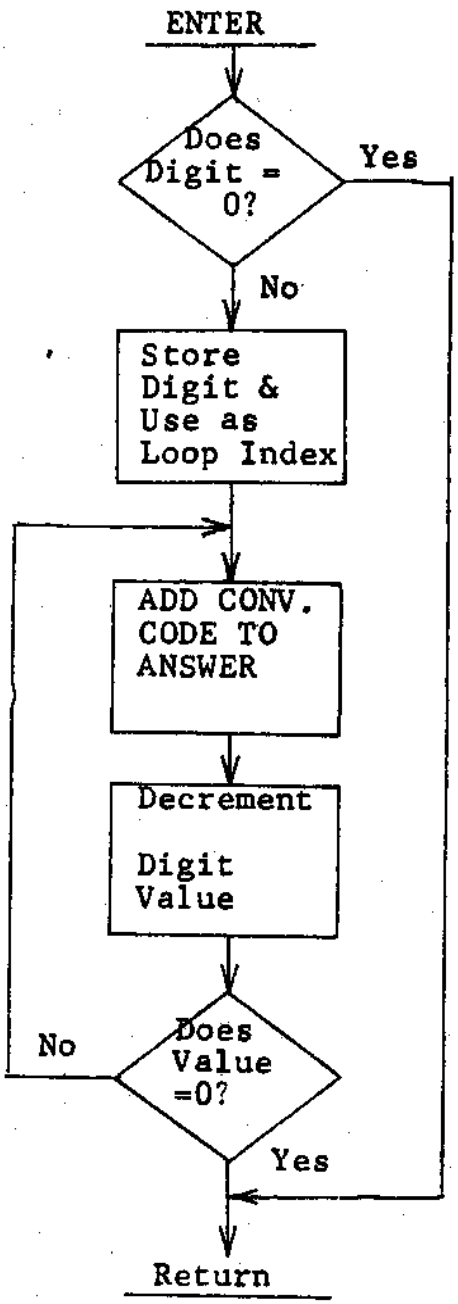


FIGURE 2 - SUBROUTINE

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

Publisher . . . . . Roger Pitkin
Editor . . . . . Bill Haslacher
Programming Assistance . . . . . Pam Gazlay
Proofreading . . . . . Ken Brown

 QUESTDATA welcomes contributions from its readers. Manuscripts will be returned only when accompanied by a self-addressed stamped envelope. Articles or programs submitted will appear with the authors name unless the contributor wishes otherwise. Payment is at the rate of $15 per published page. QUESTDATA exists for the purpose of exchanging information about the RCA 1802 microcomputer. Subscriptions are $12 for this monthly publication.

TOM PITTMAN, TINY BASIC & COSMAC PART II

In which Tom reveals some of the COSMAC's features . . .

How does the 1802 rate against the Z-80? Everyone calls the Z-80 a supercomputer, but Pittman does not think this is necessarily so. "Everyone thinks that the Z-80 is 'this big fantastic computer,' " he says. "Let's just take your plain vanilla 1802 and Z-80; running at their respective top speeds. How long does it take the 1802 to CALL a subroutine? 2½ microseconds. How long for the Z-80 to execute a CALL? 8½ microseconds." Actually, according to Pittman, the COSMAC microprocessor begins to really show its excellence about the 5th time the homebrew routine is used. The conclusion that one draws is that if you use very many subroutines at all, the RCA 1802 is faster.

"Let's take the fantastic MOVE BLOCK TO OUTPUT instruction of the Z-80." Tom says he can write an output routine to work faster in the 1802 language, due largely to the two 1802 capabilities:

- The speed of changing the **Program Counter** is faster than the Z-80 CALL.
- The OUTPUT on the COSMAC increments the **X-Register** automatically and this feature makes the routine faster even if it is called only once.

The time comparison is: 7½ microseconds RCA, 8½ microseconds Zilog. And the more you use the routine the bigger the payoffs because the PC switch is faster than a JUMP. The fact is that on most instructions the RCA 1802 is comparable to the Zilog Z-80. On some instructions the Z-80 wins A BYTE TRANSFER or BLOCK MOVED INTO ANOTHER MEMORY LOCATION will take 8½ microseconds while a short 1802 routine will execute the same thing in 12 microseconds. This is the biggest beating the RCA 1802 takes, and this is not much states Pittman. Remember that the RCA 1802 will gain on the Z-80 each time a routine is called. In short, the RCA is fast enough to satisfy even the worst speed freak. But the COSMAC 1802 has other benefits as well.

Probably the worst feature of the 1802 is its lack of relative addressing? Wrong. According to Pittman, all COSMAC owners have a form of relative addressing with page (relative) addressing. Also, Tom Pittman says it is easy to write a short routine to simulate relative addressing. Tom says this is part of the powerful pseudo-codes you can build with the RCA 1802. In effect you can build your own language. COSMAC is a micro for the do-it-yourself type or those who say, "Gee Zilog, I'd rather do it myself."

The 1802 writes the most compact programs. Tom figures the average bytes per executed instruction of the 1802 to be 1.5. He at first figured the time for the 1802 to be 1.3, but then allowed a generous amount to setting-up or initializing. "Setting-up is an easy procedure once you become used to it," Pittman says. He estimates the instructions per byte of the 8080 at 1.9 bytes per instruction and the 6800 at 1.7 bytes per instruction. He does not even consider the 6502 worthy of comment since he found it took 15% more code to write Tiny BASIC with this microprocessor. He just couldn't tune it to the 2K he wanted—a frustration which he remembers with great clarity. And then there is the gravy part of the RCA 1802 microprocessor.

The fact that the COSMAC is CMOS and can run on microwatts is all so much frosting on the cake. Also to be considered are: only one voltage (and it is variable at that), a 16 by 16 register matrix, a simple clock, four EF flags, three N-Output lines, a Q-line with latched output, moveable **Program Counter** and **Output Immediate**. According to Tom Pittman the 1802 is the only microprocessor with true built-in **Direct Memory Access (DMA)**.

Aren't there any other CMOS microprocessor chips out? Yes, there is one. . .the Intersil 6100 chip which is nothing more than the PDP-8 minicomputer in micro form. "OK, there is one reason for having the 6100, and that is if you have lots of PDP-8 software for your particular application." Pittman considers writing software for the outmoded architecture of the 6100 to be a silly thing to do. "Why take a step back in computer history," he reasons. It is interesting to note that he considers micro's in general to be a technology superior to the PDP-8 minicomputer which came into existence in the middle 1960's; and sold for $50,000 and up when peripherals were included. Pittman declares microcomputers to be an advanced generation of computers. They form a new class.

True, if you are used to working with a mini, it is difficult to get into the mindset of a micro but once you do, then, according to Pittman, you are far better off for it. In the final analysis, Pittman emphasizes, "the 1802 is not a difficult machine to understand, if you make a study of its architecture."

Tom has taught computers using the Microtutor 1801 to show students of the University of California Extension how to program. It is easy to teach when your students all have immediate access to a computer," Tom says. He comments, "With computers you can't just book learn—you have to run programs." For example, you start by writing a simple program—00. The 00 program causes the COSMAC to idle. The next program (7B; 00) turns on the Q-LED. The 7B turns on your Q-LED and the 00 brings the computer to an idle. By learning the effects of one instruction at a time, the students quickly come to grasp programming.

Flowcharting? "I don't use flowcharts when I program," Tom says. He warns, "When you program without a flowchart, you need to avoid the tangle of code which comes with a lot of branches back into the program. Beware strange branches." Tom Pittman classifies his programming as an informal structured approach. "Actually, I write from the bottom up," he comments. What does Tom think of Knuth?

(Donald E. Knuth; Stanford professor and author) "Knuth is very good, I plan to own all of his books." So far Knuth has written three of a proposed seven books. The books sell for $20 each and weigh enough to break a bookshelf; Knuth's books are recommended only for those who are very sincere and have sturdy bookshelves.

Pittman likes to do all of his program writing in Tiny BASIC when writing 1802 programs. He has even written an assembler using the tool of Tiny BASIC. The user of Tiny Basic can use the Utility Programs UT3/4 or make his own Input and Output patches to Tiny BASIC. RCA's Evaluation Kit and Development System use these Utility Programs. Readers wishing to know more about the Utility Programs provided in RCA Kits may should consult RCA manual MPM-203 (Evaluation Kit) or MPM-101 (Development System). When ordering RCA's books; Tom's Tiny BASIC machine code listing, ROM or cassette, be sure to include the tax if you live in California.

Yes, you see the holes in the tape are an art form.

The State of California considers programming taxable. "If I send Tiny BASIC to you via data coupler there is no tax." But if you order it by mail, you must include tax. When you order paper tape, what you are really paying for is what you don't get; the holes. Ha, ha," Tom Pittman laughs at the ironies of of today's society. Is programming copyrightable? "Yes, you see the holes in the tape are an art form. They are totally unique," he says.

Some checking into copyright laws shows Tom Pittman up-to-date on developments. Briefly, there is a new copyright law effective January 1, 1978 which is extremely comprehensive. It gives the owner of the copyright the exclusive right to reproduce, distribute, display, or prepare derivatives from the copyrighted works. The new law covers works "fixed in any tangible medium of expression now known or later developed, from which (it) can be perceived, reproduced or otherwise communicated, either directly or with the aid of a machine or device." This new law covers magnetic tapes, ROMs, core memory, and disk storage. The law clearly covers the copying of a programmer's work into any tangible form or media and effectively covers programming rip-offs.

When you order Tiny BASIC you get a free extra, a text editor. You see, each line of Tiny BASIC is automatically numbered. "All you have to do is start typing your text instead of the Tiny BASIC codes," Pittman states.

Pittman thinks that the COSMAC 1802 is closest in design to the PDP-11. He says that the 6800 advertises itself to be the closest to the PDP-11 (an advanced generation of minicomputer). The reality, Pittman has found, is that the 1802 is far closer to the PDP-11 than the 6800. So why doesn't everyone use the RCA COSMAC? "It isn't the style of programming that people are used to and microcomputer sales are mostly a marketing thing. The company with the best marketing ability has the edge. Also, there is a lot of existing software written for the 8080."

"With computers you can't just book learn–you have to run programs."

Is there a Full BASIC written and being used for the 1802? "No, at present there is not." There are utility programs, subroutines and even an equivalent of the Television Typewriter (TVT) humming away on certain computers. But, yes, there is no Full BASIC. Since Pittman likes to write nearly everything in Tiny BASIC, instead of the COSMAC assembler, he begs the question, "Would you prefer Tiny BASIC to Full BASIC if it were available?" "No," says Tom, "because Full BASIC has some very powerful string capabilities."

Tom is very honest about the Tiny BASIC, as he appears to be with everything you might ask him. When asked why he recommended getting the most powerful BASIC you can lay your hands on in his Kilobaud article on Tiny BASIC, he will tell you, "Because it is true, why settle for less than the most powerful tool on the market." Pittman says that he was amazed by the number of requests which came in after an article in which an author compared all of existing versions of BASIC.

The 1802 is a do-it-yourselfer's microcomputer. Its 16 registers and switchable PC facilitate this ability to build your own routine and aid the creation of compact code. The 1802 is certainly not for everybody. Counter arguments to the 1802 such as, "the 1802 is not a great number cruncher," have validity. There is more to Tom Pittman and the 1802 chip than this brief handshake with both can reveal. Pittman programs many different microcomputers but feels a special fondness for the RCA 1802. The 1802, as Tom will readily admit, is not the last word in microcomputers—so you can hang on to all those other chips you were about to toss.

[Note: Many of the ideas in the column WHAT THE MACHINE IS THINKING in QUESTDATA issue number one were derived from conversation with Tom Pittman. QUESTDATA would like to acknowledge and thank him for his help.]

VIDEO GRAPHICS SOFTWARE PART I

[This is the first in a series of articles on video graphics for your 1802/1861 Elf type computer. The articles start with a detailed discussion of the program article on video graphics which was featured in the July 1977 issue of *Popular Electronics.* Later in this series more sophisticated techniques will be discussed such as animation, etc.]

WHAT IT DOES

When the program listed in FIGURE 3 is entered into the Super Elf and started, the TV screen will light up with a seemingly random collection of light and dark squares. The upper quarter of the screen is actually a display of the program itself, since the display consists of the data in locations 0000 to 003F. With the program up and running, observe the eighth "line" from the top, the rightmost row of twenty-four squares. If a lighted square is a "1" and a dark square is a "0", then this row has a pattern of: 0110 1001 0010 0011 0110 1001. Enter via the hex keypad the code 40 with an input depression and release. Now note the same row, the pattern is now: 0100 000 0010 0011 0100 000. This is the stack area of the program and will be explained in detail later in this series of articles.

The entry of 40 is the starting location of data to be displayed on the non-program area of the screen. The next entry from the hex keypad will then fill that location. The subsequent entries will be displayed on the display as on and off patterns. In other words the first hex number entered after pressing restart and run is the starting location of your on and off pattern with the on and off pattern to follow future hexadecimal inputs. These entries are made 8 hex numbers to the line. Try entering 8 AA's and 8 55's. The screen will now display two lines of a checkerboard pattern; try other codes. By entering various codes (00 through FF) any pattern can be "drawn" on the screen. Enter these codes, each line of code is a line on the screen:

LOC	CONTENTS
40	00 00 00 00 00 00 00 00
48	F0 88 F0 F0 F0 0F 08 0F
50	80 88 90 80 90 08 08 08
58	80 88 90 80 90 08 08 08
60	F0 88 F0 E0 F0 0E 08 0E
68	10 88 80 80 C0 08 08 08
70	10 88 80 80 A0 08 08 08
78	F0 [illegible] 80 F0 90 [illegible] 0F 08

FIGURE 3 VIDEO DISPLAY PROGRAM LISTING

LABEL	LOCATION	CODE	DATA/BR	MNEMONIC/REF	COMMENTS	FUNCTION
START	0000	90		GHI (0.1)	R1.1, R2.1=00	PROVIDE BEGINNING
	0001	B1		PHI (1.1)		ADDRESS VALUES
	0002	B2		PHI (2.1)		FOR THE REGISTER
	0003	B3		PHI (3.1)	R3.1, R4.1=00	AS REQUIRED FOR
	0004	B4		PHI (4.1)		VIDEO HARDWARE IN
	0005	F8	2D	LDI	R3.D=(MAIN)	THE SUPER ELF
	0007	A3		PLO (3.0)		
	0008	F8	3F	LDI	R2.0=(STACK)	
	000A	A2		PLO (2.0)		
	000B	F8	11	LDI	R1.0=(INTERRUPT)	
	000D	A1		PLO (1.0)		
	000E	D3		SEP 3	P=(GOTO MAIN)	NOTE: NEXT ADDRESS
RETURN	000F	72		LDXA	restore D, R2+1	IS 002D
	0010	70		RET	restore XP, R2+1	[To be explained in detail
INT.	0011	22		DEC (2)	R2-1	in the next QUESTDATA]
	0012	78		SAV	save XP, @ M2	
	0013	22		DEC (2)	R2-1	
	0014	52		STR (2)	save D@ M2	
	0015	C4		NOP	no-op delay (9 cycles)	
	0016	C4		NOP		
	0017	C4		NOP		
	0018	F8	00	LDI	re-initialize ptr high	
	001A	B0		PHI (1.1)		
	001B	F8	00	LDI	R0=0000 (refresh ptr)	
	001D	A0		PLO (0.0)	re-initialize ptr low	
REFRESH	001E	80		GLO (0.0)	D=R0.0	
	001F	E2		SEX (2)	8 DMA cycles (R0+8)	
	0020	E2		SEX (2)		
	0021	20		DEC (0)	R0-1	
	0022	A0		PLO (0.0)	8 DMA cycles (R0+8)	
	0023	E2		SEX (2)		
	0024	20		DEC (0)	R0-1	
	0025	A0		PLO (0.0)	R0.0=D	
	0026	E2		SEX (2)		
	0027	20		DEC (0)	R0-1	
	0028	A0		PLO (0.0)		
	0029	3C	1E	BN1	go to refresh (EF1=0)	
	002B	30	0F	BR	go to return (EF1=1)	
MAIN	002D	E2		SEX (2)	X=2	
	002E	69		INP 1	turn TV on	
	002F	3F	2F	BN4	wait for IN pressed	
	0031	6C		INP4	set MX, D, R4.0=keypad	
	0032	A4		PLO (4.0)		
	0033	37	33	B4	wait for IN released	
	0035	3F	35	BN4	wait for IN pressed	
	0037	6C		INP4	Set MX, D=keypad	
	0038	54		STR (4)	set M4=D, R4+1	
	0039	14		INC (4)	R4+1	
	003A	30	33	BR	go to M33	

[more on page 14]

SPIRIT OF '76

By Bob Richie

The "Spirit of '76" is a music program which sounds something like a jazz version of **The Flight of the Bumblebee.** It has a playing time of 4 minutes and 18 seconds. The "Spirit of '76" is built around an oscillator program which starts at address 13 and ends at 22. Addresses 18-1A are not part of the oscillator: it is rather like a pacer or tempo keeper which is controlled by the value at address 0E. The oscillator gets its pitch value at address 01, but not directly. The value at address 01 is first decremented and then stored in R4. From there it is brought out of storage, treated with a "76" SHRC and put back into R4. A value may be treated nine times with a "76" before repetition begins. (See address 06-0C). When a value has been 76'ed nine times a freshly decremented value is given to R4. Try replacing the 76 with other logic or ALU codes. Try changing the tempo (address 0E). Bet you can see why the original program is named the "Spirit of '76"? Yes, it's because the program demonstrates audibly the essence of the op code 76 SHRC.

LOC.	CODE	DATA/BR	MNEM.	COMMENTS
00	F8	00	LDI	load starting value
02	A1		PLO (1.0)	of osc. into R1.0
03	21		DEC (1)	new value of osc.
04	81		GLO (1.0)	D=R1.0
05	A4		PLO (4.0)	R4.0=D
06	F8	0A	LDI	set timing value
08	A2		PLO (2.0)	R2.0=D
09	22		DEC (2)	R2-1
0A	82		GLO (2.0)	D=R2.0
0B	32	03	BZ	BR to 03 if time up
0D	F8	08	LDI	load tempo
0F	B3		PHI (3.1)	R3.1=08
10	84		GLO (4.0)	D=R4.0
11	76		SHRC	shift & prog. name
12	A4		PLO (4.0)	R4.0=D
13	7A		REQ	Q=0
14	84		GLO (4.0)	D=R4.0
15	A5		PLO (5.0)	R5.0=D
16	25		DEC (5)	R5-1
17	23		DEC (3)	R3-1
18	93		GHI (3.1)	D=R3.1
19	32	09	BZ	go to 09 if R3.1=00
1B	85		GLO (5.0)	D=R5.0
1C	3A	16	BNZ	go to 16 if R5.0=00
1E	31	13	BQ	go to 13 if Q=1
20	7B		SEQ	Q=1
21	30	14	BR	go to 14

CLOSE ENCOUNTERS THEME

By James C. Nicholson

Following the short "music program" listed in the February 1978 issue of *Popular Electronics*, the theme for **Close Encounters** may be played using the following data:

LOCATION	DATA	ACTION
AC	7519	B♭
AE	2D4C	rest
B0	8315	C
B2	2D4C	rest
B4	681D	A♭
B6	2D4C	rest
B8	3443	A♭
BA	2D4C	rest
BC	4E2A	E
BE	2D4C	rest
C0	7519	B♭
C2	8315	C
C4	681D	A♭
C6	3443	A♭
C8	4E2A	E
CA	00 [illegible]	repeat

It is suggested that changing location 77 of the program to the value 06 (hex) will produce a tempo closer to reality. To run the program on the Super-Elf, enter the data 3068 into locations 00 & 01. Push RESET, RUN.

Coming in the next Issue:

- Flowcharting Secrets
- Fiendishly Clever Star Machine Program
- Video Graphics Software Part II
- Machine Language Experiments
- Reader Submitted Programs
- Some Surprises and Much, Much More

Video continued...

The initialization section starts from location 0000 and and ends with the SEP instruction at location 000E. The function of INITIALIZATION is to start the program and take care of register assignments explained in more detail later. The INTERRUPT subroutine starts at location 000F and ends at location 002C. It is correct to call this a subroutine but it is special in the sense that it is not "CALLED" by the MAIN PROGRAM. In fact, it executes upon demand ("INTERRUPT") of the video generator and its job is solely to provide the location of data for display. The MAIN PROGRAM spans the memory locations from location 002D to 003B. Its job is to obtain a starting location of PAGE 0 memoryfrom the hex keypad. After that, keypad entries become data entered into PAGE 0 which are then picked up and displayed. The fourth division of the program is the stack area which functions as a temporary or "scratch" area of memory used in the functioning of the MAIN PROGRAM and INTERRUPT SUBROUTINE both.

INITIALIZATION ANALYSIS

Refer to FIGURE 3 to follow this analysis of INITIALIZATION. The program was written for a MULTI PAGE MEMORY (MORE THAN ¼K) version of the Super Elf. Therefore, to keep the program and display on MEMORY PAGE 0, the first five instructions provide PAGE 0 addresses to the four scratch registers used in the program. It does this by obtaining the data value 00 from the high half of REGISTER 0. This is a convenient way since REGISTER 0 is set to the 000 address at reset. Once 00 is obtained by means of the GHI instruction in location 0000, and stored in the D Register, it is copied into the high halves of Registers 1, 2, 3 and 4 by means of the PHI instructions in locations 0001, 0002, 0003, and 0004 respectively.

The RESET button on your Elf causes the PROGRAM COUNTER to start (or initiate itself) as REGISTER 0. Since the video generator itself must use REGISTER 0 it is necessary to switch (SEP) the Program Counter (PC) out of REGISTER 0. REGISTER 3 has been chosen as the new PC. This is accomplished by the instruction in location 005, and the LDI, which causes a fetch from location 0006, the value of 2D into D and subsequently copied into the low half of REGISTER 3 via the PLO instruction in location 0007. The job will not be finished until the SEP instruction is executed in location 000E. Meanwhile, the preparation for reassigning the X Register designation is done. As you know, X is currently referring to REGISTER 0; for the later read instructions of the keyboard, which use the X reference into the stack, another register must be chosen. If not, the video generation would "clobber" REGISTER 0 the first time it functioned. The Register chosen was REGISTER 2 since it is also used by the INTERRUPT SUBROUTINE. This choice is dictated by efficiency considerations and the use of the stack. The address of the stack is 003F and the LDI instruction in location 0008 obtains 3F, as required, from location 0009 and loads the low part of register 2 with that value by the PLO instruction in location 000A.

The next few locations now assign an address value to REGISTER 1 which, as mentioned above, is the referent of the INTERRUPT function. This is done so that REGISTER 1 will "point" to the address for the TV display. That portion of memory in fact is the INTERRUPT SUBROUTINE, thus its location (or "entry point") must be specified. The LDI instruction in location 000B obtains the value 11, stored in location 000C, into D and the PLO in location 000D completes the set up of the INTERRUPT SUBROUTINE entry address. As mentioned above, the next location contains the SEP instruction needed to switch the PC reference from REGISTER 0 to REGISTER 3. After this instruction executes, the next instruction from the location in memory pointed to by REGISTER 3 (which in this case is location 002D) is fetched.

Next month we will discuss the INTERRUPT SUBROUTINE and associated display timing requirements.

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

BUG SQUASHER PAGE

At the bottom of Page 3 in this issue the program should read: (Caught just before mailing this QUESTDATA)

LOC.	CODE	MNEM.	ACTION
00	7B	SEQ	Turn on Q
01	F8	LDI	load D
02	FF		with hex FF
03	FF	SMI	Subtract
04	01		D=D-1
05	3A	BNZ	Branch not 00
06	03		to loc. 03
07	7A	REQ	Turn off Q
08	00	IDL	HALT

There is a very similar bug (from the same bug family) on Page 5 in the upper left hand corner:

LOC.	CODE	MNEM.	ACTION
00	7B	SEQ	Turn on Q
01	F8	LDI	load D
02	FF		with hex FF.
03	FF	SMI	Subtract
04	01		D=D-1
05	3A	BNZ	Branch not 00
06	03		to loc. 03
07	7A	REQ	Turn off Q
08	F8	LDI	load D
09	FF		with hex FF
0A	FF	SMI	Subtract
0B	01		D=D-1
0C	3A	BNZ	Branch not 00
0D	0A		to loc. 0A
0E	30	BR	BRANCH
0F	00		to loc. 00 Repeat to get buzzing sound

There was also a bug lurking in QUESTDATA number 1. Reader J. Jacikas describes the bug: There seems to be an error in the listing of the "HEX ADDITION WITH ADDENDS AS DATA" (Page 10) program. If there is no overflow, the codes in locations 0A and 0B cause a conditional short branch to location 2E instead of to the end of the program. If location 2E contains a 7B instruction, you will get a false overflow indication. Changing the contents of location 0B from 2E to OD seems to fix the problem.

We thank our sharp eyed readers for their help in squashing bugs and your comments are always welcome.