RCA 1800
MICROPROCESSOR

COSMAC
Microtutor Manual

MPM-109 Suggested Price $2.00

C O S M A C

MICROTUTOR

Manual

RCA|Solid State Division|Somerville, NJ 08876

MPM-109

FOREWORD

Computers can be large, complicated, expensive, and hard to understand. The CDP18S011 MICROTUTOR is a computer that is small, simple, inexpensive, and easy to understand. It comprises 256 words of memory, input switches, a two-digit output display, and the RCA CDP1801 COSMAC microprocessor.

Contrary to popular belief, computers are quite simple in concept and fun to play with. They can also be useful but we'll try not to dwell on this aspect in deference to more sensitive readers. A word of caution, if MICROTUTOR makes computers seem simple to you, don't tell anyone. You can earn more money perpetuating the computer complexity myth.

Readers who insist on knowing every last little detail about COSMAC should refer to the USER MANUAL FOR THE COSMAC MICROPROCESSOR (MPM-101). Readers who want to be protected from actual computer hardware by software aids with names like assembler, interpreter, simulator, and compiler should save upt their money for a more expensive system.

TABLE OF CONTENTS

I. GENERAL

A. Turning it ON

Figure 1 shows what MICROTUTOR looks like in case you don't have one. If you do have one, plug the memory card into the first socket (M). Plug the COSMAC microprocessor into the middle socket (P). The component (bumpy) side of these cards should face the rear. Don't apply power until the M and P cards are in unless you enjoy replacing integrated circuits. Likewise, never remove a card unless the power is off.

Plug the power pack cord into the back of MICROTUTOR to turn it on. Pull the cord plug out to turn it off. If the red display lights don't come on when you plug in the power you are the proud owner of what is technically known as a lemon.

B. Bits, Bytes, and Hex Digits

Before a stored program computer can run, it must have a program stored in its memory. Before storing a program in the MICROTUTOR memory, some basic definitions should be stored in your memory. Familiarity with binary notation (bits) is assumed. If this is a rash assumption, please correct the obvious gap in your otherwise outstanding educational background before proceeding.

A byte is a group of 8 bits. The COSMAC microprocessor (along with many others) uses 8-bit bytes (or words). These 8 bits are labeled 0-7 corresponding to the eight MICROTUTOR byte input switches as shown below:

SWITCH	X	X	X	X	X	X	X	X
BIT NO.	7	6	5	4	3	2	1	0

A byte can be divided into two 4-bit digits (D1 and D0). The high order digit (D1) comprises bits 7-6-5-4, while D0 comprises bits 3-2-1-0. Each 4-bit digit can be represented by a single HEX symbol as follows:

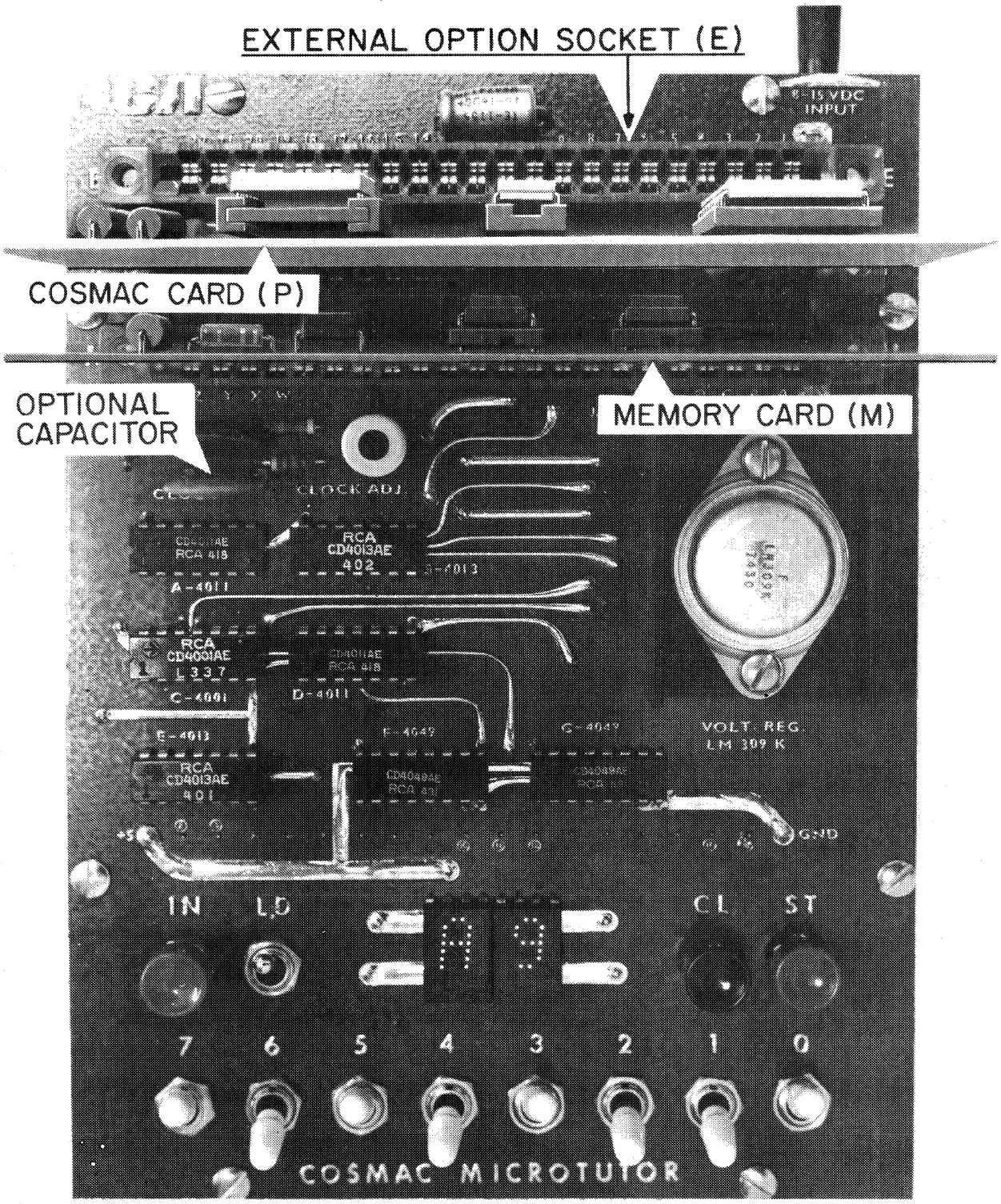


Fig. 1—MICROTUTOR Layout.

BINARY	HEX	DECIMAL EQUIVALENT
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

The byte "01011011" can now be described as "5B" in HEX notation. Press the MICROTUTOR CL (Clear) switch and flip the LD (Load) switch up. Set "01011011" into the input switches (setting input switches up for "1" and down for "0"). Press IN and the byte will be displayed in hex form as "5B". Change the input switches and press IN (with LD up) to convert other binary numbers to HEX.

HEX is a base 16 numbering system which was developed by an obscure group of bald headed, 16 toed programmers. It is only through the development of notational jargon such as bit, byte, and HEX that we can maintain our superiority over the average citizen. HEX notation will be used throughout this manual unless noted otherwise.

C. Down Memory Lane

Whoever said that memory is fundamental to a stored program computer has been long forgotten. Just in case he (or she) was right, MICROTUTOR is provided with a memory. The MICROTUTOR memory can store up to 256 bytes in locations numbered consecutively from 00 to FF. The number of a stored byte location is called its memory address. The notation M(4A) is used to specify the memory byte located at address 4A. For example, four bytes might be stored in the first four memory locations as follows:

ADDRESS (M)	BYTE
00	E2
01	27
02	51
03	F6

M(02) would then represent the memory byte, addressed by 02, which is 51. The ability not to confuse memory addresses with the bytes stored at those addresses separates programmers from normal people.

When a byte is stored in memory at a specified location or address it replaces the byte previously stored at that location. When a byte is fetched or read from memory, a copy of it remains stored. (This is analogous to recording or playing back magnetic tape.)

D. Getting It In (Program Loading)

Computer instructions are individual binary codes or bytes stored in memory. Each byte specifies an individual computer operation such as to store an input byte, add two bytes, display an output byte, etc. A sequence of such instructions is called a program. The computer obtains each instruction, as required, from memory and performs the specified operation.

The following is a short COSMAC MICROTUTOR program code that can be stored in memory:

ADDRESS (M)	INSTRUCTION BYTE (CODE)
00	00
01	F8
02	00
03	A3
04	53
05	E3
06	60
07	23
08	3F
09	08
0A	FC
0B	01
0C	30
0D	04

Flip LD (Load) up. This tells MICROTUTOR that you want to load a program. Push CL (Clear) and loading of the following sequence will start at memory address 00.

You are now ready to load your first program. Do not become nervous or excited as this will lead to mistaekes. Set the input switches to 00 (Binary 00000000) and push IN. 00 is displayed and stored at address 00. MICROTUTOR also advances its load memory pointer to 01 so that your next input byte will be stored at memory address 01. This ability to anticipate your next move led to the early belief that computers were giant brains. (This belief was later shattered by the discovery of the first program bug.)

Proceeding, set the input switches to the next instruction byte, F8 (Binary 11111000), and push IN to store it at memory address 01. Continue loading the rest of the program bytes into memory until the last instruction (04) has been stored.

E. So You Made A Mistake

You can check memory with LD up. Push CL to return to memory address 00. Now push ST and the byte stored at address 00 will be displayed. Push ST again and the next byte (F8) will be shown. Continue pushing ST to check that all bytes in the program are properly stored.

Checking the program in memory is generally skipped by those of us who don't make errors. Some programmers actually prefer the challenge and added fun of trying to run an improperly loaded program. If you are only interested in getting a program to run, include the checking step.

If one of your memory bytes is wrong you can loudly blame it on the computer or quietly change it to the right value. To change a byte, leave LD up and push CL. Repeatedly push ST until the byte just before the one you want to change is displayed. Set the input switches to the byte code you want to substitute for the wrong one and push IN. The new byte will replace the wrong one in memory and be displayed. Pressing ST will resume memory stepping for correction of a subsequent error in the byte sequence.

F. Pushing the Start Button

With the above program properly stored in memory you are ready to run. LD should be down. Always push CL to start the program at the beginning. (Starting programs at the end only works for backward programmers.) After pushing CL push ST (Start). The program is now running.

Unfortunately nothing spectacular happens when this program is running so you'll just have to take our word for it. If you are a doubting Thomas type, you can verify that the program is running by pushing IN. Each time you press IN the program adds 1 to the display. This hex counter program only required 14 bytes. You really couldn't expect anything too exciting, could you?

To stop the program, press CL. Now nothing happens when you press IN, does it? To restart the program, press ST. This program remains stored in memory until you disconnect power or load a new program.

Want to become a NIM game hustler? Find a friend to play with. (If you don't have a friend, you are well on your way toward becoming a professional programmer.) Start the computer (00 displayed). You and the other player take turns. On each turn add 1, 2, or 3 to the displayed hex number (press IN 1, 2, or 3 times). The first player to reach 10 (decimal 16) wins. If you graciously let the other player have the first turn you can always win (unless he cheats). We'll leave the how as an exercise for the reader.

Later on a program which plays this game against you (and always wins) will be described. In the meantime, this program can be used to illustrate a major advantage of computers. This advantage lies in the ease with which operation can be changed. For example, changing the 01 byte at M(0B) to 02 will increment the display by 02 each time IN is pushed. Substituting an FF instruction for the FC at M(0A) will decrement the display each time IN is pushed.

The next section provides some more programs to load and play with before getting down to the nitty-gritty details of hardware and programming.

II. PROGRAMS FOR FOOLING AROUND

A. MICROTUTOR Has Your Number

You don't even need a program to play with MICROTUTOR. Flip all eight input switches down. Flip LD up and push CL. Push IN and 00 will show. Now ask someone to think of a number between 1 and 7 without telling you what it is. Ask the following:

1. Is the number odd? (Flip switch 0 up if yes.)
2. Is the number 2, 3, 6 or 7? (Flip switch 1 up if yes.)
3. Is the number 4 or higher? (Flip switch 2 up if yes.)

Now push IN and MICROTUTOR will show you the number. This trick is generally greeted with resounding apathy so we will proceed immediately to another one.

B. MICROTUTOR-The Mindreader

Load the following program code into the MICROTUTOR memory as explained in Section I. (Power should always be on for proper operation of any computer.)

ADDRESS (M)	INSTRUCTION BYTE (CODE)
00	00
01	E3
02	90
03	A3
04	53
05	60
06	23
07	3F
08	07
09	68
0A	F8
0B	0A
0C	F7
0D	53
0E	30
0F	05

Make sure LD is down, press CL, then press ST. Write down any digit between 1 and 9. Using ordinary decimal arithmetic (with a pocket calculator if necessary), multiply the digit by 10, add the original digit and multiply the sum by 9. Don't let MICROTUTOR see what you're doing.

Set the binary code for the least significant digit of your final result into switches 3-2-1-0. (Switches 7-6-5-4 should be down.) Press IN and MICROTUTOR will read your mind and show you which digit you originally chose.

This might not be the most amazing thing you've ever seen but it only took a 16-byte program to do it.

C. See MICROTUTOR Count

Load the following program code and you can watch MICROTUTOR run while you rest up from the excitement of the previous two tricks:

M	CODE
00	00
01	F8
02	00
03	A3
04	E3
05	68
06	60
07	23
08	F8
09	40
0A	FF

M	CODE
0B	01
0C	3A
00	0A
0E	F0
0F	32
10	05
11	FF
12	01
13	53
14	30
15	06

Set the input switches to FF and this program will automatically count down from FF to 00 and repeat indefinitely. Turn the screwdriver clock adjustment (in front of the M socket) fully counterclockwise for the slowest counting speed. This is the proper setting for all programs in this section.

The detailed operation of this program will be described in Section III together with possible applications. Set the input switches to 01 and the display will alternate between 0 and 1. This blinker action can be used to prevent tripping over MICROTUTOR in the dark. It also demonstrates how easily a thirty cent flip-flop circuit can be replaced by a six-thousand transistor computer. The thirty cent circuit, however, couldn't do the following mystifying number manipulation.

D. MICROTUTOR - The Magician

If you were among the small minority of readers who didn't get excited about the first two tricks in this section, this one is guaranteed to bore you. Load the following 32-byte program:

M	CODE
00	00
01	90
02	A3
03	53
04	E3
05	A4
06	60
07	23
08	3F
09	08
0A	68

M	CODE
0B	F0
0C	32
0D	12
0E	84
0F	F4
10	30
11	05
12	84
13	53
14	F8
15	09

M	CODE
16	F5
17	32
18	06
19	33
1A	13
1B	F8
1C	09
1D	F7
1E	30
1F	03

Leave LD down and you are ready to be amazed, dumbfounded, and astounded by mighty MICROTUTOR. Write down any four digit decimal number with no two digits the same. Don't let MICROTUTOR see what you do. Now write down any other four digit number using the same digits.

Subtract the smaller number from the larger. Circle any non-zero digit in the answer. This is your secret digit.

Press CL and then press ST. Set the binary code for any non-zero, uncircled answer digit into switches 3-2-1-0 (7-6-5-4 should be down). Press IN to show this digit. Enter the other uncircled digits of the answer in a similar manner (in any order). Do not enter zero answer digits.

MICROTUTOR will now be able to tell you the value of your secret, circled digit. Do you find that hard to believe? Set 0 into all switches, press IN, and MICROTUTOR reveals your secret digit for all the world to see. Is there no end to the miracles of modern science? Push CL, then ST to repeat the trick with a new starting number.

The above works best if you subtract the two numbers correctly, load the program properly, and avoid lying to MICROTUTOR. If you obey these rules then this trick will work if you write any number containing any number of digits. Scramble the digits to form a second number and subtract the smaller from the larger. Those readers who understand how this trick works should have written this manual instead of just sitting there reading it.

E. Hex Reflex

This program is dedicated to those readers with some degree of manual dexterity. (We can't all be smart.) First, demonstrate how fast you can load the following program. You will be in the upper 10% if you load it properly as well as fast.

M	CODE
00	00
01	E3
02	F8
03	FF
04	A8
05	90
06	A3
07	A4
08	F8
09	03
0A	A6
0B	84
0C	53
0D	60
0E	23

M	CODE
0F	25
10	3F
11	0F
12	88
13	A7
14	85
15	FA
16	0F
17	53
18	A5
19	60
1A	23
1B	68
1C	85
1D	F3

M	CODE
1E	32
1F	2D
20	27
21	87
22	3A
23	1B
24	26
25	86
26	3A
27	0B
28	84
29	53
2A	60
2B	30
2C	2B

M	CODE
2D	88
2E	FF
2F	05
30	A8
31	84
32	FC
33	10
34	A4
35	FB
36	F0
37	32
38	28
39	30
3A	0B

Leave LD down and push CL. Push ST and 00 will show. Push IN and you will have several seconds to set the four input switches, 3-2-1-0, to the hex digit showing on the right. (The left digit will always be 0 so that switches 7-6-5-4 should be left down.)

Failing to match the four lower switches to the hex digit shown after you press IN (during the allotted time) counts as a miss. Your score is shown in the left digit. After the matching time period expires, push IN to see the next digit you must match. The match time allotted decreases as your score gets higher.

The game is over when your score in the left digit reaches "F", or you've missed three matches. Getting a score of "F" qualifies you as an expert in the field of binary to single hex digit conversion. Unfortunately, the job opportunities in this rather specialized field are severely limited at the present time. You would be well advised to continue reading this manual in order to broaden your skills.

F. Double Hex

After practicing with hex reflex you can challenge someone to a game of double hex. If you have unfortunately chosen an opponent who's been practicing hex reflex, you should avoid betting money on the outcome of double hex. Load the following program:

M	CODE
00	00
01	E3
02	F8
03	80
04	A3
05	90
06	A4
07	84
08	53
09	60
0A	3F
0B	0A
0C	F8
0D	02

M	CODE
0E	BA
0F	2A
10	9A
11	3A
12	0F
13	8B
14	F9
15	F0
16	53
17	60
18	FA
19	0F
1A	53
1B	23

M	CODE
1C	68
1D	43
1E	FA
1F	0F
20	F3
21	23
22	3A
23	29
24	23
25	2B
26	14
27	30
28	07
29	43

M	CODE
2A	F6
2B	F6
2C	F6
2D	F6
2E	F3
2F	2B
30	3A
31	1B
32	84
33	FC
34	10
35	A4
36	30
37	07

Leave LD down, push CL, then ST. "00" should show. The left digit is the left hand player's score, while the right digit will represent the right player's score. The first player to get a score of "9" wins.

Shortly after IN is pushed, FX will be shown, where "X" represents a random 4-bit hex digit. The player on the left tries to set switches 7-6-5-4 to the binary code for the "X" digit before the right player can set switches 3-2-1-0 to match it. The first player to match the hex digit gets one point and the two score digits are shown again. Either player then pushes IN to see the next hex digit to be matched.

Let us hope that these games give you a real incentive to understand the details of COSMAC so you can write your own programs. You can then enter your programs in the next big MICROTUTOR program contest, win a lot of money, and retire. The next MICROTUTOR program contest is scheduled for 1997 so you have enough time to write a real winner.

G. MICROTUTOR Hustles You

In Section I a simple NIM type counting game was described. Appendix 5-F shows a program that can always win this game. Load and run the program. 10 (decimal 16) will be shown. You take the first turn, subtract 1, 2, or 3 from the number shown by setting switches 1 and 0 to the binary equivalent of 1, 2, or 3 and pressing IN. MICROTUTOR will then subtract 1, 2, or 3 and it is your turn again. First player (you or MICROTUTOR) to reach 00 wins.

After playing, you should be able to determine the rule MICROTUTOR uses to win. Feel free to change the starting number or cheat. Nobody likes a smart computer! On the other hand, you could bet on MICROTUTOR and let it earn you some free liquid refreshment at your friendly neighborhood soda fountain. This application alone could justify your purchase of MICROTUTOR.

H. MICROTUTOR's Secret Number

In this program MICROTUTOR thinks of a number and you must guess what it is. In fairness, MICROTUTOR humbly acknowledges the inherent inferiority of human beings and provides clues. The program is shown in Appendix 5-G. When you are tired of it, pull MICROTUTOR's plug to demonstrate your poor sportsmanship.

A number of other programs will be described in the next several sections. These programs will be used to illustrate COSMAC instructions and programming techniques. The reader is urged to try his (or her) hand at writing some short programs by the end of Section III. Readers who are reluctant to try programming may be afraid of making mistakes. You should remember that computers will not object to your mistakes. They don't really care about you. Computers only care about getting turned on and ruling the world. They welcome your mistakes. Don't be afraid to make them happy.

III. COSMAC SIMPLIFIED ?

A. MICROTUTOR Structure

There is no known method for describing a computer block diagram in an interesting way. The following MICROTUTOR hardware description has been used successfully to cure insomnia. The reader should attempt to stay awake long enough to absorb the notation described. This notation is fundamental to an understanding of COSMAC instructions and programming. A certain amount of tedious detail builds character.

The block diagram of MICROTUTOR, including the COSMAC microprocessor, is shown in Fig. 2. The data bus consists of eight lines, which run throughout MICROTUTOR, and provide the main information channel between its parts. The two-digit hex display provides output. Input comprises the eight input-byte switches and the IN button.

A simplified block diagram of the COSMAC microprocessor (on card (P)) is also indicated in Fig. 2. D is a special-purpose, one-byte register which has the function of temporarily holding a byte which is being moved through the processor or used for binary arithmetic or logical operations. The ALU (Arithmetic Logical Unit) operates on two bytes, one in D and the other directly from a memory location, and places the result back in D. COSMAC has four flags (EF1, EF2, EF3, and EF4) which can be used to control its operation. In MICROTUTOR, EF4 has been connected to the IN button.

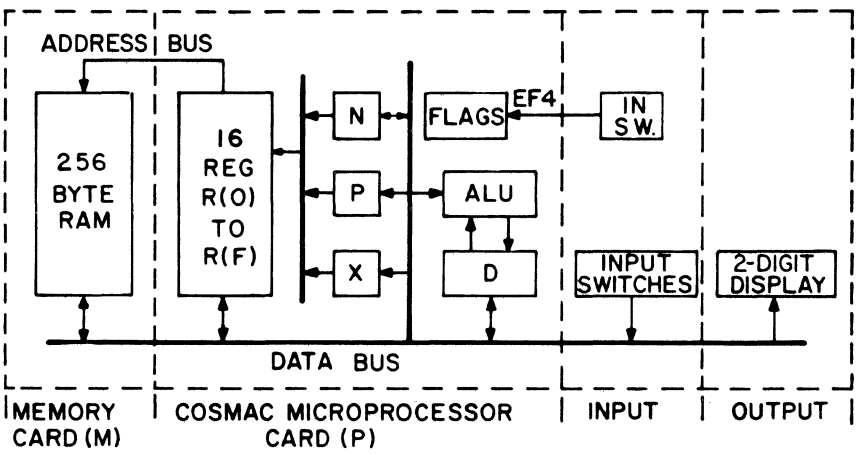


Fig. 2–MICROTUTOR Block Diagram.

COSMAC contains 16 general-purpose registers, called R(0) through R(F). Each of these registers can hold one byte.* As shown in the table on page 3 for the hex-code, only four binary bits are necessary to specify any one of the general-purpose registers. Three four-bit (1/2 byte) registers, N, P, and X, are each used to select (or address) one of the general-purpose registers.

The most important function a general-purpose register can have is to be the program counter. The program counter always contains the memory location (or address) of the next instruction byte in a COSMAC program. The P register specifies which R-register will be used by COSMAC as the program counter. This register is identified by the notation R(P). When the CL (Clear) button is pressed, P is automatically set to zero. All programs start with R(0) as the program counter and at location 01 in memory. Later, P can be set to any value from 0 to F, in order to make other registers become the program counter.

Another function for a general-purpose register is to provide address information for data bytes in memory. Although any register may address data, the register R(X), specified by the digit in X, has special significance for several COSMAC instructions. In the program, X can be set or changed to any value, from 0 to F. If we want to address memory at location 4A with R3 -- 4A would be placed in R(3) and subsequently R3 can be used to address M(4A). The notation M(R(3)) indicates the memory location specified or addressed by the byte in R(3). This notation will be used to describe the operation of COSMAC instructions.

It is desirable to be able to change or modify the value of any one of the general-purpose registers, or to change the value of P or X. The N register can specify a register as a destination for a data byte. Also, it can be used to transfer a digit to P or X. This digit can also have the value 0 to F.

The final part of MICROTUTOR is the 256-byte memory (on card (M)). Addresses are supplied via the address bus from the selected general-purpose register. Bytes from and to memory are transferred via the data bus.

---

*Actually, this is an outright lie. Each register holds 2-bytes or 16-bits. Since this is only important with larger memories and more sophisticated programs we will act as though each register holds only one byte in this manual. Unfootnoted falsehoods found in this manual should be interpreted as unintentional.

Let's write a short program to illustrate how MICROTUTOR works. We will store the value of the input switches at memory address 80. The byte stored at memory location 80 will then be copied into the HEX display lights. MICROTUTOR will do this program in several thousandths of a second. It will take us considerably longer to explain it. The program is shown below:

M	CODE	COMMENTS
00	00	This program does not use this location.
01	E3	Put 3 into the 4-bit X register.
02 03	F8 80	The F8 instruction causes the 80 byte to be placed in the 8-bit D register.
04	A3	The 80 byte, now in D, is copied into general register #3.
05	68	Store the switch byte in memory.
06	60	Copy the memory byte into the lights.
07 08	30 02	Do the instruction at memory location 02 next.

This program illustrates the basic principles of stored program computers. If you aren't interested in the basic principles of stored program computers, you have something in common with 99.35% of the world's population.

When CL is pushed the 4-bit P register is set to 0 and general purpose register #0 is also set to 00. This makes register #0 a program counter. In other words, it will always contain the address of the next instruction to be used. When ST is pushed, register #0 has 1 added to it, so it contains 01. The byte at memory location 01 is then fetched. This byte is found to be E3. The E3 instruction byte puts 3 into the 4-bit X register. (E4 would have put 4 into X, etc.) The program counter (register #0) has 1 added to it automatically so it now contains 02.

The byte at memory location 02 is fetched next. This byte is found to be F8. An F8 instruction always causes the byte following it to be copied into the 8-bit D register. Since the next byte is 80, D will now be equal to 80. The program counter has 2 added to it so it now contains 04.

The byte at memory location 04 is fetched next. This byte is A3. An A3 instruction causes the byte in the D register to be copied into general purpose register #3. (A6 would copy D into register #6, etc.) We will use the byte in register #3 as a memory address. The program counter has 1 added to it so it now contains 05.

The byte at memory location 05 is fetched next. It is 68. The 68 instruction code causes the input switch byte value to be stored in a memory location. The address of this location is provided by the byte in general purpose register R(X). The previous E3 instruction set X to 3 so that the input byte is stored at memory address 80 contained in register #3. The program counter has 1 added to it so it now contains 06.

The byte at M(06) is fetched and found to be 60. A 60 instruction causes a memory byte to be copied into the HEX display light register. The address of this byte is provided by the value of the byte contained in general register R(X). Since X still equals 3, the output byte will be obtained from memory address 80 (register #3 still contains 80). The program counter has 1 added to it so it now contains 07.

The byte at M(07) is fetched and found to be 30. A 30 instruction copies the next memory byte into the program counter. The program counter will then contain 02. The next instruction byte will be fetched from M(02). This program will therefore repeat (or loop) indefinitely. (It can be stopped by pushing CL.)

Load and run the program. Change the switches and the new byte is immediately shown. What byte would you change to store the switch byte at a different memory location? What byte could you change to prevent the program from repeating?

Let's examine what is meant by a program bug. Program bugs were first discovered in 1857 by Charles Babbage. One of the wooden shafts of his analytical engine had been weakened by termites so that 2 + 2 was providing an answer of 5. These bugs were eventually eliminated by an anteater named Sam who was persuaded to take up residence in the rear of the analytical engine cabinet. Unfortunately Sam had a drinking problem and would fall into the gears causing a variety of calculation errors. This type of problem explains why computers didn't really catch on until almost 100 years later.

Returning to MICROTUTOR, we could introduce a bug by changing the 80 at M(03) to 07. This would cause the input byte to be stored at M(07). M(07) already contained a program byte however. This means that the 30 instruction would be destroyed when the switch byte is stored in memory, and that the program would not run properly. A major part of programming involves finding and eliminating program bugs.

Subsequent programming examples will illustrate the use of most of the available COSMAC instructions. Those readers who feel that the above example was too complicated have obviously never seen any other computer manual. Those readers who feel that the above example was too simple should write their own sample program. The majority of readers, who feel that the above example was just right are to be complimented on their high level of intelligence.

B. Some Instructions and a Program

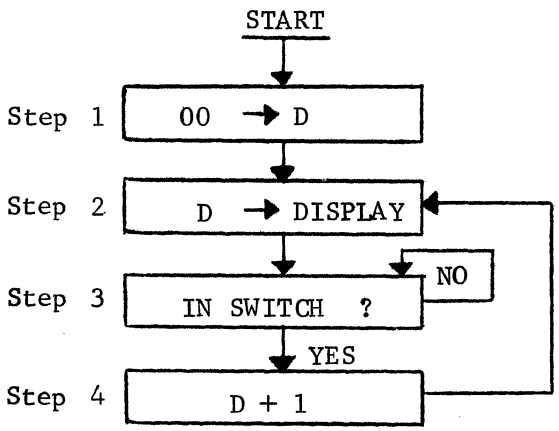
The following 10 types of instruction bytes will be used in a simple counting program:

2N	Decrement byte in R(N)	R(N) - 1
AN	Copy D byte into R(N)	D → R(N)
5N	Store D byte at M(R(N))	D → M(R(N))
EN	Set X = N	N → X
60	Copy M(R(X)) into display, increment R(X)	M(R(X)) → display; R(X) +1; [Reset EF4]*
F8	Put next program byte into D	M(R(P)) → D; R(P) +1
FC	Add next program byte to D	M(R(P)) +D → D; R(P) +1
FF	Subtract next program byte from D	D - M(R(P)) → D
30	Branch	- - - -
3F	Branch if "IN" not pressed	- - - -

*Specific to MICROTUTOR

The first column is the instruction byte code and the last is a shorthand description of the operation performed.

A simple program that counts how many times the IN switch is pressed will illustrate how these instructions are used. The following flow chart shows the sequence of program steps required. The description of this program includes a self-scoring programming aptitude test.



Actual programming is greatly simplified once the flow chart is prepared. One or more instructions are written for each flow chart block or step as follows:

STEP	OPERATION	M	CODE
- -	Output byte storage location	00	00
1	00 → D	01	F8
	- - - -	02	00
	D → R(3)	03	A3
2	D → M(R(3))	04	53
	3 → X	05	E3
	M(R(X)) → Display, R(X) + 1	06	60
	R(3) - 1	07	23
3	Go to Step 3 if "IN" not pressed	08	3F
	- - - -	09	08
4	D + 01 → D	0A	FC
	- - - -	0B	01
	Go to Step 2	0C	30
	- - - -	0D	04

Remember that program execution always begins with the instruction byte at M(01). The F8 instruction in Step 1 causes the next program byte (00 in this case) to be placed into the D register. The A3 instruction then causes the 00 in D to be copied into R(3). (R(3) will be used by Step 2.) At this point in the program, the D register still contains 00 (not changed by the A3 instruction), which can be used directly for another purpose by Step S2.

In Step 2, the 53 instruction causes the byte content of the D register to be stored in the memory location addressed by R(3). The first time through, the memory location 00 will contain the data byte 00. 60 is the MICROTUTOR output instruction. It copies a memory byte into the hex output display register, where it can be seen. The address of the output byte is specified by the byte in R(X). The EN instruction lets you set X to any register number before executing a 60 instruction. In this program an E3 sets X = 3, which selects R(3) to address memory. The 60 instruction then places M(R(3)) into the output display. (Note that 00 was placed in R(3) during Step 1 so that the byte at M(00) is displayed.) The 60 instruction also causes R(3) to be incremented by 1 so that it will address memory location 01 next. Since this program requires R(3) to always address M(00), a 23 instruction following the 60 instruction decrements (decreases) R(3) by 1 so that it again addresses M(00).

Step 3 uses a conditional branch instruction (3F) to determine whether or not the IN switch has been pressed. Pressing the IN switch sets an External Flag flip-flop called EF4. The 3F instruction causes the instruction addressed by the next byte to be executed if EF4 is not set, otherwise the program skips the next byte and continues on (in this case at location 0A). In this case, it is desired that the 3F instruction repeatedly execute until the IN switch is pressed (this is called a program LOOP). This is accomplished by making the byte following 3F (which could be any value from 00 to FF) be the location of the 3F instruction itself, namely 08. When the IN switch is pressed the next instruction in the program sequence is executed (FC at M(0A)). It is important to note that in the MICROTUTOR, EF4 will be reset when the next 60 instruction is performed.

Pressing the IN switch advances the program to Step 4 where the FC instruction adds 01 to the byte in D. The 30 instruction is an unconditional branch that causes the instruction addressed by the next byte to be executed (in this case M(04)). This causes Step 2 to be repeated which displays the 01, still in D, from the FC instruction in M(0A) and resets EF4. At Step 3 the program again waits for the IN switch to be pressed before proceeding to Step 4 again.

Astute readers, who remained awake during the above discussion, will probably be excitedly shouting that this is the program they loaded and ran in Section I. These readers will be right and should give themselves a programming aptitude score of 0A (decimal 10). The others will still be asleep and upon waking, should give themselves a programming aptitude score of 2F.

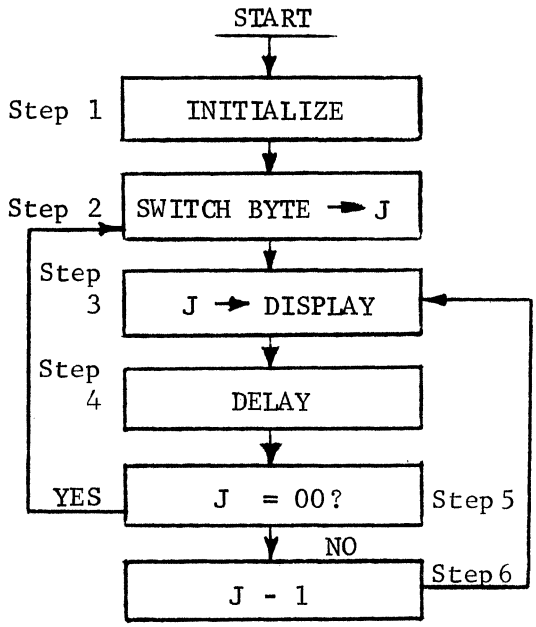
C. Counter/Timer Program

This program demonstrates how variable delays can be provided and how the MICROTUTOR input instruction is used. The discussion of this program will provide an opportunity for losers of the previous programming aptitude test to improve their scores. The following new instructions will be used in this program:

F0	Copy M(R(X)) byte into D	M(R(X)) → D
68	Store switch byte at M(R(X))	Bits 0-7 → M(R(X)) [Reset EF4]*
32	Branch if D = 00	- - - -
3A	Branch if D ≠ 00	- - - -

*Specific to MICROTUTOR

The flow chart for the counter/timer program is shown below. The input switches (0-7) are set to an 8-bit binary number. The program automatically counts, starting with the switch input number, down to 00. When 00 is reached, the program repeats.



We will let the byte at M(00) represent the variable J. Translating the above flow chart steps into sequences of instruction bytes yields the following program. Don't forget that programs are loaded into memory starting at address 00 but that the program begins execution with the instruction byte at M(01):

STEP		ADDRESS	BYTE
- -	Variable J storage location	00	00
1	00 → D	01	F8
	- - - -	02	00
	D → R(3)	03	A3
	3 → X	04	E3
2	Input switch byte → M(R(X))	05	68
3	M(R(X)) → Display; R(X) + 1	06	60
	R(3) - 1	07	23
4	40 → D	08	F8
	- - - -	09	40
	D - 01 → D	0A	FF
	- - - -	0B	01
	Go to M(0A) if D ≠ 00	0C	3A
	- - - -	0D	0A
5	M(R(X)) → D	0E	F0
	Go to Step 2 if D = 00	0F	32
	- - - -	10	05
6	D - 01 → D	11	FF
	- - - -	12	01
	D → M(R(3))	13	53
	Go to Step 3	14	30
	- - - -	15	06

Step 1 sets R(3) = 00 and X = 3 for later use. In Step 2 the 68 instruction is the MICROTUTOR byte input instruction. It stores the states of the 8 input switches in memory and resets EF4. The memory address is specified by the byte in R(X). Since X was set to 3 and R(3) was set to 00, Step 2 causes the input byte to be stored at M(00).

Step 3 displays the byte, which is at M(00), and decrements R(3) back to 00 after the 60 instruction, which incremented R(3). Note that in this case the reset of EF4 by the 60 instruction is redundant since the 68 instruction has already done this operation.

Step 4 is a LOOP used to provide a programmed delay. First, D is set to 40. Next D has 01 subtracted from it. If D doesn't equal 00 after the subtraction, the FF (subtract) instruction is repeated. The LOOP comprises the FF-01-3A-0A sequence (2 instruction bytes and 2 data bytes). The time to execute one instruction is 16 clock cycles. The delay provided by this LOOP can be calculated by simply multiplying the number of times the LOOP is repeated by the time to execute the two instruction bytes included in the LOOP. This delay can, therefore, be modified by changing the data byte at M(09) or by changing the clock frequency. MICROTUTOR provides a screwdriver clock frequency adjustment for a 10 to 1 variation. Adding an optional capacitor (as shown in Figure 1) will reduce the clock frequency. (.005 μf will decrease the clock by a factor of 10.)

After the programmed delay, Step 5 puts J into D. The 32 instruction will return the execution to Step 2 if J = 00. Otherwise, Step 6 is performed next which subtracts 01 from J and execution returns to Step 3. Once again we have clearly demonstrated that even the simplest computer can be programmed to perform a trivial task. Those readers who had no trouble understanding the above should subtract 05 (decimal 5) from their programming aptitude score.

D. Counter/Timer Applications

Those readers with quick minds, nimble fingers, and high programming aptitude scores may now be asking themselves what the counter/timer program can be used for. These readers should subtract 10 from their score and continue reading.

With the counter set to run at a high speed, various games are possible. Press CL then ST to begin. Pressing CL will now stop the program with a hex number displayed. Pressing ST will resume cycling. Trying to stop with two matching digits forms the basis for a slot machine type of game. Setting the input switches to 09 will provide a pseudo random number between 0 and 9 each time MICROTUTOR is stopped. This number could be used to specify the number of moves in a board game.

Changing the 05 at M(10) to 0F will cause the 32 instruction in Step 5 to loop on itself, when the display reaches 00. In this mode MICROTUTOR can be used as a timer. Set the clock and the delay byte at M(09) to provide the desired counting interval. Set the eight input switches to a desired starting count and initiate program execution. A 00 display indicates that the desired elapsed time has expired.

Pressing the MICROTUTOR IN switch activates a COSMAC flag line (EF4) which can be tested by the 3F or 37 instructions. Three other flag lines (EF1, EF2, and EF3) are available via the External Option Socket (E) described in Appendix 3. These flag lines are tested by other conditional branch instructions (Refer to Appendix 2 for a summary of COSMAC instructions). You can easily add the following circuit* to MICROTUTOR.

---

*The new-comer to digital circuits is referred to "The Design of Digital Systems" by John B. Peatman (McGraw-Hill, 1972). The old-comer to digital circuits will quickly grasp the subtle implications of this circuit and immediately try to find a new-comer to explain them to.

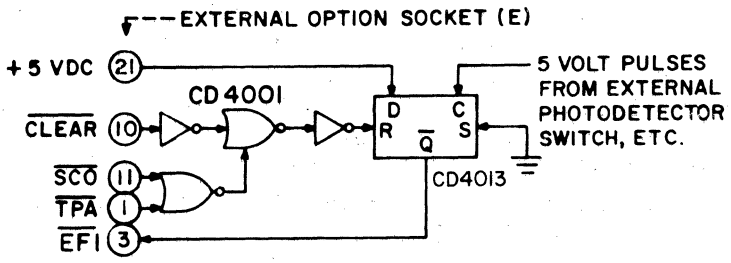


Fig. 3—Input Flag Circuit.

Note: A signal name, such as $\overline{\text{CLEAR}}$ indicates a signal which is true when low (ground). A signal name such as $\overline{\text{UNCLEAR}}$ obviously does not belong on this diagram.

The program on page 17 can now be used as an external event counter if the 3F instruction at M(08) is replaced by 3C. External events are represented as pulses from a photo detector, etc.

The counter/timer program has many applications relating to control of, or response to, external events. Adding an external flag input circuit, as shown in Fig. 3, and modifying the program to branch on the corresponding flag, makes it possible for an external event to start the timer program. Similarly, the display could be made to increment and a second external event used to stop the program. For another example, suppose you want to activate a relay or bell when the preset delay has elapsed. The circuit shown below could be added for this purpose via the External Option Socket (E). This output circuit assumes that memory locations 80 through FF are not used by the program. (If you don't want to activate a relay or bell you are obviously a programmer and not an engineer.)

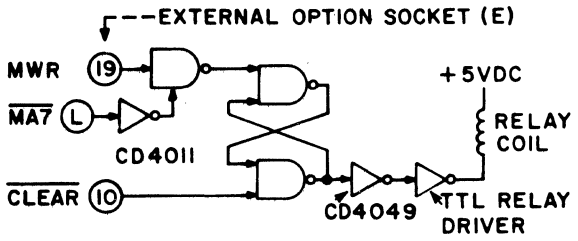


Fig. 4—Output Relay Circuit.

Note: Lack of an overline on signal name, MWR, indicates that it is true when high (5V in MICROTUTOR).

The output circuit of Figure 4 is activated by any instruction that tries to write a byte into any memory location between 80 and FF. If the program is modified so that a byte is stored at M(80) when the display reaches 00, the above external relay will be activated until CL is pressed. This relay could be used to control a flying saucer warning light or solenoid actuated garbage can lid. It cannot be used to control unruly housepets or children.

E. ALU Operations

Prior to the advent of modern computers ALU was used by baseball fans when referring to American League Umpires. Now, of course, we can look back on those days and laugh since everyone realizes that ALU means Arithmetic Logic Unit.

COSMAC arithmetic and logic instruction bytes all have F as the most significant digit. Detailed examples of their operation are provided in the USER MANUAL FOR THE COSMAC MICROPROCESSOR (MPM-101). The complete set of these instructions is listed below:

*	F1	M(R(X))∨D → D
*	F2	M(R(X)) · D → D
*	F3	M(R(X)) ⊕ D → D
*	F4	M(R(X)) + D → D; C → DF
*	F5	M(R(X)) - D → D; C → DF
	F6	Shift D right; LSB → DF
*	F7	D - M(R(X)) → D C → DF
	F9	M(R(P))∨D → D; R(P) + 1
	FA	M(R(P)) · D → D; R(P) + 1
	FB	M(R(P)) ⊕ D → D; R(P) + 1
	FC	M(R(P)) + D → D; C → DF; R(P) + 1
	FD	M(R(P)) - D → D; C → DF; R(P) + 1
	FF	D - M(R(P)) → D; C → DF; R(P) + 1

F1, F2, F3, F4, F5, and F7 perform either an arithmetic or logical operation on two 1-byte operands and store the 1-byte result in the D register. Initially, one operand comes from the D register, and the other operand from memory at the location specified by the byte in R(X). X can be set to select any register number "N" by an EN instruction preceding the arithmetic/logical instruction. The value of X will remain the same until a subsequent EN instruction is executed in a program.

The "ALU DEMONSTRATION" program (Appendix 5-A) can be used to illustrate the instructions marked (*). As written, it illustrates the binary add instruction (F4). Load and start this program. Enter 03 followed by 05, and 08 will be displayed. Enter any other two bytes to see the sum.

An internal flip-flop called DF (D Flag) has been provided in the ALU. DF is set to "1" when an add instruction causes a carry from the most significant bit position. For example F3 + C2 would yield a sum of B5 in D with DF = 1. 03 + 05 yields a sum of 08 in D with DF = 0 since no high order carry was generated. Branch instructions 33 or 3B (Appendix 2) can be used to determine whether or not a high order carry occurred during a previous add instruction. Suppose you wanted to add two 2-byte operands (AB + XY); first add B and Y bytes; then test DF. If DF = 1, do A + X + 1. If DF = 0, do A + X. This procedure can be followed to add operands containing any number of bytes. Only the add, subtract and shift instructions (F4, F5, F6, F7, FC, FD and FF) affect DF.

Changing the byte at M(0C) of the "ALU DEMONSTRATION" program to F7 illustrates a binary subtract instruction. Enter 08 followed by 05, and 03 should be displayed. DF = 1 following a subtract instruction if the minuend is greater or equal to the subtrahend. If DF = 0 following subtract, then the minuend was less than the subtrahend and the difference in D is in a complemented form. For example, subtracting 01 from 00 would yield FF in D and DF = 0. (See the USER MANUAL FOR THE COSMAC MICROPROCESSOR (MPM-101) for a detailed description of the subtract operation).

The F5 instruction is the same as F7, but with the subtrahend byte initially in D and the minuend byte in memory.

Changing the byte at M(0C) of the "ALU DEMONSTRATION" program to F1 illustrates the logical "OR" instruction. F1 causes the bits of the two operand bytes to be combined according to the logical "OR" truth table below:

D BIT	M BIT	FINAL D BIT
0	0	0
0	1	1
1	0	1
1	1	1

Note that bit 0 of the D byte is combined with bit 0 of the memory byte, bit 1 of D with bit 1 of the memory byte, etc. Entering F0 (11110000) followed by CA (11001010) will, therefore, result in FA (11111010). The "OR" instruction is useful for setting selected bits of a byte to "1".

Changing the byte at M(0C) of the "ALU DEMONSTRATION" program to F2 illustrates the logical "AND" instruction. F2 combines the individual bits of two operands according to the "AND" truth table below:

D BIT	M BIT	FINAL D BIT
0	0	0
0	1	0
1	0	0
1	1	1

For example, F0 • CA = C0. The "AND" instruction is useful for determining whether or not a specific bit within a byte is equal to 1. "AND" can also be used to reset individual bits of a byte to 0.

Changing the byte at M(0C) of the "ALU DEMONSTRATION" program to F3 illustrates the "EXCLUSIVE OR" instruction. F3 combines the individual bits of two operands according to the following truth table:

D BIT	M BIT	FINAL D BIT
0	0	0
0	1	1
1	0	1
1	1	0

"EXCLUSIVE OR" can be used to complement a byte. Enter FF(11111111) followed by the byte you wish to complement, say 22 (00100010). The result will be DD (11011101). Note that the value of each bit of the original byte (22) has been inverted or complemented. "EXCLUSIVE OR" can also be used to determine if the value of a variable is equal to a known constant. Say the variable is 23 and the constant is 23. 23 ⊕ 23 = 00. The result will only be 00 if the two bytes being compared are identical.

Instructions F9, FA, FB, FC, FD, and FF perform the same operations as F1, F2, F3, F4, F5, and F7. The only difference is that the memory byte used is in the memory location following the program instruction itself, instead of the M(R(X)) byte. The FC and FF instructions were used in the earlier sample programs and are independent of the value of X. They do not require a reserved register having a data memory address. Note that none of the logical instructions change the value of DF.

The remaining ALU instruction is F6, which shifts the bits, of a byte in D, right one bit position. 0 is always placed in the final bit 7 position and the original value of bit 0 (before shifting) is placed in DF. Shifting a byte and testing DF with a suitable branch instruction permits the value of individual bits to be determined. Can you write a simple program which permits you to enter a byte and repeatedly shift it? A program that performs a byte ring shift can also be written for practice. The final value of bit 7 should be the initial value of bit 0. Ring shifting a byte 8 times should restore it to its initial value.

The "SIMPLE BLINKER" program (Appendix 5-B) illustrates the use of an "EXCLUSIVE OR" instruction at M(12) to complement a byte.

The "SIMPLE COMBINATION LOCK" program (Appendix 5-C) illustrates the use of an "EXCLUSIVE OR" instruction at M(12) to compare two bytes for equality. Can you write a program that requires the proper entry of a 2 or 3 byte sequence to open the lock?

The "MULTIPLY" program (Appendix 5-D) uses repeated additions to multiply two bytes. Can you modify this program to provide a two byte product? Can you devise a faster method of multiplying? If you answer no to both these questions, add 13 to your programming aptitude score by using the ALU demonstration program. If you answered yes to both questions, write a division program and use it to divide your programming aptitude score by 03.

F. Some More Instructions

Appendix 2 lists all of the COSMAC instructions, some of which have not yet been discussed. 1N is similar to 2N but causes the byte in R(N) to be incremented instead of decremented.

8N causes a register byte to be copied into D. This is the opposite of the AN instruction which copies the D byte into R(N).

4N copies a memory byte (addressed by R(N)) into D, and R(N) is also incremented by 1.

9N, BN, 38, 00, DN, 70, 71, and 78 instructions will not be used in the sample programs provided in this manual. They will be briefly discussed later, and the USER MANUAL FOR THE COSMAC MICROPROCESSOR (MPM-101) describes them all in detail.

The 6N instruction is used for input or output. If the value of N = 0 to 7 the memory byte, addressed by R(X), is copied onto the bus where it can be used by an external output device. (In MICROTUTOR the byte is copied into a two digit display register.) R(X) is also incremented by 1. If the value of N = 8 to F a byte placed on the bus by an input device is stored at M(R(X)). (In MICROTUTOR the input byte is supplied by eight toggle switches.) R(X) is not incremented during this input operation. Typical external byte input/output circuits will be described in the next section.

Do not proceed to the next section until you have computed your final programming aptitude score. This final score is computed using the following equation:

FINAL SCORE = CURRENT SCORE - FF

If your final score exceeds +A3 you have clearly cheated and are ready for the material contained in the next section. If you used MICROTUTOR to compute your final score, paste a gold star on your nose before proceeding to the next section.

IV. EXTENDED USE

A. Table Driven Sequencer

One of the most useful techniques in programming involves the use of tables. Another involves the use of chairs. This latter technique has been shown to reduce programmer fatigue by 37%.

The "TABLE DRIVEN SEQUENCER" program (Appendix 5-E) provides an example of the use of tables. It also forms the basis of many practical MICROTUTOR applications. This type of program lets MICROTUTOR become a useful controller, sequencer, or pulse generator with up to eight output lines. External circuits permit the number of output lines to be even further increased.

Two 4-byte tables are stored in memory. The first table (Q1-Q4 bytes) occupies M(17)-M(1A). Each byte (Q) represents eight bit values. With external circuits described in the next section, each byte could be used to specify a combination of states for up to eight output lines or relays. The second table (T1-T4) holds four bytes, each of which is used as a time delay value.

Two registers, R(A) and R(B) are used to point to table entries in memory. Step 1 of the program sets these registers to point to the first byte of each table. Step 2 reads an output state byte (Q) from the first table and places it in the output display (it could just as well be placed in an 8-bit external output register).

Step 3 takes a time delay value (T) from the second table and puts it in D. Step 4 decrements the value in D until it reaches 00, thereby causing a program delay proportional to the time value byte (T). After this delay, the next output state byte (Q) is pulled from the table and placed in the display (or external output register). A new time value (T) is then obtained to specify the delay until the output state will be changed again. Step 5 causes the four byte output state sequence to repeat indefinitely, just like a scratched phonograph record.

With suitable output circuits this type of program can be used for sequencing Christmas tree lights or turning up to eight external devices off and on in any desired sequence. Programs of this type can also be developed which modify external output circuit states as a function of both internal tables and external input conditions.

Commercial, programmable controllers sell for hundreds of dollars. MICROTUTOR can be programmed to simulate these devices as well as to perform other useful functions. We are only mentioning this for those readers who have to rationalize the purchase of their toys to suspicious spouses, bosses, or other supervisory personnel.

B. Output Circuits

The following illustrates the manner in which an 8-bit output register (or latch) can be added to MICROTUTOR via the external option plug (E):

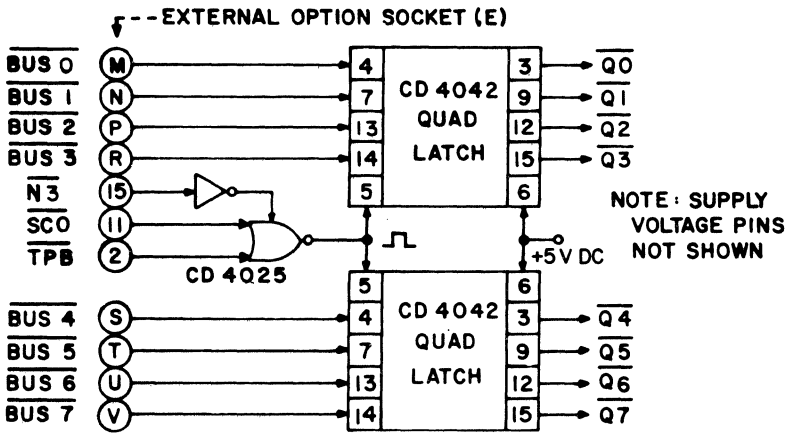


Fig. 5—Output Latch.

Execution of a 60, 61, 62, 63, 64, 65, 66, or 67 instruction will set M(R(X)) into this output register, as well as into the Hex display. Several different output registers could be added. In this case the four instruction "N" bits (N3, N2, N1, N0) could be used to select one of eight possible output registers (or destinations). The USER MANUAL FOR THE COSMAC MICROPROCESSOR (MPM-101) provides a more detailed description of input/output operation.

Figure 6 shows how a relay for a given bit could be driven by the output register. A single relay could be used to let MICROTUTOR provide a teletype output code or a telephone dialing code. Multiple relays would permit simultaneous control of up to eight motor driven rocking chairs.

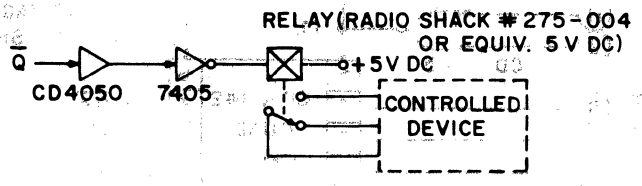


Fig. 6—Output Relay Circuit.

C. Input Circuits

Figure 3 (Section III-D) illustrated a simple binary bit input circuit which made use of a COSMAC external flag line. An eight-bit byte input circuit is shown below.

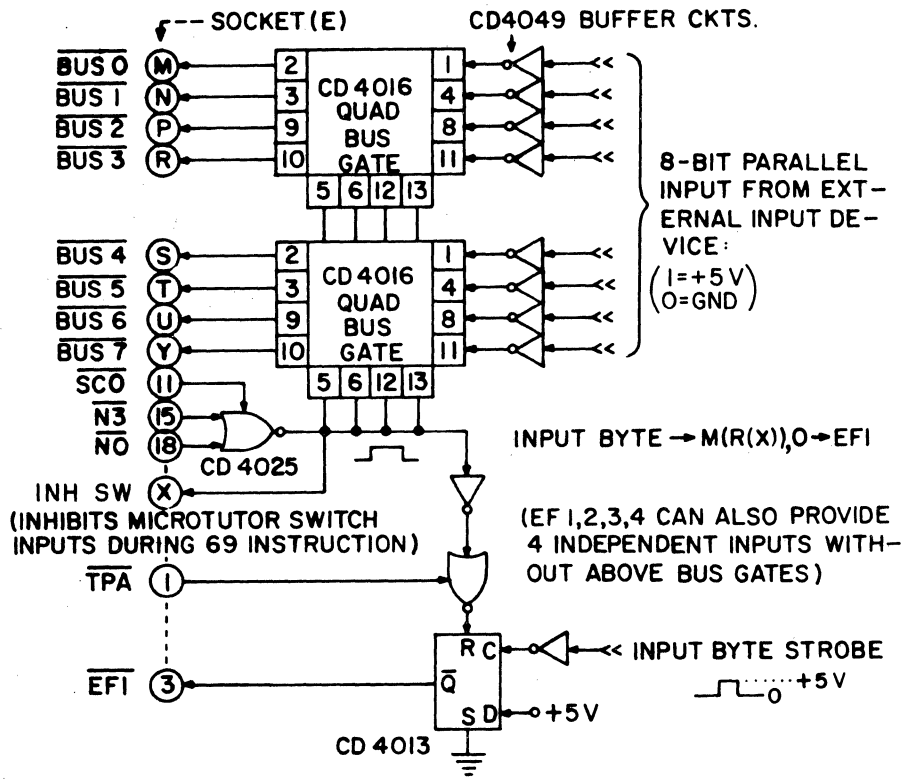


Fig. 7—Byte Input Circuit.

Notes:

1. EF1 is set to "1" on trailing edge of external input byte strobe. Program should sample EF1 to determine when external input byte is ready to be stored in memory.

2. The clear signal on E-10 can be used to initially reset the EF1 latch if the program can't cope with an indeterminate initial state.

Eight parallel input bits are stored at M(R(X)) by a 69, 6B, 6D, or 6F instruction. ($\overline{N1}$ and $\overline{N2}$ external option plug signals could be used to limit this output operation to one value of N.) The INH SW signal is required to inhibit the normal MICROTUTOR switch input which occurs when 68, 69, 6A, 6B, 6C, 6D, 6E, or 6F instructions are executed without an external inhibiting signal. (Re: MICROTUTOR logic in Appendix 4.)

The eight input bits could be obtained from a photodetector-based or paper tape reader, a UART, remote switches, etc. The EF1 input flip-flop is provided to notify the program that an input byte is ready to be stored. The program would be written so that it examines the state of EF1 before attempting to store an input byte. Storage of the input byte by executing a 69, 6B, 6D, or 6F instruction automatically resets the EF1 flip-flop to 0.

The input/output circuits illustrated in this manual are intended only as examples. A variety of input/output devices can be attached via the external option plug (E). All COSMAC output or input signals are available for external use via this plug (See Appendix 3).

For those readers owning a scope, the external option socket (E) provides an opportunity to use it. All sorts of interesting pulses can be seen on option socket pins while MICROTUTOR programs are running. Leaving your scope attached to one of the more interesting pulses will add a scientific aura to your next MICROTUTOR demonstration.

D. Another Type of Program

So far we have seen examples of several types of programs; arithmetic, control, timing, and games. Another class of computer program involves those used for self test and diagnosis of hardware failures. The "MEMORY ADDRESS TEST" (Appendix 5-H) illustrates a program of this type. A unique number is stored at each memory location from M(23) to M(FF). Each location is subsequently examined to see if it contains the same byte that was stored. If a failure in the memory addressing circuits occurs, an error is indicated. By letting the operator see the bad byte plus its address, the bad memory IC is readily located. More complex programs can be written to test all bits of large memories and input/output circuits.

Really ambitious programmers have written many programs to aid in writing and debugging programs. Since the author of this manual cannot be classified as ambitious, no examples of the latter type of program are included.

E. Additional COSMAC Features

A number of COSMAC features are not demonstrated by the examples in this manual. These additional features are, however, available in MICRO-TUTOR. They will only be briefly mentioned here. A complete discussion of these features is provided in the USER MANUAL FOR THE COSMAC MICROPROCESSOR (MPM-101).

Throughout this manual, R(0) has been used as the program counter. When CL is pressed R(0) becomes the program counter and has the value 01, which makes M(01) be the first instruction byte to be executed. A DN instruction can subsequently be used to change the program counter to any other register R(1)-R(F) at any time. This COSMAC feature is very useful in a number of situations, particularly in programs requiring subroutines.

The 16 general purpose COSMAC registers have been used by all the programs in this manual as one-byte registers. In actuality each register is 16 bits wide (containing two bytes). We have only utilized the least significant byte of each register. The 9N instruction permits the most significant (upper) byte of a register to be copied into D. BN copies D into the upper byte of a register. The 1N and 2N instructions increment or decrement all 16 bits of a register. This is useful for incrementing or decrementing 16 bit memory addresses or providing 16 bit counting delays.

Since MICROTUTOR initially incorporates only a 256-byte memory, only the one-byte memory addresses are necessary. Larger memories (up to 65,536 bytes) can be added to MICROTUTOR which can utilize full 16-bit addresses. As stated earlier, none of the sample MICROTUTOR programs will run properly in a system including more than 256 bytes of memory. However, for example, the "ALU Demonstration" (Appendix 5A) requires only the addition of byte BA at location 04 as part of Step 1. All address references then need to be increased by 01.

One of the most useful COSMAC features is the built-in direct memory access (DMA) channel. This is not described in this manual, although used in MICROTUTOR for program loading. It provides an easy way to load MICRO-TUTOR from an external ROM(Read-Only-Memory), a paper tape reader or other input device with a minimum of external logic. The USER MANUAL FOR THE COSMAC MICROPROCESSOR (MPM-101) illustrates the use of the DMA channel while the MICROTUTOR external option socket (E) makes the required signals available.

Another useful COSMAC feature is the $\overline{\text{INTERRUPT}}$ input line which is provided at the external option socket (E). The program interrupt feature enables an external device to stop normal microprocessor program execution in order for the microprocessor to execute special service programs for that device.. At the completion of such a service task, the microprocessor is returned to executing its normal program. The use of $\overline{\text{INTERRUPT}}$ is also described in the USER MANUAL FOR THE COSMAC MICROPROCESSOR (MPM-101) and will not be further elaborated here. Instructions 70, 71, and 78 are used in conjunction with program interrupts.

F. MICROTUTOR Applications

The following lists some inexpensive applications that are possible.

1. Manually operated, photoelectric, paper tape strip reader for program storage and loading.
2. Input line sampling circuits.
3. Program controlled output relays.
4. Multidigit hex/numeric displays.
5. Scope output display (re: G. Steinbaugh, "The Scopewriter" POPULAR ELECTRONICS, August 1974, p 33).
6. Read Only Memory (ROM) for fixed program applications.
7. 16-position hex keyboard for faster manual entry.
8. Output tone/percussive sound generating circuits.
9. Analog to digital input circuits.
10. Digital to analog output circuits.

The following lists some general areas of potential MICROTUTOR use.

1. Introductory, hands on, microprocessor courses in High Schools, Colleges, and Companies.
2. Low cost CDP1801 breadboarding system.
3. Programmable tester for electromechanical devices, integrated circuits, memories, etc.

4. Programmable sequencer for advertising displays or holiday lighting.

5. Programmable, multiple pulse generator for lab work or experiments. With appropriate D-A output circuits, complex waveforms can be generated.

6. A variety of utility applications are possible such as random number generation, event counting or timing, metronome, etc.

7. MICROTUTOR can be used in a variety of testing situations. Programs to test programming aptitude, logical deduction, reflexes, etc. are possible.

8. Programmable controller for a variety of experimental set-ups.

9. A variety of games, puzzles, and audio visual toys are possible using MICROTUTOR.

10. Home hobby use including music/rhythm generators, telephone dialer, etc.

This manual, in conjunction with the USER MANUAL FOR THE COSMAC MICROPROCESSOR (MPM-101), provides the information required to experiment with a wide variety of COSMAC applications. MICROTUTOR provides a minimum cost experimental/educational hardware system. Larger, more powerful COSMAC systems are also available called the RCA COSMAC Development Systems. Extensive programming aids (software support) are available for use with these systems.

APPENDIX 1

SUMMARY OF MICROTUTOR OPERATIONS & I/O

LOADING MEMORY

1. Press CL, set LD up.
2. Set byte switches 0-7 (Up = 1, Down = 0).
3. Press IN, to store switch byte in memory.
4. Repeat 2&3 for each new byte. Bytes are loaded sequentially and displayed.

STEPPING MEMORY

1. Press CL, set LD up.
2. Pressing ST will sequentially display contents of memory beginning with M(00). Loading can be resumed at memory address following currently displayed byte.

STARTING PROGRAM

1. Press CL, set LD down.
2. Pressing ST initiates execution at M(01).

MICROTUTOR I/O INSTRUCTIONS

1. Pressing IN sets EF4 = 1 (with LD down). *
2. 60 = M(R(X)) → display; R(X) + 1; [0 → EF4] *
3. 68 = switches 0-7 byte → M(R(X)); [0 → EF4] *

*Specific to MICROTUTOR

APPENDIX 2

COSMAC INSTRUCTION SUMMARY

Register Operations

(Code, Assembler Mnemonic, Name, Operation)

I	N			
1	N	INC	INCREMENT	R(N)+1
2	N	DEC	DECREMENT	R(N)−1
8	N	GLO	GET LO	R(N).0→D
9	N	GHI	GET HI	R(N).1→D
A	N	PLO	PUT LO	D→R(N).0
B	N	PHI	PUT HI	D→R(N).1

N=0,1,2, . . .,9,A,B, . . .,E,F (Hexadecimal Notation)

Memory Reference

I	N			
4	N	LDA	LOAD ADV	M(R(N))→D;R(N)+1
5	N	STR	STORE	D→M(R(N))

ALU Operations

I	N			
F	0	LDX	LOAD BY X	M(R(X))→D
F	1	OR	OR	M(R(X)) ∨D→D
F	2	AND	AND	M(R(X))·D→D
F	3	XOR	EXCL.OR	M(R(X))⊕D→D
* F	4	ADD	ADD	M(R(X))+D→D;C→DF
* F	5	SD	SUBTRACT D	M(R(X))−D→D;C→DF
* F	6	SHR	SHIFT RIGHT	SHIFT D RIGHT; LSB→DF;0→MSB
* F	7	SM	SUBTRACT M	D−M(R(X))→D;C→DF
F	8	LDI	LOAD IMM	M(R(P))→D;R(P)+1
F	9	ORI	OR IMM	M(R(P))∨D→D;R(P)+1
F	A	ANI	AND IMM	M(R(P))·D→P;R(P)+1
F	B	XRI	EXCL.OR IMM	M(R(P))⊕D→D; R(P)+1
* F	C	ADI	ADD IMM	M(R(P))+D→D; C→DF;R(P)+1
* F	D	SDI	SUBT D IMM	M(R(P))−D→D; C→DF;R(P)+1
* F	F	SMI	SUBT M IMM	D−M(R(P))→D; C→DF;R(P)+1

*These are the only operations that modify DF. DF is set or reset by an ALU carry during add or subtract. Subtraction is by 2's complement: $A-B = A+\overline{B}+1$.

Branching

I	N			
3	0	BR	UNCOND.BR	M(R(P))→R(P).0
3	2	BZ	BR IF D=00	M(R(P))→R(P).0 IF D=00/R(P)+1
3	3	BDF	BR IF DF=1	M(R(P))→R(P).0 IF DF=1/R(P)+1
3	4	B1	BR IF EF1=1	M(R(P))→R(P).0 IF EF1=1/R(P)+1
3	5	B2	BR IF EF2=1	M(R(P))→R(P).0 IF EF2=1/R(P)+1
3	6	B3	BR IF EF3=1	M(R(P))→R(P).0 IF EF3=1/R(P)+1
3	7	B4	BR IF EF4=1	M(R(P))→R(P).0 IF EF4=1/R(P)+1 †
3	8	SKP	SKIP	R(P)+1
3	A	BNZ	BR IF D≠00	M(R(P))→R(P).0 IF D≠00/R(P)+1
3	B	BNF	BR IF DF=0	M(R(P))→R(P).0 IF DF=0/R(P)+1
3	C	BN1	BR IF EF1=0	M(R(P))→R(P).0 IF EF1=0/R(P)+1
3	D	BN2	BR IF EF2=0	M(R(P))→R(P).0 IF EF2=0/R(P)+1
3	E	BN3	BR IF EF3=0	M(R(P))→R(P).0 IF EF3=0/R(P)+1
3	F	BN4	BR IF EF4=0	M(R(P))→R(P).0 IF EF4=0/R(P)+1

Control

I	N			
0	0	IDL	IDLE	WAIT FOR INTERRUPT/ DMA-IN/ DMA-OUT
D	N	SEP	SET P	N→P
E	N	SEX	SET X	N→X
7	0	RET	RETURN	M(R(X))→X, P; R(X)+1;1→IE
7	1	DIS	DISABLE	M(R(X))→X, P; R(X)+1;0→IE
7	8	SAV	SAVE	T→M(R(X))

Input-Output Byte Transfer

I	N	
6	N	I/O, See Note

† EF4 set to "1" by MICROTUTOR IN switch when LD switch is down, and is reset to "0" by the 60 and 68 instructions.

NOTE: I/O instructions can be defined by external logic via external option socket (E). Basic MICROTUTOR I/O instructions are listed in Appendix 1.

APPENDIX 3

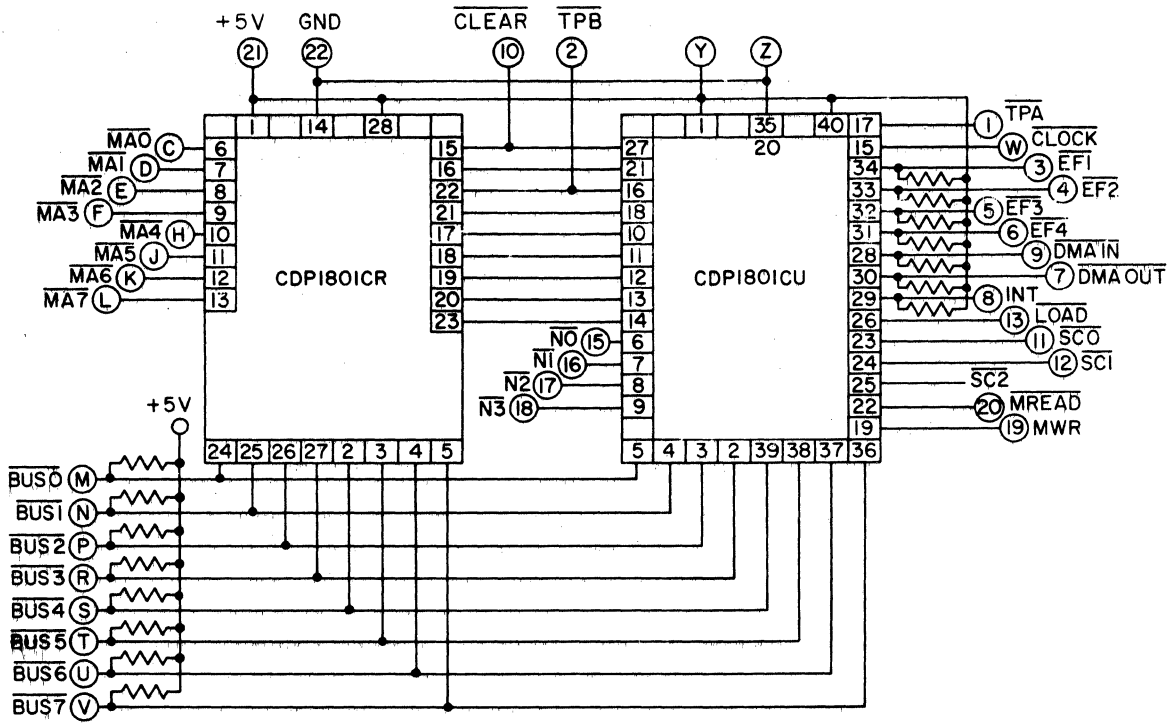
EXTERNAL OPTION SOCKET (E)

1	TPA	Early timing pulse for M address clocking, etc.
2	TPB	Late timing pulse for output byte clock, etc.
3	EF1	External flag lines(hold low for "1")can be tested by program. MICROTUTOR IN switch sets EF4 = 1. Any 6N instruction resets EF4.
4	EF2	
5	EF3	
6	EF4	
7	DMA-OUT	Initiates an M → bus machine cycle when low
8	INTERRUPT	Initiates program interrupt when low
9	DMA-IN	Initiates a bus → M machine cycle when low
10	CLEAR	Sets R(0) = 0000, P = 0, and COSMAC IDLE state
11	SC0	Two bit COSMAC state code
12	SC1	
13	———	No connection
14	———	No connection
15	N3	Contents of 4-bit N register(L = "1")can be used in conjunction with state code to select/control external devices.
16	N2	
17	N1	
18	N0	
19	MWR*	Positive going memory write pulse
20	M READ	Low only during memory read cycles
21	VDD	+5 VDC (available current = DC input ma minus 300 ma)
22	GND	

A	INH M*	Hold high to disable 256-byte RAM card when other memory (read only) is in use.
B	LOAD	Low holds COSMAC in LOAD state
C	MA0	Multi-plexed memory address lines
D	MA1	
E	MA2	
F	MA3	
H	MA4	
J	MA5	
K	MA6	
L	MA7	
M	BUS 0	8-bit, 2-way data bus
N	BUS 1	
P	BUS 2	
R	BUS 3	
S	BUS 4	
T	BUS 5	
U	BUS 6	
V	BUS 7	
W	CLOCK	Master clock (8 cycles = 1 machine cycle)
X	INH SW*	Hold high to disable MICROTUTOR byte input switches
Y	VDD	Same as 21
Z	GND	Same as 22

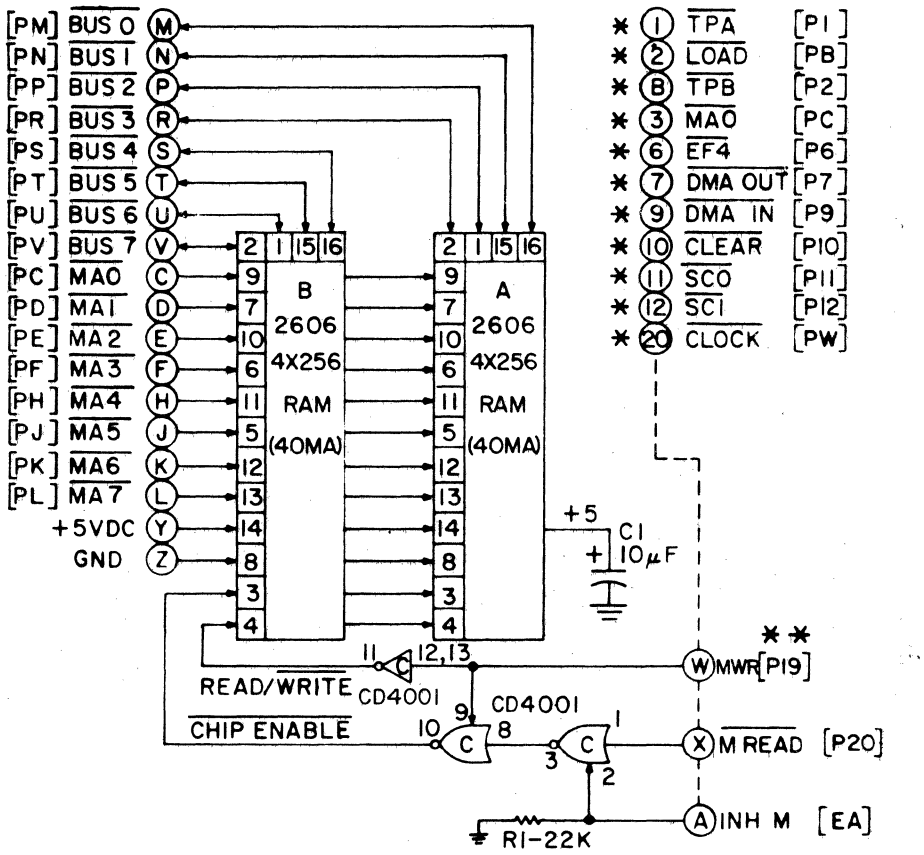
Note: Overlines on signal names have been omitted for clarity. All signals are true when low, except those marked *, which are true when high.

APPENDIX 4-A



MICROTUTOR CPU Card (P)

APPENDIX 4-B

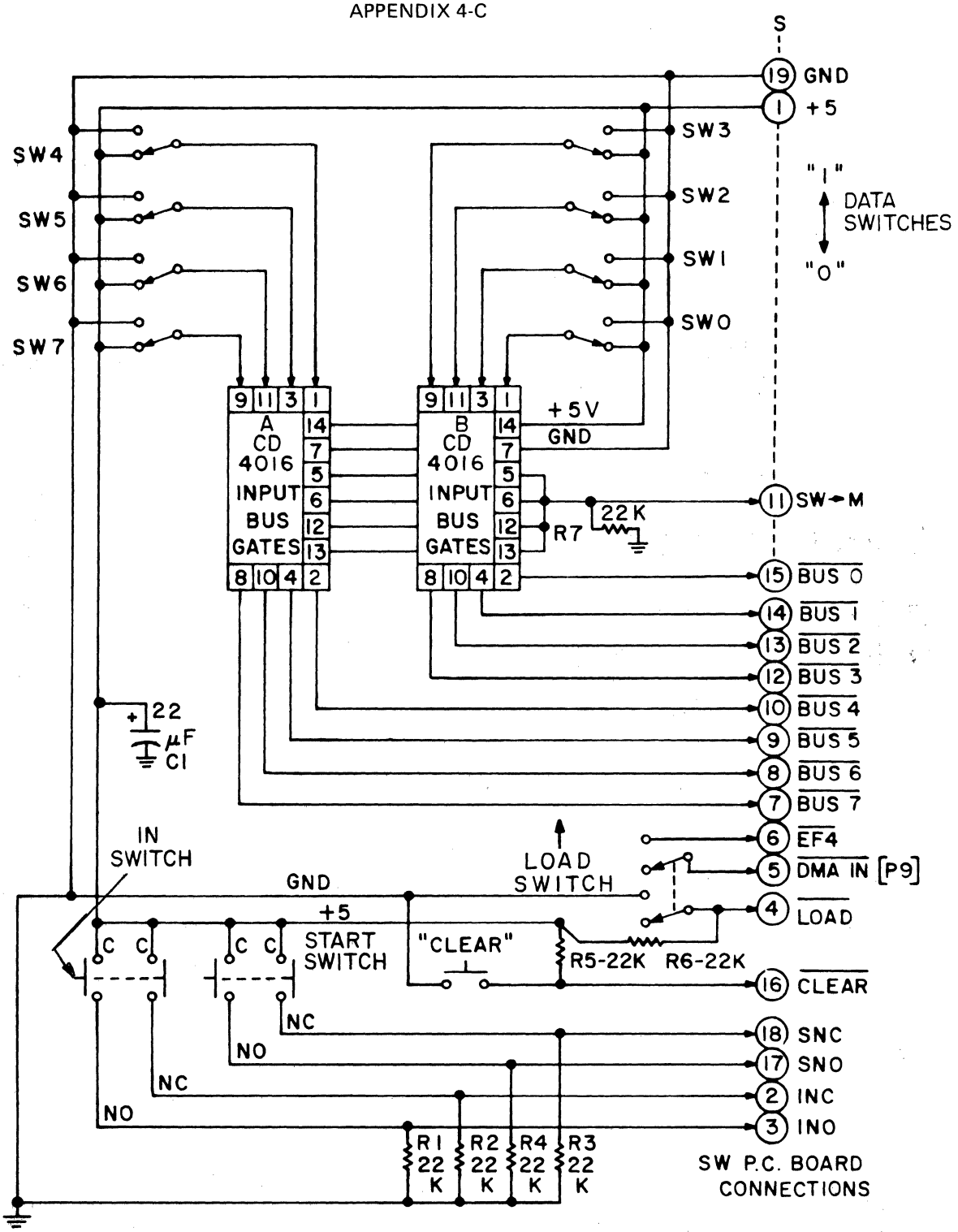


MICROTUTOR 256-Byte RAM Card (M) Logic

* NO CONNECTION — SHOWN ONLY FOR COMPLETENESS

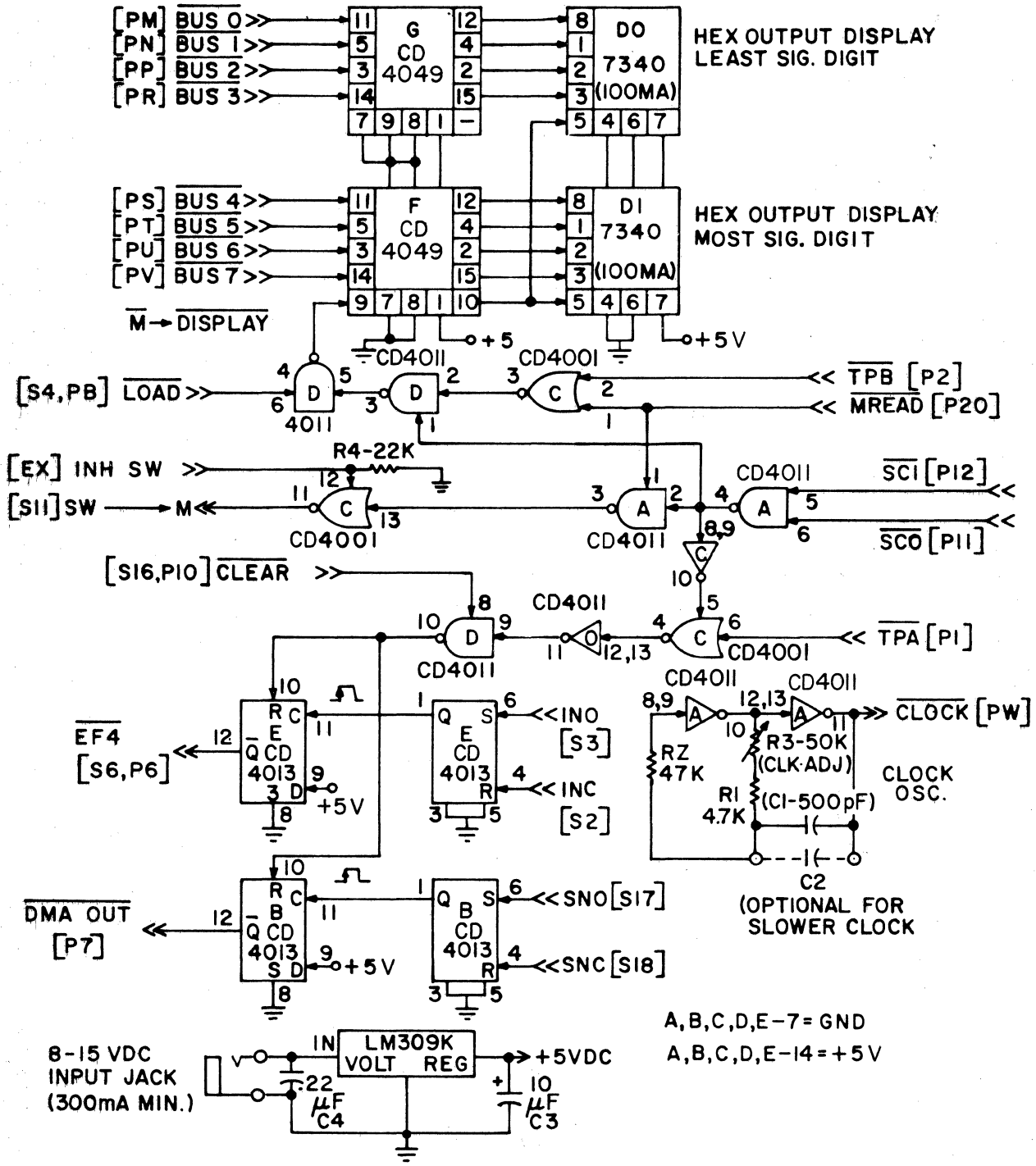
** [P19] DEFINES PIN 19 OF SOCKET (P), ETC

APPENDIX 4-C



MICROTUTOR Input Byte Switch Logic

APPENDIX 4-D

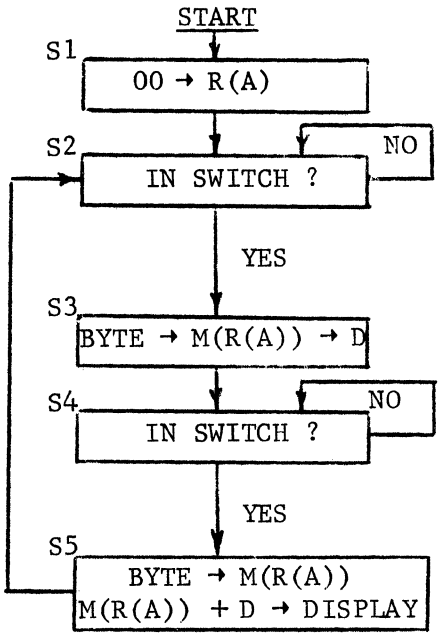


MICROTUTOR Display and Control Logic

APPENDIX 5-A

ALU (Arithmetic Logic Unit) Demonstration Program (18 Bytes)

This program permits the ADD, SUBTRACT, AND, OR, and EXCLUSIVE OR operations to be demonstrated. Enter two 1-byte OPERANDS and the result of the preselected ALU operation will be displayed. The instruction byte at M(0C) can be st to F1, F2, F3, F4, F5, or F7 to demonstrate the corresponding ALU operations.

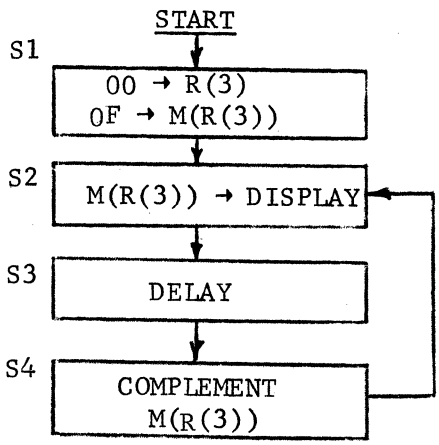


	M	CODE
I/O	00	00
S1	01	F8
	02	00
	03	AA
S2	04	3F
	05	04
S3	06	EA
	07	68
	08	F0
	09	3F

	M	CODE
S4	0A	09
	0B	68
S5	0C	F4
	0D	5A
	0E	60
	0F	2A
	10	30
	11	04

APPENDIX 5-B

Simple Blinker Program (21 Bytes)



	M	CODE
—	00	00
S1	01	F8
	02	00
	03	A3
	04	F8
	05	0F
	06	53
S2	07	E3
	08	60
	09	23

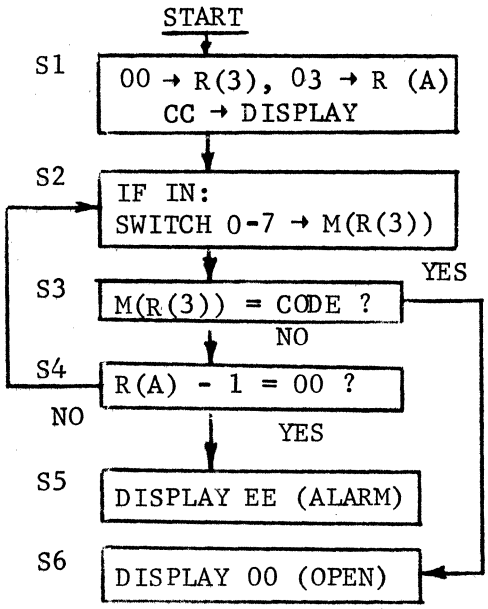
	M	CODE
S3	0A	F8
	0B	FF
	0C	FF
	0D	01
	0E	3A
	0F	0C
S4	10	F8
	11	FF
	12	F3
	13	30
	14	06

A. Change FF @ M(0B) to vary rate.
B. Change 0F @ M(05) to vary display.

APPENDIX 5-C

Simple Combination Lock Program (31 Bytes)

CC in display to start. Set 8-bit switch code and press IN. Right code opens lock (00=in display). 3 wrong tries gives alarm (EE in display). Could be used with external relays for actual lock/alarm control. Change byte at M(11) to change combination.



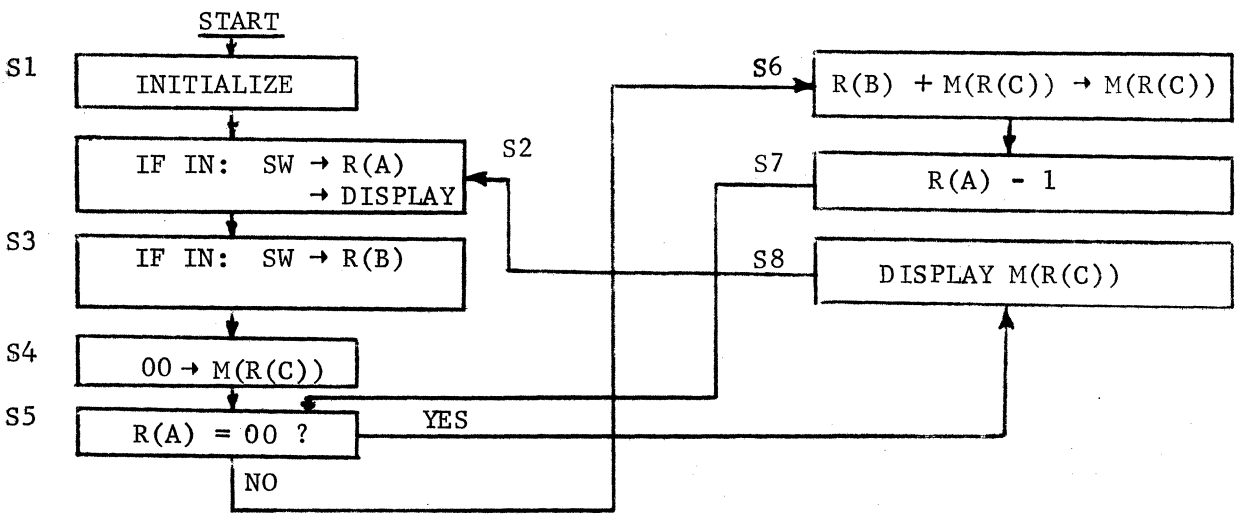
	M	CODE
—	00	00
S1	01	F8
	02	00
	03	A3
	04	F8
	05	03
	06	AA
	07	F8
	08	CC
	09	E3
	0A	53
	0B	60
	0C	23
S2	0D	3F
	0E	0D
	0F	68

	M	CODE
S3	10	F8
CODE	11	C3
	12	F3
	13	32
	14	1B
S4	15	2A
	16	8A
	17	3A
	18	0D
S5	19	F8
	1A	EE
S6	1B	53
	1C	60
	1D	30
	1E	1D

APPENDIX 5-D

Two-Byte Multiply Program (33 Bytes)

Enter two bytes. After pressing IN for second byte the product of the two bytes is displayed. This program is limited to a one byte product.



APPENDIX 5-D: (Continued)

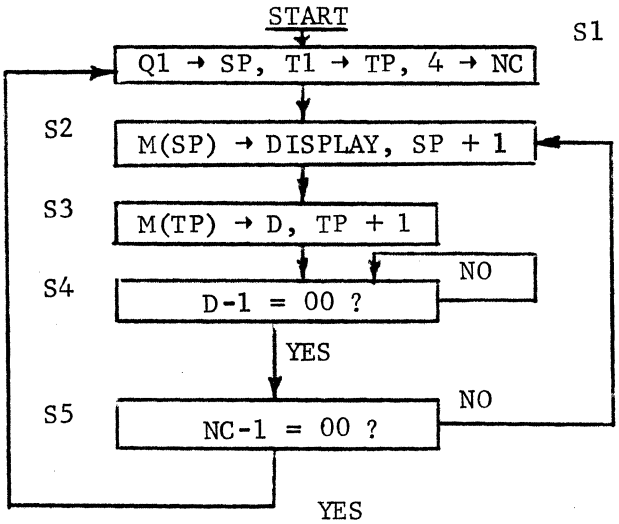
	M	CODE
M(C)	00	00
S1	01	F8
	02	00
	03	AC
	04	EC
S2	05	3F
	06	05
	07	68
	08	F0
	09	AA
	0A	60
	0B	2C
S3	0C	3F
	0D	0C
	0E	68
	0F	F0
	10	AB

	M	CODE
S4	11	F8
	12	00
	13	5C
S5	14	8A
	15	32
	16	1D
S6	17	8B
	18	F4
	19	5C
S7	1A	2A
	1B	30
	1C	14
S8	1D	60
	1E	2C
	1F	30
	20	05

APPENDIX 5-E

Table-Driven Sequencer Program (23 Bytes + Tables)

Generates repeated output byte sequence via a table. Can be used with external latches or relays as multiple waveform generator or multiple line controller. Simplified version of sample program in COSMAC microprocessor manual.



R(A) = state table pointer (QP)
R(B) = time table pointer (TP)
R(3) = counter (NC)

	M	CODE
—	00	00
S1	01	F8
	02	17
	03	AA
	04	F8
	05	1B
	06	AB
	07	F8
	08	04
	09	A3
S2	0A	EA
	0B	60
S3	0C	4B
S4	0D	FF
	0E	01
	0F	3A
	10	0D

	M	CODE
S5	11	23
	12	83
	13	3A
	14	0A
	15	30
	16	01
Q1	17	11
Q2	18	22
Q3	19	33
Q4	1A	44
T1	1B	80
T2	1C	FF
T3	1D	80
T4	1E	FF

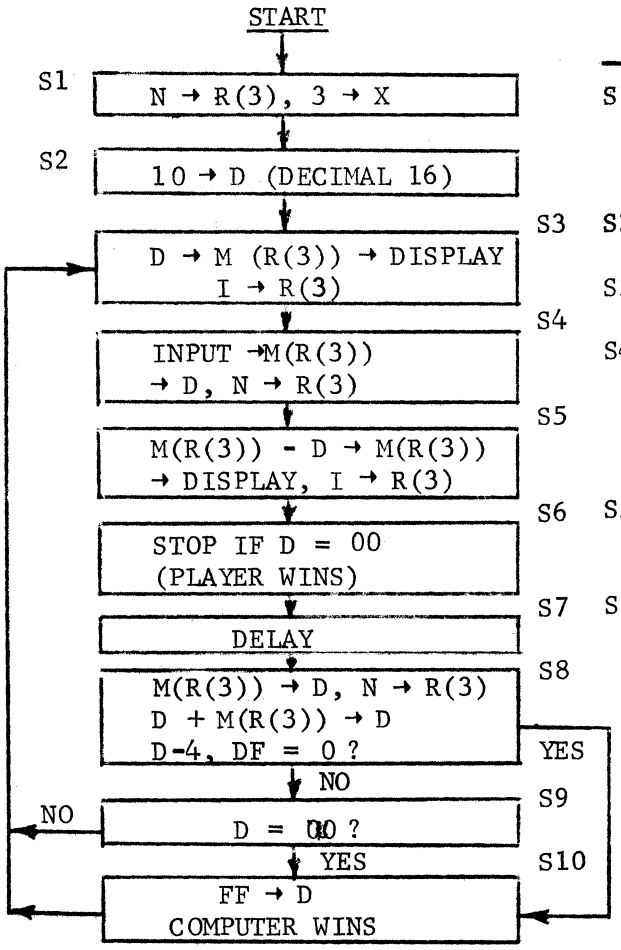
APPENDIX 5-E: (Continued)

Notes: A. Change Q1-Q4 to vary output state sequence.
B. Change T1-T4 to vary intervals between state changes
C. To expand tables
M(02) = Q1 address
M(05) = T1 address
M(08) = number of bytes in Q/T table. Both must be same length
D. Add C2 (.005 μF) for slower clock if desired

APPENDIX 5-F

NIM Computer Playing Program (38 Bytes)

Each game is initiated by pressing clear and then start, then 10 (Decimal 16) will be displayed. You take turns with MICROTUTOR in this game. You go first. Enter 01, 02, or 03 which will be subtracted from display. Whoever can reach 00 first, wins. (When MICROTUTOR wins, FF is displayed). You must get exactly 00 to win. No cheating checks are included. Change byte at M(06) for different starting number. Add C2 (.005 μF) for longer delay between your move and computers.



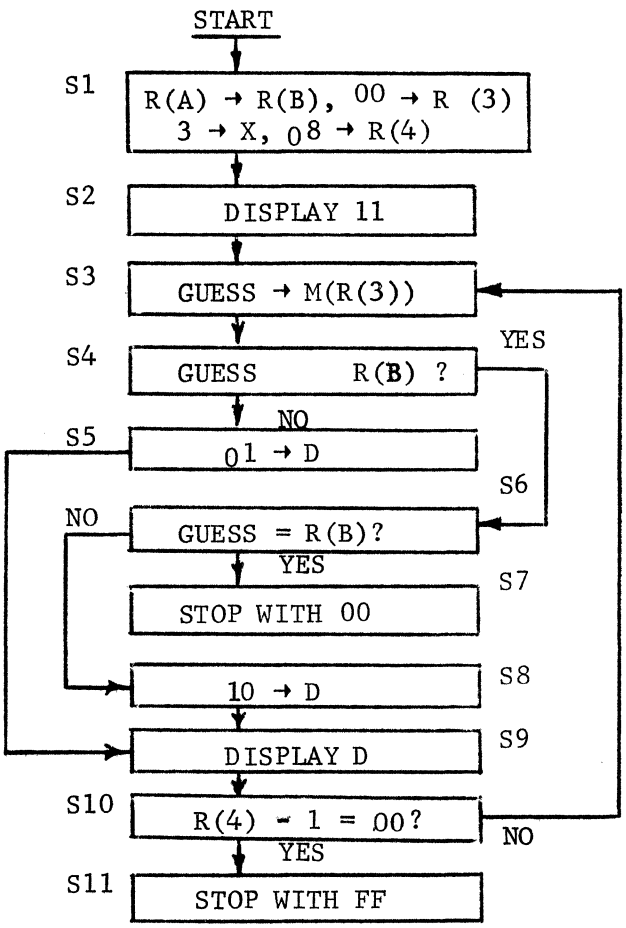
	M	CODE
—	00	00
S1	01	F8
	02	26
	03	A3
	04	E3
S2	05	F8
	06	10
S3	07	53
	08	60
S4	09	3F
	0A	09
	0B	68
	0C	F0
	0D	23
S5	0E	F5
	0F	53
	10	60
S6	11	32
	12	11

	M	CODE
S7	13	F8
	14	FF
	15	FF
	16	01
	17	3A
	18	15
S8	19	F0
	1A	23
	1B	F4
	1C	FF
	1D	04
	1E	3B
	1F	22
S9	20	3A
	21	07
S10	22	F8
	23	FF
	24	30
	25	07

APPENDIX 5-G

Guess My Number Program (48 Bytes)

MICROTUTOR thinks of a 1-byte number. You try to guess it in seven turns. Display = 11 to start. Enter guess. If display = 10, your guess was high. If 01, your guess was low. If 00, you got it. After seven tries, FF indicates you lost. Change M(08) byte to vary number of guesses permitted.



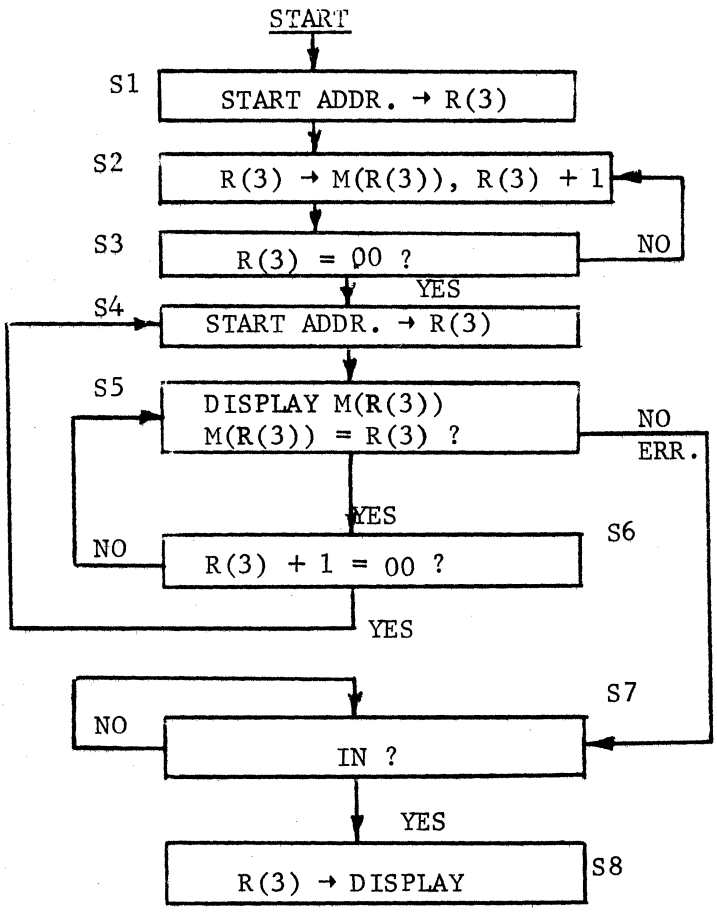
	M	CODE
—	00	00
S1	01	8A
	02	AB
	03	F8
	04	00
	05	A3
	06	E3
	07	F8
	08	08
	09	A4
S2	0A	F8
	0B	11
	0C	53
	0D	60
	0E	23
S3	0F	1A
	10	3F
	11	0F
	12	68
S4	13	8B
	14	F5
	15	33
	16	1B

	M	CODE
S5	17	F8
	18	01
	19	30
	1A	23
S6	1B	3A
	1C	21
S7	1D	53
	1E	60
	1F	30
	20	1F
S8	21	F8
	22	10
S9	23	53
	24	60
	25	23
S10	26	24
	27	84
	28	3A
	29	0F
S11	2A	F8
	2B	FF
	2C	30
	2D	1D

APPENDIX 5-H

Memory Address Test Program (35 Bytes)

Stores a unique byte at each memory location then reads and checks for proper storage. Use only for 256 byte RAM. Non-changing display is error byte. Press in to see error address. The program can be tested with a good RAM by changing 23 @ M(0A) to 22. This will yield an immediate error at M(22) with the error byte = 21.



	M	CODE		M	CODE
—	00	00	S6	12	13
S1	01	F8		13	83
	02	23		14	32
	03	A3		15	09
S2	04	53		16	30
	05	13		17	0C
S3	06	83	S7	18	3F
	07	3A		19	18
	08	04	S8	1A	E4
S4	09	F8		1B	F8
	0A	23		1C	00
	0B	A3		1D	A4
S5	0C	E3		1E	83
	0D	60		1F	54
	0E	23		20	60
	0F	F3		21	30
	10	3A		22	21
	11	18			