Questdata

©

Issue #1

QUESTDATA
"YOUR 1802'S BEST FRIEND"

THIS newsletter is your publication. We are now gathering the many good ideas you have been sending us about your COSMAC and reading them for publication possibilities. Quest, suppliers of the Super Elf, regular Elf and COSMAC VIP want you to get the most out of your system. We have found it necessary to form this publication in order to better serve your needs. We will also honor the contributions of owners of other RCA 1802 systems.

QUESTDATA will be a monthly publication and will cover the subjects you need to know in order to keep your COSMAC well fed with data and hardware nourishment. Your knowledge and ideas will make a great contribution to other computerists and give you a chance to see your words in print. We don't have any special requirements for submitting your thoughts. Write us a letter and if we print it we will pay you a fee at the rate of $15.00 per page published. Don't worry about spelling and how you say the stuff—we have a proofreader to worry about it. If he lets too many errors slip by we chop off his head. So this is your forum.

Most of all we think you want programming. You want good documentation also so that you can modify our programs to suit your needs. How do you design a certain program that is your hearts desire? We will show you. Send us your ideas and we will work on them and announce them to others so they might help out.

Programs are like puzzles, the more insight you get into working them the better. For this reason we are going to show you all 91 machine language instructions and the knowledge of how they work. We will give lots of examples in machine language. So stay tuned for the continuing series: WHAT THE MACHINE IS THINKING.

The COSMAC systems are built to grow with your needs and knowledge of computers. With problems such as hexadecimal addition (page 10), you will get a chance to use the hex numbering system to perform useful functions. The hex chart is included in the addition experiments so you can put a hex on all your friends. Happy COSMACing!

All of the programs in this issue of QUESTDATA will run on Super Elf, Elf, Elf II or COSMAC V.I.P. systems. The timing of the clock will make a difference in the THREE MINUTE TIMER on page 10. The location and information for adjusting this timing are given in the TIMER article. Since the V.I.P. does not have a display you will not be able to use the display immediate part of the MACHINE LANGUAGE section. In future issues of QUESTDATA we will discuss the differences between the various COSMAC 1802 systems.

Coming in the next issue:

- Tom Pittman and Tiny BASIC Part II
- Machine Language Experiments 11 thru 20
- Decimal or Octal to Hexadecimal Conversion Program
- Video Graphics Software Part I

Coming in future issues:

- The Super Elf Expansion Board
- Using the Expansion Board on Your VIP, Elf II, or other system
- Video Games
- Interfacing Tiny BASIC
- And much, much more

IN THE BEGINNING. . .
Computers Past and Present

Where did the microcomputer get its start and where is it going? The COSMAC Elf story really starts with the birth of the big computers. Studying the big oldies is your key to perspective and insight into microcomputers. This tour of ours will take you along a computer memory lane filled with brilliant men and stunning innovations. You will see how your COSMAC was born—or rather evolved—from big IBM parents. Yes, your Elf owes it all to the big mainframe computers which paved its way. If, however, you do not care to join this exciting tour of computer evolution, you may go directly to the QUESTDATA programing section and plug the given programs into your Elf.

The secret to understanding the nature of the computer lies in the study of the simple on-off switch and its brother, the relay. The computer is a machine which combines and stores many on-off codes. Your machine operates in binary. Bi means two. Hence we have a number system with only two numbers; an on (logical "1") and off (logical "0"). By combining these simple on-off patterns, and filing them away in a large matrix storage area, we are able to contain both the program instructions and data. We Americans call this storage area "memory." It is interesting to note that the British refer to "memory" as "storage." They feel "memory" is best reserved for the human mind. Any way you look at it, a computer stores things (1's and 0's) and recalls them.

It is memory, working together with CPU (1802) logic that yields the results to what we feed our Elf. The speed of today's computer owes much to the speed of the electrons which travel through the wire veins of all computers. This speed is close to the speed of light. The speed of electricity is 300,000,000 meters per second. Thus, when you turn on a light switch in a dark room, it seems like the lights come on almost instantly and, indeed, it is very fast. But not fast enough to please some people who are experimenting with lower temperature (freezing temperature) to get the speed even faster within specialized circuits. The effect the lower temperature has is to reduce the friction of the electrons moving within the circuits, thus allowing smoother operation. Just as friction in a mechanical motor is a problem, so it is with electrical circuits. Some day, maybe, the speed of the electron may prove to be a limiting factor in computers but it is fast enough to suit our purposes.

You can, if you wish, build a computer around mechanical or water pressure principles. Remember the old mechanical Monroe calculators? They would take as long as a minute to do a long division problem. You may laugh at the water or hydrocomputer if you want, but the Russians, for a time, took them very seriously. Given a system of valves (a valve being an on-off switch) you can transmit information at a fast rate. And, instead of an electrician, you could call a plumber to fix them. But with what plumbers charge these days . . .

The most surprising thing that has happened in the last 50 years, in the opinion of many, is the fact that computers have shrunk in both size and cost by an amazing amount in the last five years. Five years ago, computers were big beasts that occupied rooms. Today computers with the power of an IBM 360 system cost the same as a good hi-fi system. Instead of occupying a small room, 16K (K=thousand) bytes of memory can be held in one hand. In the old days, 16K was considered a lot of memory (or at least enough for most applications). Today, a megabyte (million bytes) is not an unusual amount of memory to find in a system.

Memory seems to get filled, no matter how much you have. It is like living in a house—the bigger the house, the more stuff you find to fill it. The fact of the matter is that to program the 256 bytes of Elf memory will take a professional programmer at least a week. It takes many months to program something like Tiny BASIC (which occupies 2K). The Tiny BASIC program makes it possible to program your computer more easily (in an English like language) and pays its way by freeing the program to think about the problem instead of the machine language.

If you wish to understand microcomputers and learn the basics of their jargon, you should visit your nearest library to see a copy of the September 1977 issue of *Scientific American*. This fantastic issue is devoted entirely to the microcomputer and the new age of computers which is to follow.

It is unfortunate that at this time computers are still a cult which is deeply embedded in a computer jargon that even the experts can't get together on. For example, the word "microprocessing" has two meanings. It refers to the microprocessor chip that we have in our Elf and also to something that existed long before the microprocessor came into existence. In its other sense IBM coined the word to mean "the way basic codes are put into the IBM 360 computer so that it will recognize the machine code instruction set."

FROM MAINFRAME TO MICRO

The theoretical principles underlying digital computers were first given by Charles Babbage in 1833. His invention of the computer or analytical engine had to wait another hundred years to achieve reality, however. This Englishman's machine is now on display in a British museum. Despite its historical importance, the machine never worked. Some say that if Babbage had worked with a two's system instead of a ten's system, it would have worked during his lifetime. But, as it was, the technology was not up to the exacting machining of parts which he required. This made him very angry—he died frustrated and pennyless. The story can be read in detail in the book **Babbage, The Irascible Genius.**

You would be upset too if your computer didn't work. In this respect, Babbage has a lot in common with today's computer hobbyist, who is often frustrated and most certainly pennyless after his investment in microprocessor, memory, input and output. A friend of this writer who recently bought a hobby computer explains, "It is not the computer that costs you money, the microprocessor is cheap, its the (explicative deleted) input and output that will get you."

We live in a world which fulfills Babbage's dream.

Anyhow, Babbage knew he was on to something big and he once said to a friend that he would give ten years of his life to come back and see the world of the future with a guide to explain things (and he meant it). We live in a world which fulfills Babbage's dream.

John Von Neuman, in the 1930's, developed the stored program concept at the Institute for Advanced Studies at Princton University and since then electronic computers have evolved into the machines we know them as today. One of those who worked on the first computers with Von Neuman was Alan Turing. He showed with the Universal Turing Machine that with a long strip of paper for memory and a piece of cardboard for a scanner, and with a dial to keep track of the instruction you are on (Program Counter), you are in possession of a paper and pencil computer. With just three directives, or instructions, for the computer of paper and pencil, you can get the computer to do anything that any other computer can do. In other words, all computers are created equal. The belief that Turing machines are adequate to perform any numerical or symbolic algorithm is known as Church's Thesis, after the logician Alonzo Church.

Alonzo Church's Thesis cannot be proved mathematically because it cannot be stated rigorously, but it has stood the test of time. No one has been able to express an algorithm that Turing's machine cannot perform. The catch to the Turing machine is that it takes time and tape (for memory). Indeed, it takes yards of tape just to do a simple mathematical problem.

Computers over the last 50 years have come a long way. They started out using relays for storage (memory) and registers. This system sounded like a bunch of ladies with knitting needles. To store something in memory you activated the appropriate relay and you were in business. The bit (standing for binary digit) was either on (logical 1) or off (logical 0). But this system cost a lot and though one relay by itself might be fast, when you get a room full of them all tugging away at their magnetic coils to pull down the ones and zeros, you have a slow system. Also, when you have this many relays you need a very large amount of power to drive them.

The next thing to come along to function as a computer was the electron tube. Tubes could also store ones and zeros, and were a lot faster than the relay. However, when you get that many tubes together you have quite a power problem and you also have a lot of tubes to replace all the time. At least one tube every fifteen minutes bit the dust in these big old systems.

With the advent of core memory, we really had something. Now by magnetizing the ferrous material in one direction you got a zero and by magnetizing it in the other direction you got your binary one. This was a very big step forward in speed. When you sense a core to tell which way it is magnetized, it can be sensed in nanoseconds (billionths of a second). The core memory, however, takes up a lot of room and takes a lot of equipment to maintain the system. Core memory also tends to be expensive compared to the new semiconductor systems.

Today we have RAM (Random Access Memory) which is 100% semiconductor in nature. They are very small, you can hold an array of 8 bits by 1,000 locations in the palm of your hand.

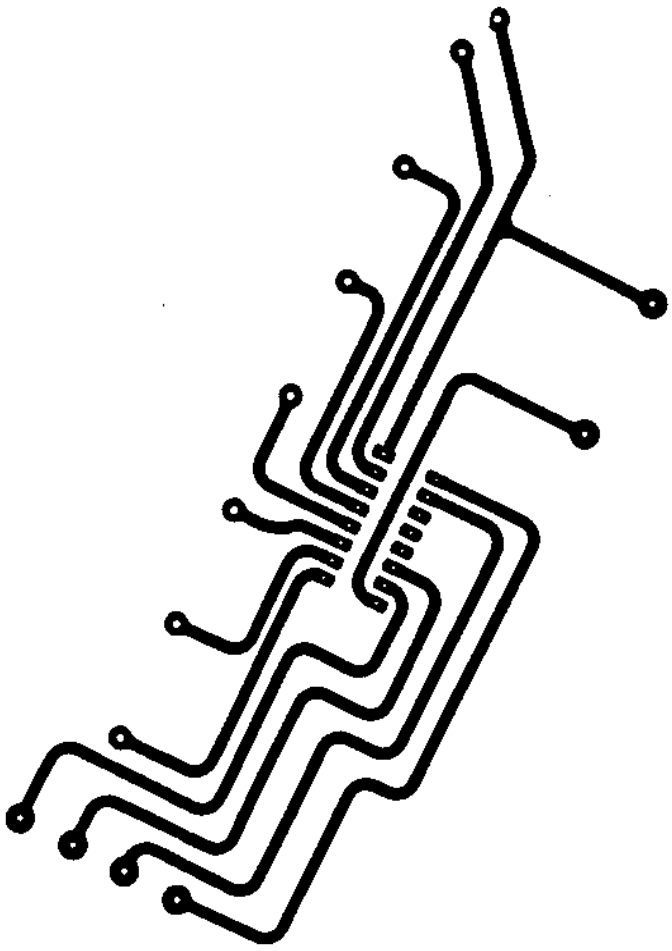
The Central Processing Unit (CPU) which performs the logical functions upon the stored memory program, has undergone the same dramatic size reduction as semiconductor memory. At the heart of our COSMAC systems is a register array of 16 bits by 16 bits; these are temporary storage places which we can save the starting addresses of CALL routines and use as scratchpads to temporarily hold data. This register array is composed of a lot of flip-flops similar in kind to the flip-flops which make up RAM memory. The microprocessor Integrated Circuit (IC) forms the

WHAT THE MACHINE IS THINKING

brains of the COSMAC system. The 1802 IC chip contains thousands of transistors which act upon the RAM memory. The RAM is like a list of things to be done: It is the microprocessor which carries out and steps through the list of instructions to be performed.

At least one tube every fifteen minutes bit the dust . . .

When reading the COSMAC microprocessor User Manual, MPM-201, you will find the many logical and register operations that are carried out by the microprocessor IC. Deciphering what an instruction actually does is a lot like reading an income tax form. The instruction list contains very exacting and precise definitions of what the microprocessor does with the memory it reads. It is an amazing number of operations when you consider the old relay computers would take a large size room to contain the same "brain" that is now contained in the 1802 IC chip.



Is this a crazed arachnid, a printed circuit or art? Send us your ideas.

MACHINE LANGUAGE

In the days of big computers, programmers would sometimes have to wait days to get their programs back from the processing center. With a microcomputer, you can see the result of your program as soon as you run your computer. Thus you have instant feedback since you can recall exactly what you did in most cases and are able to see the effects of your programming almost instantly. Gone is the trying experience of guessing what you did and why you did it to a program, often days later in the case of big computer installations.

The QUESTDATA approach to programming mastery is geared toward building a vocabulary of machine language. By seeing how each instruction affects the output, you will see how programs are built. In programming, there are many ways to accomplish the same task. You will develop your own programming style as we progress through the 91 COSMAC instructions. Certain patterns or blocks of instructions repeat themselves with great frequency. The addition of one instruction, at this point, will show how the new instruction affects the output. In later lessons we will build what programmers call subroutines or CALLS out of lists or blocks of instructions which occur with great frequency.

EXPERIMENT 1

Enough talk. Grab your Elf and punch in the hexadecimal code 00. Now press the run button. The Elf does nothing, right? This is because 00 tells it to idle (the computer equivalent of twiddling its thumbs or revving its engine out of gear). Actually the machine *is* doing something, it is waiting for an Interrupt or DMA request, but since we are not using these hardware devices at the moment, we consider the computer to be just idling.

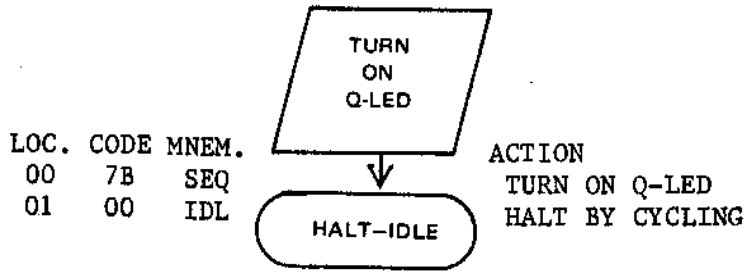
```
LOC.  CODE                    MNEM.  ACTION
 00    00     HALT-IDLE       IDL    CYCLES ON ITSELF
```

EXPERIMENT 1: IDLE

Now what can we make of this instruction? If we put it at the end of a program, the machine will go through its list of machine language things to do and then it will come to a rest or idle. If we left it to chance, there is a possibility that the computer could do something strange and unpredictable since, when the Elf is turned on, a bunch of random data fills its memory. This is like putting a period at the end of a sentence so that two sentences will not be read as one. So much for Experiment 1.

EXPERIMENT 2

The instruction 7B turns on the Q-LED. The Q line is a latched output. This means that the Q-LED will remain on until it is told through another instruction (7A) to turn off. We can also test the Q line to find out whether it is on or off with a one byte instruction –but that's another experiment.



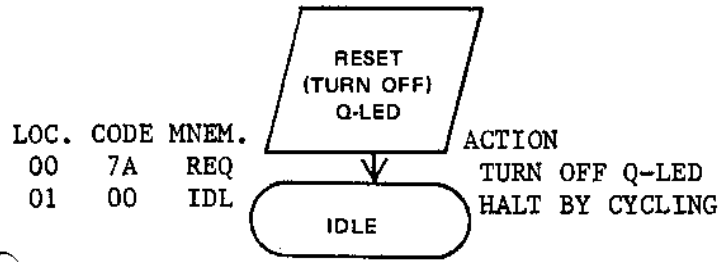
7B=TURN ON Q-LED

EXPERIMENT 2

When we put 7B into the computer and follow it with 00 we get a lit LED (gives you a feeling of power to command your computer to do something, doesn't it?) If your Q line was attached to a relay or solenoid, with a little external circuitry, you could tell your computer to turn on your TV or do some form of work. All things are possible with a micro. (Well, almost all.)

EXPERIMENT 3

The next experiment is a bit disappointing, after the last. Another nothing? Actually, it is not a nothing – it just looks like nothing. You have told the computer to turn off its Q-LED with the 7A instruction. The Q-LED is started or initialized (in programmer lingo) at the beginning of every run to be off. So you see a doubly off LED since both you and the micro (as part of its oun housekeeping – initialization) have turned the computer off. What happens, you are probably asking, if you put in 7B and quickly follow it with 7A. Well, try it. You have just seen how fast your machine really is. If your clock is running at 1.7898 MHz. (millions of cycles per second) your LED flashes on for 9.088843 millionths of a second. This is so fast that you can't see it. Oh well, onward.

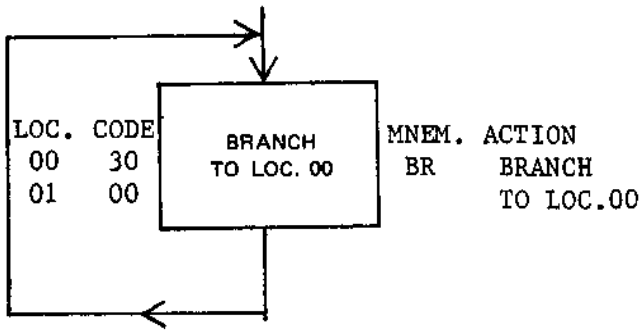


7A=TURN OFF Q-LED

EXPERIMENT 3

EXPERIMENT 4

But what if we loop back around and continually execute 7B followed by 7A? Then we might have something. This is what the 30 instruction will do. It is an unconditional branch. What this means is that it doesn't ask any questions, it just branches. If you are starting your program at 00 and you wa it to branch back to 00, you just put 30, 00 into your computer and it will do this. In other words, the thing following 30 is not an instruction but a memory location. This is slightly different than the instructions we have come across so far. This is called a two byte instruction and in this case it expects to find two hexadecimal digits following the 30 command. Thus, in this case the 00 does not mean idle but go to location 00. In the basic Super Elf we have 256 bits or one page of memory. The 30 unconditional branch instruction branches within this page. It is a short branch since it stays within 256 bits (FF in hex). There are long branch instructions which will take you out of the page you are in but only after you buy more memory for your Super Elf. (Some people still persist in believing that the Long Branch is a saloon. Oh, well...)

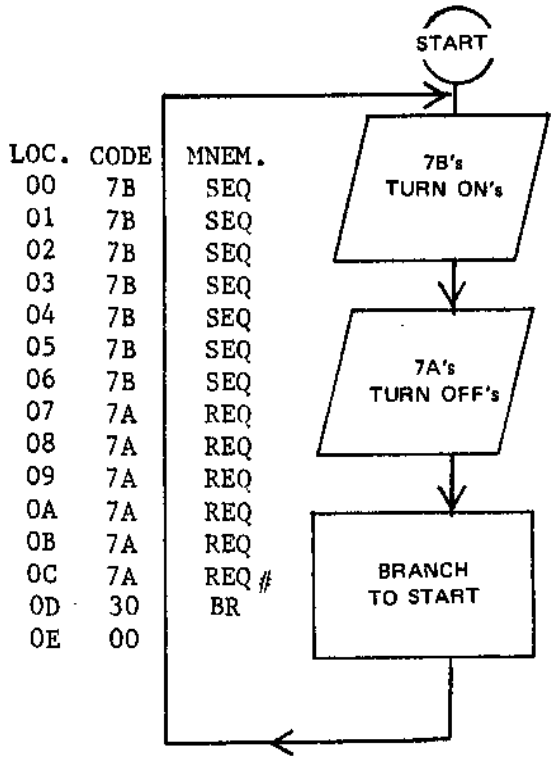


UNCONDITIONAL BRANCH

EXPERIMENT 4

EXPERIMENT 5

So let's put in a whole bunch of 7B instructions (like about seven) and a whole bunch of 7A instructions (like about six) and a 30 and a 00 and see what we get. On the Super Elf you will get a tone. If you are using the monitor, you can put in 7B's and 7A's and branch back to location 21 (30 followed by 21). So why do we get a sound? The reason we do is that sound is vibration and a bunch of ons and offs can make a tone. The more 7B's and 7A's you have, the lower the note you hear will be. Try it. The Q-LED will give you a visual look at the frequency of the on-off cycles.



LOC.	CODE	MNEM.
00	7B	SEQ
01	7B	SEQ
02	7B	SEQ
03	7B	SEQ
04	7B	SEQ
05	7B	SEQ
06	7B	SEQ
07	7A	REQ
08	7A	REQ
09	7A	REQ
0A	7A	REQ
0B	7A	REQ
0C	7A	REQ #
0D	30	BR
0E	00	

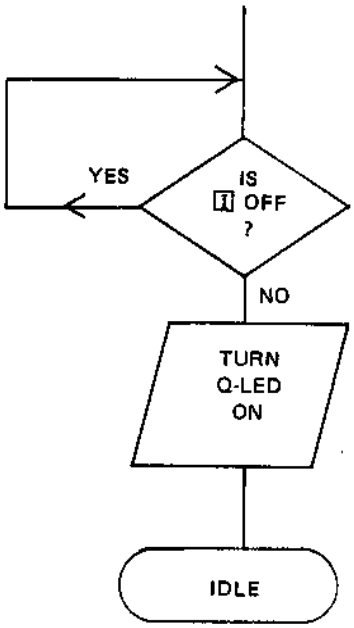
LOOP WITH UNCONDITIONAL BRANCH
EXPERIMENT 5

#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.

EXPERIMENT 6

So if 30 is an unconditional branch, what is a conditional branch? A conditional branch asks a question. On the basis of the answer to the question, the computer makes a decision. The decision it makes is where to go next. The instruction 3F asks the question: "Is INPUT off (INPUT button not depressed)?" If the INPUT button is not depressed, the memory location following 3F will be read as the next location to go to. Now if INPUT is depressed, the 3F instruction will tell the computer to skip over its next location (the one containing the other branch choice). The best way to see this is to try the following example: 3F, 00, 7B, 00.

After keying in the instructions and running the program (not using the monitor), you will find that nothing happens until you press the INPUT button and then the Q-LED comes on. This is the second experiment with a frill added to it.



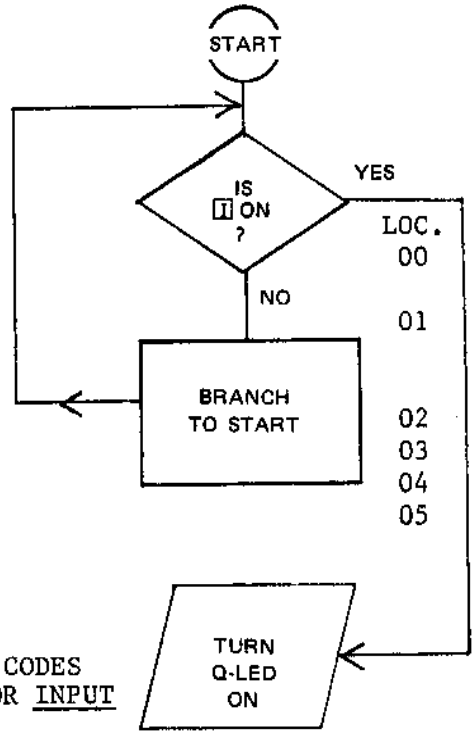
LOC.	CODE	MNEM.
00	3F	BN4
	3E*	BN3*
01	00	
02	7B	SEQ
03	00	IDL

CONDITIONAL BRANCH
EXPERIMENT 6

*VIP USES THESE CODES
PUSH ANY KEY FOR INPUT
TO RUN ON VIP

EXPERIMENT 7

There is another way to ask about the condition of the INPUT button (up, depressed); we can ask if the INPUT button is depressed. Although this seems to be a more direct way to verbalize the question, it is a two byte longer program. The reason for this is that we have to branch unconditionally back to our starting location, if we use this approach. Try it: 37, 04, 30, 00, 7B, 00.



LOC.	CODE	MNEM.
00	37	B4
	36*	B3*
01	04	
02	30	BR
03	00	
04	7B	SEQ
05	00	IDL

*VIP USES THESE CODES
PUSH ANY KEY FOR INPUT
TO RUN ON VIP

CONDITIONAL AND UNCONDITIONAL BRANCH
EXPERIMENT 7

EXPERIMENT 8

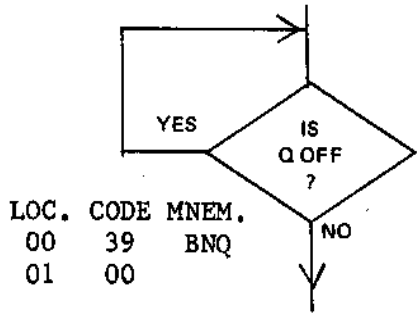
In Experiment 2 we turned the Q-LED on and said that we could have the computer sense (or test) whether the Q-LED was on. This probably seemed like a pretty silly question for a computer to ask itself but computers are pretty dumb. The experiment we are now going to try will turn the Q-LED on the first time you press the INPUT and off the next time you press the INPUT button. Here we go: 3F, 00, 37, 02 39, 09, 7A, 30, 00, 7B, 30, 00. If you wish to use the Super Elf monitor, since this is a longish experiment, its: (starting at location 21) 3F, 21, 37, 23, 39, 0A, 7A, 30, 21, 7B, 30, 21.

Why the inclusion of testing for both INPUT depressed and INPUT not depressed? This is done for debouncing of the INPUT switch. Since the microcomputer operates in micro-seconds, there is a chance that the INPUT button will bounce or chatter when it is released. By asking both questions (Is it on, off?) we make sure INPUT bounce does not interfere with things.

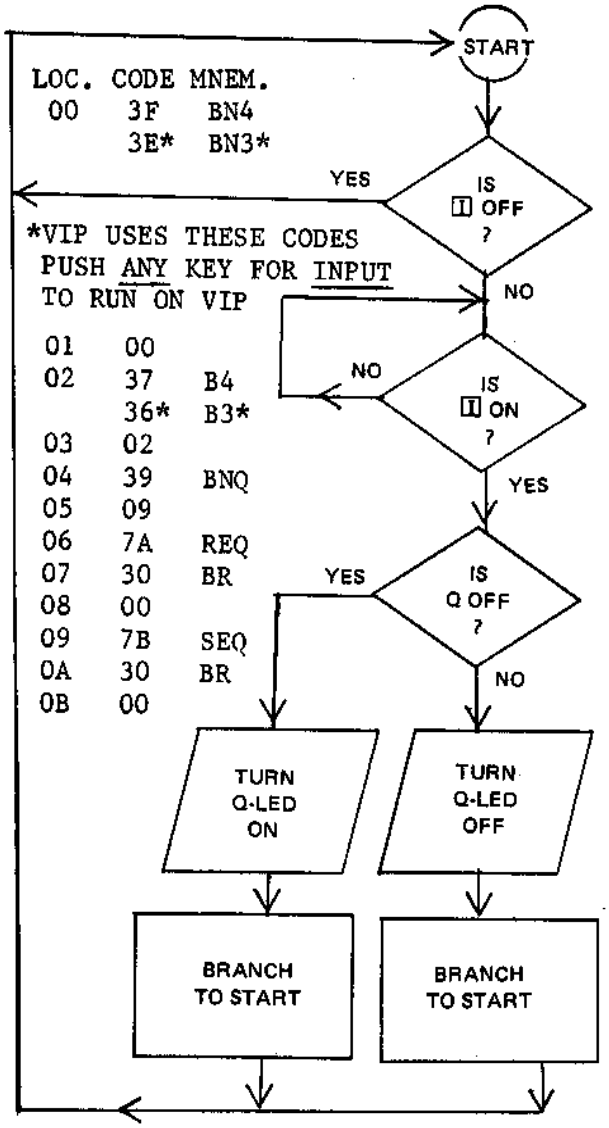
The COSMAC has a really unique and neat instruction. It is **display immediate. Display immediate** is not the normal mode of output for the COSMAC—**register indirect** is. We will be covering **register indirect** in the next issue of QUESTDATA; so don't worry about it now. If you have come this far you have done enough pondering for one day. Anyhow, **display immediate** is when the **P-register** is equal to the **X-register**. Since the **P-register** starts out as 0 and the **X-register** starts out as 0 (more micro startup housekeeping—initialization). When we are at the start of a program **P=X** since 0=0. So we start out in the **display immediate** mode of computer thinking (if computers can be said to think). The **P-register** is the **Program Counter** and the **X-register** is an **index register**. Since we don't need to worry about registers at the moment, we won't. You have already used a type of register, the Q-latch is a register. The Q line (register) stores only one thing (on=1 or off=0). It is an unusual register in that it talks to the outside world directly. So don't let registers scare you. You have had register experience and qualify for the job of register person. More about registers in the next QUESTDATA. Stay tuned.

EXPERIMENT 9

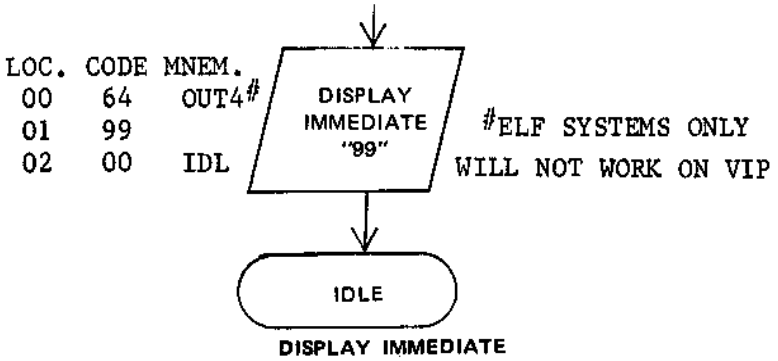
So—**display immediate**. What it means is that when **P=X** the micro reads the next thing (location) as output to those seven-segment LEDs. Thus, since the output instruction for a Super Elf and Elf is 64, the thing following the 64 gets its name in LED lights. Try it: 64, 99, 00. Push run and what do you get? 99. Aren't computers obedient?



TESTING Q's CONDITION
EXPERIMENT 8A



TURN Q-LED ON THEN OFF
EXPERIMENT 8B



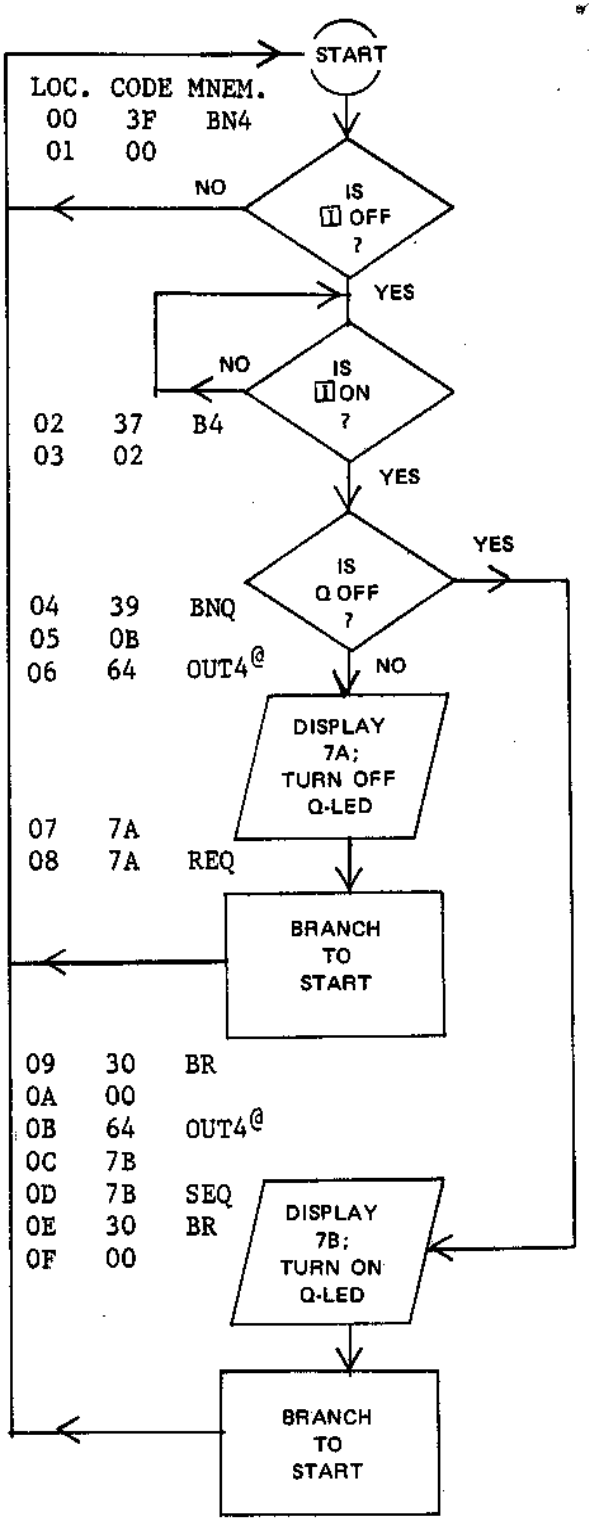
DISPLAY IMMEDIATE WHEN P=X
EXPERIMENT 9

EXPERIMENT 10

Let's do the last experiment; the one where we pushed the INPUT and the Q-LED came on and then when we pushed it again it went off, again; but with a new twist. This time when it is on we will display 7B and when it is off 7A on the seven-segment LEDs. Here goes: 3F, 00, 37, 02, 39, 0B, 64, 7A, 7A, 30, 00, 64, 7B, 7B, 30, 00. Using the monitor on the Super Elf: (starting at location 21) 3F, 21, 37, 23, 39, 2C, 64, 7A, 7A, 64, 7B, 7B, 30, 00. The first 7A in our program gets displayed and the other one is the command that does its 7A thing (turning off the Q-LED). Pretty good, the Elf tells you what it is thinking or doing.

Congratulations! You have just tried out some of the whistles and bells of the Elf and given it some programs to think about. Computers have an insatiable appetite for programs, so we have presented some more in the PROGRAMMING section. Future QUEST-DATAs will include more programs which have been written for the QUESTDATA CLUB. Send us your programs and ideas and see your name and programs in print.

Try variations and innovations on the stuff in this monthly feature. If you come up with a good innovation we would like to hear about it. For review, pick up the instruction manual 201 and read over the instructions you have learned. Try new things on your own. Have fun. Bye for now.



TURN Q-LED ON – DISPLAY 7B
TURN Q-LED OFF – DISPLAY 7A
EXPERIMENT 10

@NOTE: VIP DOES NOT HAVE LED DISPLAY

Welcome to the COSMAC club! If you own an Elf, Super Elf, Elf II, COSMAC VIP, COSMAC Development System, CDP18S020 Evaluation Kit or Homebrew 1802, you are one of us. At the heart of our systems beats the RCA 1802.

We will bring you all the latest news each month concerning your COSMAC. We feel that there is a need for 1802 owners to communicate with each other in a quality monthly publication. You will see lots of programs and information about your microcomputer. We will feature a continuing series on machine language. TINY BASIC will be covered in detail. Each issue will be from ten to twelve pages.

Cost for this monthly publication will be $12 for a years subscription. Our software and hardware people are here to help you. This is your publication and we sincerely hope to support and grow with your needs. So if you haven't already ordered your subscription, fill out the entry form on page 11 or use a seperate piece of paper and sign up. We COSMAC's must stay together.

TOM PITTMAN, TINY BASIC & COSMAC PART I

Tom Pittman—an interview

"The 1802 is the best microprocessor—bar none," Tom Pittman enthusiastically states.

So who is Tom Pittman and why is he saying such nice things about the 1802? Tom Pittman writes software for a living; he writes for all kinds of computers and has nothing to gain from preferring one system over another except a better (faster, more compact) program.

Pittman wrote the original assembler for the Intel 4004. "People still come up to me, even today, and remember me as the one who wrote the 4004 assembler," he says with a smile. The 4004 is a four bit processor, as compared to the COSMAC 8 bits. Take 4004 and double it and you get 8008, the next generation Intel gave birth to. So mix these numbers around and you get 8080, the next step up the microprocessor family tree. So Tom is an early pioneer from the days of 1972 but he is also a leader in the cause of software vendors everywhere.

Tom Pittman could easily make his living writing software for just the industrial or commercial user. But Tom would like to find a way of helping the computer hobbyist. He wanted to test the claim that if you make software cheap enough, it will not get ripped off. Tiny BASIC for the 6800, 6502 and 1802 is the result of this effort. Although this has effectively stopped rip offs, the effort has not been altogether successful in proving to be a viable market. Why is selling software not all that successful? At present, Pittman cites a number of factors which limit the software market. One is that the personal computer market has not matured enough to recognize the importance of software, and there is not the demand for it. More important is the fact that good software is expensive to produce. Good software is time consuming to write; it is a known fact that the average programmer output is one line of debugged and documented code per hour, so an 8K 8080 program written in assembly language represents two man-years of labor, if done right. By comparison, he finds that hardware design is a piece of cake (he knows—he has done quite a bit of that, too). It is the pondering of such questions as "How can programming be made to pay its authors?" and "How can microcomputer hardware and software be standardized?" which are, perhaps an even greater contribution to the hobby computer community than his software.

If you wish to read more of Pittman's pioneering effort to get more software for hobbyists, you should write to *People's Computers*, 1263 El Camino Real, Box E, Menlo Park, CA 94025. The article to ask for is "FREE SOFTWARE? or Support Your Local Software Vendor."

The Tiny BASIC which Tom wrote for the 1802 is shorter in coding length than the Tiny BASICs for other microprocessors now in the marketplace. That is quite a pat on the back for the COSMAC. Let's explore the development of Tiny BASIC to find out how and why it was written.

Tiny BASIC was the brainchild of Bob Albrecht and Dennis Allison (of *People's Computers*). It was designed to be a language which did not occupy a whole lot of memory and would be easy for children to learn and use. To this end, Dennis Allison developed and described an Interpretive Language (IL). The IL is a language within a language. It forms the framework or skeleton on which Tiny BASIC is built. One feature of writing in IL is that programs can be more easily rewritten for different microprocessors. The framework given by IL helps some but one must still sit down and write the code, this is what Tom Pittman did.

One difference between an interpreter and a compiler is that everything you type into the computer gets saved in RAM in the interpretive approach. Thus, you can save space in writing Tiny BASIC programs by abbreviating PRINT to PR for example.

Tom laughs when recalling his writing of the 6502 program. "A lot of people think that the 6502 is a lot like the 6800," he explains. "It isn't, believe me. The microcomputer world was just sitting around waiting for someone to write Tiny BASIC for the 6502 since the 6800 version was out and everyone thought the two micros were similar." Finally, Tom decided to step in and write the 6502 Tiny BASIC." He wrote the 6800 Tiny BASIC in exactly 2048 or what programmers refer to as 2K of memory, but writing Tiny BASIC for the 6502 took 200 more bytes, and due to the way the microprocessor functions it just could not be reduced to less. "And I had all kinds of reasons to want it to fit since it could be made into ROM that way, but it just wouldn't." Then RCA funded the development of Tiny BASIC for the 1802. "Without even trying, the 1802 fit Tiny BASIC into about 200 bytes less than 2K."

"The ability to change the Program Counter (PC) is one of the outstanding features of the COSMAC," says Pittman. "Did you know that the 1802 was developed entirely by one man—Joe Weisbecker?" Tom adds, "One person designing something can do a lot more than a committee; it is the only way to do something. Believe it or not, there are more features and the microprocessor is even more elegant than Joe Weisbecker intended." "This microprocessor is so good that even RCA is not really aware how good it is," Tom Pittman sighs. He continues, "The 1802 is a complete and symmetrical microprocessor."

[Part II of this two part article will be in the next QUESTDATA]

QUICK, WHAT IS 3A PLUS 4B?

This machine language program will add 3A to 4B and display the answer on your LED display. Adding two numbers together in hexadecimal shows the Elf's numbering system in action. It also has a useful programming application—modification of addressing to allow you to move a program up in memory. The shortest way to write this program is to use the two numbers to be added together as data in the program. See HEX ADDITION WITH ADDENDS AS DATA listing.

When you key in the given program and run the experiment, you will see the answer to the addition problem displayed. If you are working with the COSMAC VIP, you will find the answer as the contents of location OE.

The arrows on the listing refer to the two numbers to be added. By changing the contents of these locations you will change the problem.

The 64 instruction is what puts the answer on your display. This instruction tells the computer to display the contents of the location contained in the register pointed to by X. Since X points to OE, you see the contents of this location. Once you become used to this method of display **(display indirect)** it will seem quite elegant. The next issue of QUESTDATA will cover the indirect family of instructions in detail.

The add instruction itself is FC in machine code. This instruction tells the computer to add the next byte it encounters to its D-register. Since the next thing it comes accross is 4B, bingo—it adds the two.

If the Q-LED is on you have a three digit result with 1XX in the left most position (X's represent Hex numbers of resultant denominations).

How can you relocate memory using this procedure? You have only to take your absolute branches and add the hex number that you want to move your program up in memory to it.

If the program is a short one this method of moving things up in memory will work fine.

Try lots of numbers in this experiment. The more you work with hexadecimal the more second nature it will seem. Hexadecimal is a number system based on sixteen. If we had sixteen fingers we might be using it as our standard number system. The binary is given in the examples and Hex chart so you can add numbers like the microcomputer. The microcomputer uses the given rules to operate on the on's (1's) and off's (0's). Try working out the binary example and then look up the answer on the Hex Equivalency Card. If you have an original Elf you will see the results given to you in binary.

This addition problem shows the relationships which exist between the binary, decimal and hexadecimal number systems. Both the addressing and OP CODES were designed for hex in the 1802 microcomputer so the better you know hex the more fun you can have with your COSMAC.

HEX ADDITION WITH ADDENDS AS DATA

```
LOC.  CODE MNEM.  ACTION
 00    E4   SEX    REG.4 IS POINTER
 01    F8   LDI    LOC. 0E TO STORE
 02    0E         ADDITION ANSWER
 03    A4   PLO    R4.0 POINTS LOC.0E
 04    F8   LDI    LOAD FIRST NO. INTO
→05    3A         D REG.(EXAMPLE=3A)
 06    FC   ADI    ADD NEXT BYTE
→07    4B         TO D REG.(E.G.=4B)
 08    54   STR    STORE ANS. LOC.0E
 09    64   OUT4   DISPLAY ANS.
 0A    3B   BNF    OVERFLOW CONDITION?
 0B    0D         GO TO END IF DF=0
 0C    7B   SEQ    IF DF=1 TURN ON Q
 0D    00   IDL    "END"
 0E               ANSWER STORED HERE
```

BINARY		DECIMAL	HEX
●●●●	=	0	0
●●●○	=	1	1
●●○●	=	2	2
●●○○	=	3	3
●○●●	=	4	4
●○●○	=	5	5
●○○●	=	6	6
●○○○	=	7	7
○●●●	=	8	8
○●●○	=	9	9
○●○●	=	10	A
○●○○	=	11	B
○○●●	=	12	C
○○●○	=	13	D
○○○●	=	14	E
○○○○	=	15	F

Hexidecimal Equivalency Card

HOW TO ADD HEX

Example:

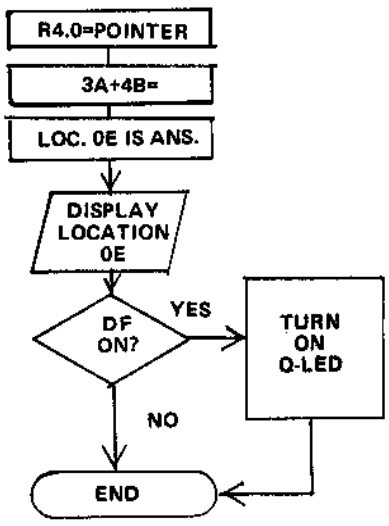
```
  ●●○○  ○●○●
+ ●○●●  ○●○○
```

ANSWER TO PROBLEM IS ON PAGE 12

Rules:

● + ● = ●
○ + ● = ○
● + ○ = ○
○ + ○ = ●
with ○ carry

○ = on
● = off



Three Minute Long Distance Telephone Timer

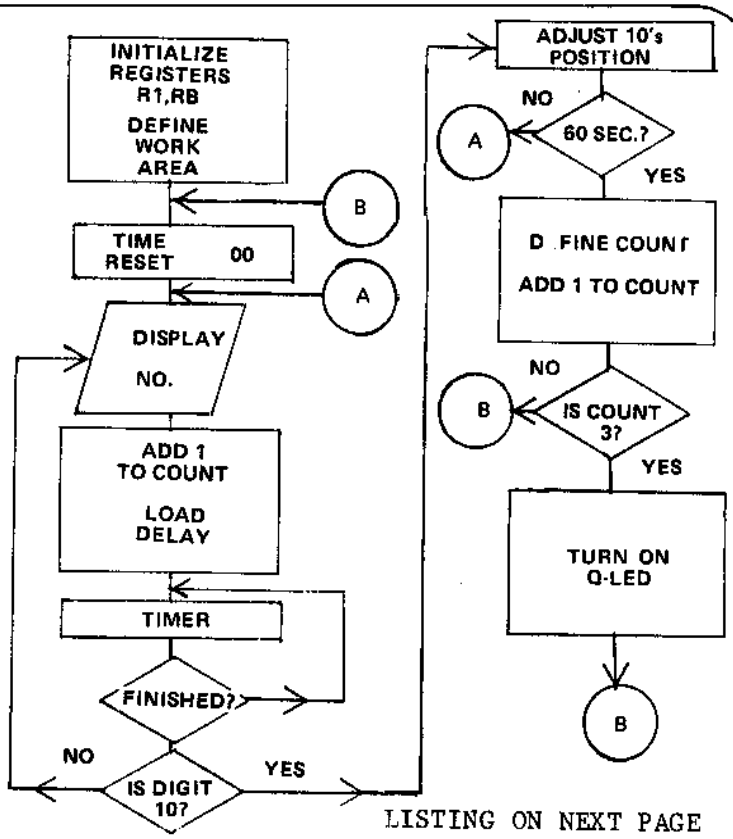
Now you can time your long distance calls and know exactly when you are going into overtime. When the three minutes are up the Q-LED lights up and you know you are going into extra innings. Why make Ma Bell richer than she already is?

This program should work in all COSMAC systems in its present form. This includes Elf's with extended memory and COSMAC VIP systems. When the Q-light comes on think up an interesting conclusion to your telephone conversation. "Just remembered that the bath water is running, see ya, bye." By cutting your talk-athon short you can save some money which you can invest in your computer.

The Quest electronic engineer who developed the Super Elf, points out that there is a good trick for saving time when loading a long program such as this with your Super Elf. What you do is use the monitor (with all its loading advantages) to load from location 21 Hex and up. Then you release the monitor from duty and load from location 00 to 21.

The way to control the duration of the seconds is with hex location 16. If you wish a delay longer than 3 minutes, then change hex location 37 to the delay you wish. If you own a COSMAC VIP you will hear a tone as well as see your Q-LED come on at the end of three minutes. The delay stored in hex location 16 is based on systems with the TV interface (clock rate 1.7 Mhz.).

Electronics magazines feature hardware timers for darkrooms, games and other activities from time to time. With a COSMAC computer you have many devices in one. If you are using your microcomputer to time a chess game, and you want 5 minutes per move, you have only to change the contents of location 37 (hex) to 05. Since your COSMAC system is very portable it can be taken to where the action is. Thus, you can take your computer to the eggs, games and sprinklers you are timing.



LISTING ON NEXT PAGE

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

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

Listing For Three Minute Timer

ANSWER TO THE PROBLEM ON PAGE 10-
(3A+4B=?) IS 85 HEX

LOC.	CODE	MNEM.	ACTION
00	F8 FC	LDI	INITIALIZE LOC.FC
02	AB	PLO RB	RB.0 IS POINTER
03	F8 00	LDI	MORE INITIALIZING
05	BB	PHI RB	OF HI REGS.
06	5B	STR R4	CLEAR LOC. FC
07	B1	PHI R1	R1.1=00
08	F8 FF	LDI	DEFINE WORK AREA
0A	A1	PLO R1	R1.0=WORK POINTER
0B	F8 00	LDI	INITIALIZE LOC. FF
0D	51	STR R1	MEM. LOC.FF=00
0E	E1	SEX R1	X=1
0F	64	OUT 4	SHOW LOC.FF(M)
10	21	DEC R1	RESTORE ADDRESS R1
11	F0	LDX	D REG.=LOC.FF(M)
12	FC 01	ADI	ADD 01 TO D REG.
14	51	STR R1	PUT RESULT LOC.FF
→ 15	F8 91		AMT. OF DELAY
17	B2	PHI R2	R2.1=DELAY
18	22	DEC R2	R2-1
19	92	GHI R2	D REG.=R2.1
1A	3A 18	BNZ	GO TO LOC.18 IF D REG.≠00
1C	F0	LDX	D REG.=LOC.FF(M)
1D	FE	SHL	SHIFT LEFT TO
1E	FE	SHL	GET RID OF HIGH
1F	FE	SHL	ORDER DIGIT
20	FE	SHL	. . . (THUS 4 SHL)
21	FB A0	XRI	TEST FOR 10
23	3A 0E	BNZ	IF≠10 THEN LOC.0E

LOC.	CODE	MNEM.	ACTION
25	F0	LDX	D REG.=LOC.FF(M)
26	FC 10	ADI	ADD 10 TO ADJUST
28	FF 0A	SMI	SUB. 0A TO ADJ. UNITS
2A	51	STR R1	STORE IN LOC.FF(M)
2B	FB 60	XRI	TEST FOR 60 SEC.
2D	32 31	BZ	IF 60 THEN LOC. 31
2F	30 0E	BR	IF NOT 60 THEN LOC.0E
31	EB	SEX RB	POINT TO RB
32	F8 01	LDI	ADD 1 TO COUNT
34	F4	ADD	D REG.+1
35	5B	STR RB	STORE RESULT
→ 36	F8 03	LDI	3 MIN. ?
38	F3	XOR	COUNT=3?
39	3A 0B	BNZ	IF COUNT≠3
3B	7B	SEQ	Q=ON
3C	30 0B	BR	SET TIME=00

((CLOCK FREQ. IN MHZ.)(20833))−256=
(CONVERT ANS. TO HEX)=
HHXX WHERE HH IS LOC.16

MUSIC AND GAMES FOR BASIC ELF

A new booklet entitled *Programs for the COSMAC Elf—Music and Games* is available. The author, Paul C. Moews, has written programs for "Morra" (a match wits with the computer guessing game), Bridg-it, reaction time tester, tic-tac-toe, music programs, monitor type subroutines and more. The 45 page booklet was written for the basic 256 byte Elf but getting the programs to work in expanded memory requires nothing more than initializing the high order addresses to 00. The explanation of each program is good and the programs are documented. The booklet can be ordered by sending $2.50 plus 50¢ for shipping to QUEST Electronics.

COSMAC CLUB COSMAC CLUB COSMAC CLUB COSMAC CLUB COSMAC CLUB COSMAC CLUB COSMAC

1

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

ADDRESS CORRECTION REQUESTED

BULK RATE
U.S. Postage Paid
QUEST
Electronics

Permit No. 649
Santa Clara, CA