Questdata
Volume 1 Issue # 7 ©

THE MACHINE IS THINKING WHAT?

Alice had been looking over his shoulder with some curiosity. "What a funny watch!" she remarked. "It tells the day of the month, and doesn't tell what o'clock it is!"

"Why should it?" muttered the Hatter. "Does your watch tell you what year it is?"

–ALICE IN WONDERLAND

EACH microprocessor is a game with rules set by the manufacturer. A computer should have many addressing schemes, says one microprocessor company. A microprocessor needs to have only a few powerful addressing modes, replies another.

Let's continue our study of the inner workings of the COSMAC machinery with a look at the logic instructions. Within the COSMAC's world of logic we get our choice of R(X) pointer access or immediate. The R(X) mode, although it is nice and well suited to searching stacks and arrays, for now we are going to cover logical immediate instructions.

In dealing with the logic operations it is necessary to think in terms of the individual bits which comprise a byte. If we look at the eight cells of a typical byte with the zero bit in the far right hand corner we see:

7	6	5	4	3	2	1	0
0	1	1	0	0	1	0	1

The one's represent high and the zero's low voltage signals. Each 0 and 1 alternative is a discrete logical unit. They can represent on or off, true or false, yes or no. Taken as a group the binary units in the hex number can represent a code in much the same manner as the Morse code fashions a message.

In the immediate mode of addressing we use the D register as a sort of bucket. First we dump something into the bucket, let us call this D(s) where the (s) stands for start. Then we mix the bucket with the contents of the immediate digits (this is the function immediate part–ORI [F9], XRI [FB], ANI [FA] followed by binary digits). The results wind up in the D register bucket. We will call the result D(f) for the final value of D. Here is what a typical logical immediate operation looks like:

The Mad Tea-Party

7	6	5	4	3	2	1	0	BIT #
0	1	1	0	0	1	0	1	D(s)
0	0	0	0	1	1	1	1	Immediate
0	0	0	0	0	1	0	1	D(f)

AND IMMEDIATE

Betty and Jane went to a movie together. This statement is true only if BOTH Betty and Jane went to a movie. If Betty went by herself or took someone else to the movie, then the statement is false and whoever made it is a liar. If the top digit on the far right represents Betty (Bit #0) and the next bit down stands for Jane, then we can consider D(f) to be the answer to the question of whether or not Betty and Jane went to a movie. Since the logical operation in the illustration is an AND function, it looks like Betty and Jane went to a movie. But Fred is a 0 (false) in BIT #1 of D(s), so any statement made about it is bound to be false. Fred and Jack went to the movie is a 0 (false, no, low, off). We are using a convention called POSITIVE LOGIC when we refer to 1 as being the presence of something (a yes, a high, an on), and 0 its opposite. This system seems natural and makes sense.

So when we are talking about the logic instructions, we are referring to the individual bits and finding out

whether the outcome of the logic operation is a 1 or 0 signal. In the case of the AND logic function we have the following truth table:

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

As you can see, the only time we have a 1 (true) as the final result D(f) is when both D(s) and the Immediate value are both 1. In the case where one or the other of these two operands is false then the final value is false. To help you think in terms of bits (binary digits), a Hex chart of equivalent values is included in this section. If you happen to have one of the very first Elf's with a discrete LED display, then you will see the results directly without having to translate them. In some instances it is OK to think of the AND being performed upon a hex number. In our AND example we have changed the left-most nibble of 65 to 0. The result, 05 can be considered a masking out of the high nibble. Indeed, any number in the high nibble of D(s) would have turned to 0 in the D(f) final result. Now, anytime you want to make an upper nibble disappear, all you have to do is AND with 0F (0000 1111).

What would you do if you wanted to make the lower nibble disappear? Right, AND with F0 (1111 0000) and whatever was in the lower nibble is turned to 0 hex. So much for MASKING.

What if you want to test to see if one individual bit is on (1)? If bit number 6 is the one you are after and you wanted to branch to a special routine if this bit is on, how would you go about it? You know that the digit 0 will wipe out 1's in the result. What you do is find a number with the bit configuration 0100 0000 and AND with that number. It turns out that [redacted] hex will do the job for us. 40

Why would you want to do something like this? One case would be if you wanted to find out some binary piece of information, like owns a pet or doesn't own a pet. If the person does not own a pet the 0 will be ANDed with the 1, and give 0 as a result. If the no. 6 bit position is a 1 then the result will be a 1. Since the result will be 80 or 00, you can easily use BNZ or BZ to branch to your pet ownership routine. Let's say bit 4 is the one you want to test, it represents "owns a home." What hex digit would you use for this test?

The following program is a LOGIC LABORATORY. The lab starts out with our first AND example. You can try any values you want in loc. 01 and loc. 03 to experiment with the results. If you own a VIP or system without a direct LED display of some kind, you will have to look in location 10 hex for the answer. Try any binary combinations you wish. For now, put in the two NOP's which we will be using very soon, and best to stick with function FA (ANI) for the moment.

LOC.	CODE	MNEM.	ACTION
00	F8	LDI	Load starting
01	65		D(s)
02	FA	ANI	functions: [FA], [F9], [FB]
03	0F		Immediate digits
04	C4	NOP	two nop's to be used to
05	C4	NOP	complement later logic lab.
06	A4	PLO4	Save answer in R4.0
07	90	GHI0	Initialize R5.1
08	B5	PHI5	R5.1=00 if page 0
09	F8	LDI	Initialize pointer to ans.
0A	10		
0B	A5	PLO5	thus ans. to appear loc. 10
0C	E5	SEX5	Set the X Register to 5
0D	84	GLO4	put answer back in D reg.
0E	55	STR5	Store ans. via 5 pointer (loc.10)
0F	64	OUT 4	Display what X is pointing to
10	00		ANSWER LOCATED HERE!
11	00	HLT	this 00 tells 1802 to HALT

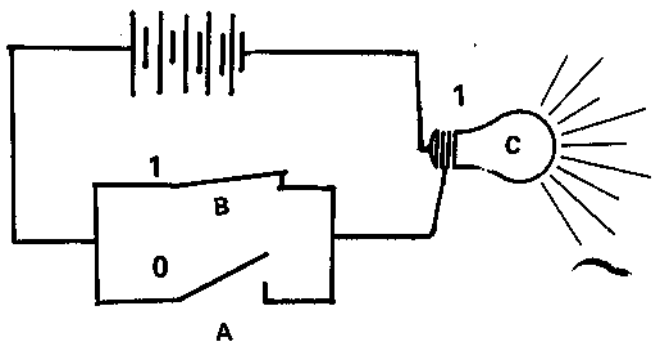
LOGIC LABORATORY

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

Binary and Hexadecimal

INCLUSIVE OR IMMEDIATE

To make our logic instruction discussion more meaningful, QUESTDATA presents a physical analogy to the INCLUSIVE OR:



It is not necessary to build this circuit, but if you want to–go ahead. If A is closed OR B is closed the light bulb C will turn ON. If both A and B are ON then C will be ON. The only time the bulb will be turned OFF (0), will be when both A and B are OFF (0). The logic of this situation is reflected in the truth table below:

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

The sentence "Fred is going to paint his house blue or red," could be represented by the truth table if Fred plans on painting his house blue or red or BOTH. He might decide he wants the windows outlined in red against the blue background. My feeling is that he is better off with just blue or red since the BOTH condition does not really make for a beautiful house. The point is that the OR condition includes the BOTH condition. It is this both condition that makes for a plain vanilla OR. The RCA 201 manual refers to this OR as just an OR, but it is really an inclusive OR because in includes the BOTH condition.

Inclusive OR is used to add bits to already existing data, without changing anything except the desired bit or bits. One way that you could change the hex number 08 into the ASCII code for 8 (which is 38) is to use the OR function. The whole operation would look something like this:

7	6	5	4	3	2	1	0	Bit No.
0	0	0	0	0	0	0	0	D(s)
0	0	1	1	0	0	0	0	Immediate
0	0	1	1	1	0	0	0	D(f)

INCLUSIVE OR ILLUSTRATION

Thus, if we perform an ORI [F9] upon 08 and 30 we get 38 as the result. Our ASCII printer will now recognize the code and print 8. Perhaps you think it is pretty silly that the ASCII code for 8 is hex 38. If you want some real laughs, all you need to do is take a look at the IBM EBCDIC code. There is even a joke that goes along with the EBCDIC code. It goes, "What kind of a code does the worlds biggest computer company speak?" Answer: "Any code they wish." There are, of course, reasons for all codes but since we are discussing logic, we must save that analysis for another time.

The usual way of converting codes is with a table lookup but the OR method could be used. Actually, there is a lot of personal preference in programming. That is to say you can choose to use any one of a number of ways of going about constructing a program. If your program achieves the desired results and met whatever specifications have been set for it, then it is OK code. With memory prices dropping all the time, there is no real reason to compact code to the Nth degree.

To try the OR function in the LOGIC LABORATORY, just put 08 in loc. 01 for D(s) and put 30 in loc. 03 for the Immediate value. The function ORI is the machine code F9 and it goes in location 02.

Inclusive OR is used to add bits to already existing data, without changing anything except the desired bit or bits.

If you would like to try complementing, a topic which will be covered in more detail in next QUESTDATA, you are free to do so. Briefly you place FB in location 04 and FF in location 05. What you get is the opposite or NOT of what you put in. If you are experimenting with the AND function, you will get a NAND (stands for NOT AND). If you are working with OR, you will find the LEDs displaying the NOR of the function. Try making up a truth table for the NAND and NOR functions. When hardware people link up 7400 gates with 7432 gates, they are linking a NAND with an OR. These types of hardware things can all be done in software using your COSMAC. One of the advantages to software, as you know doubt know, is that you don't have to put in new gates or rewire thing to change a program. It is almost axiomatic to say that anything that can be done in hardware is possible in software. True, there are certain things that require very high Hz that are not possible in purely software, but in general the software-hardware interchangeability holds true.

Next QUESTDATA we will cover the EXCLUSIVE OR which is used to make comparisons and complement data.

Since the topic is logic, try your hand at the following Lewis Carroll challenge: "Which is more accurate –a clock which is stopped or a clock which loses a minute a day?" The answer to this brain cruncher is on page 11.

In the construction of games, the logic instruction plays an important part. The games in this issue bear looking into for the facile uses to which logic instructions are put. Games are fun but you might find the study and creation of game programs even more fun.

YOU CAN BEAT THE DEALER

By Edgar Garcia

If you own an Elf microcomputer with 2K or more RAM and an 1861 video display, you can play Blackjack with the accompanying program. This program along with the program "A Tic-Tac-Toe Game for Your Elf Computer" by E. M. McCormick in the November 1978 *Popular Electronics* are two machine language programs which you can show off to friends the next time you are asked the inevitable question, What does it do?"

Program execution starts at memory location 00 00 and the program requires up to location 06 6B for data storage. After initialization of the appropriate registers, the program counter is changed to RF to free R0 as the DMA pointer. The next 24 bytes form a routine which places 52 card addresses (each two bytes in length) in memory starting at location 06 04 and ending at location 06 6B. The card addresses are from 04 00 to 05 CB and each address differs from its preceding address by nine. This is due to the fact that each playing card consists of eight bytes for the display and one for the numerical value.

Next, the value 34 in hex is placed in memory location 01 20 to allow the card fetch subroutine to run through a full deck of 52 cards. The card fetch subroutine, when called upon, runs through a full deck of card addresses until the INPUT switch is depressed. When the input switch is depressed and released, a card address selected by R3 is placed in R7. In order to keep from obtaining the same card address in a future card selection, the address that was just chosen is replaced by the card address following it in memory effectively throwing that card out of the deck. The next card address is replaced by the address following it, etc., until every card address following the card address selected is moved in memory. Next memory location 01 20 is reduced by one to tell the computer to run through only 51 card addresses next time the subroutine is called. Although this procedure seems to be a little complicated, it is necessary to give every card an equal chance of being chosen.

When there are only 13 cards left in the deck, the Q light comes on indicating that the game in progress should be completed and then a new deck should be opened. A new deck is opened simply by resetting the computer and starting it again.

The interrupt subroutine is standard and places the 256 byte page whose high order address is 03 on the display.

The display subroutine takes the card address stored in R7 after a card is fetched and displays it in the location specified by R8. Each card or message i.e. HT? STD, etc. is displayed in an 8 bit by 8 bit grid with the data for these displays taking the most part of the memory needed by this program. This data is stored in memory from 04 00 to 06 03.

The Ace/bust/draw subroutine is called after a card is displayed and it takes the value of the card, located at the end of the 9 byte data string, and adds it in memory location 01 BE for the player or in location 01 BF for the dealer (computer). In addition the value of the card is checked to see if it is an ACE (0B). If it is, the three highest order bits of the corresponding memory location, 01 BE or 01 BF, have 01 added to them for future use. Next, the score located in memory is checked to see if it is over 21. If it is, the highest order three bits are checked to see if there is an Ace. If there is, the Ace is counted as a 1 instead of an 11 and the over 21 condition goes away. If there is no Ace, the subroutine is exited at the appropriate location to display a bust (BST) ending the game. In addition, the subroutine keeps a check on the dealers score and as soon as it goes over 16, but is less than 21, the subroutine is exited at the proper location to check to see who won the game.

The main program is explained in detail in the accompanying flow diagram and needs only a few comments. A card is fetched under control of the INPUT button, thus making it random in the sense that cards are selected by the player and not by the computer which could follow a pattern.

Upon starting the game, the INPUT button is depressed three times and three cards are displayed. The first card displayed is the dealers, the next two are for the player. After the second card for the player is displayed, the screen shows HT? which is the Elf's way of asking you if you want a hit. Unfortunately, you can't simply nod your head as is done in Las Vegas because the Elf doesn't understand that gesture, so what you must do is put 00 on the hex keyboard or toggles and press the INPUT button upon which the HT? is extinguished and another card is displayed when the INPUT button is pressed again. To indicate that you want to stand, simply put a nonzero byte on the keyboard and press INPUT upon which the screen displays STD where the HT? was and the game is ended. Of course, anytime his total is over 21 a BST is displayed and the game is over.

When the player decides to stand, the next few presses of the INPUT button control the cards chosen by the dealer and are displayed on the upper half of the screen. The game ends when the dealer gets a total of 17 or larger or when he busts. There is even provision for a draw game which is my favorite display. Any time a game ends, the next time the INPUT button is pressed, the screen is cleared and a new game starts. If, upon pressing the INPUT button, a strange looking display is placed in a card location, this means that the computer has run out of cards and needs to be reset to obtain a full deck. If the dealer gets five cards and his total is less than 17, he automatically wins. This has only happened once for me in a few hundred games. If his total is 17 or larger, the computer checks to see who is closest to 21.

With this program and a book of Blackjack strategy, and with lots of practice, maybe the next time you go to Las Vegas you can get lucky and break even. Anyway, at least you can impress friends the next time they ask that infernal question.

BLACKJACK

INITIALIZATION

```
00 00  F8 00 B2
   03  A4 BF
   05  F8 01 B1
   08  B6 B9 BA
   0B  BB BC
   0D  F8 02 A1
   10  F8 03 B8
   13  F8 04 B4 A3
   17  F8 06 B3
   1A  F8 1F A9
   1D  F8 20 A6
   20  F8 56 AA
   23  F8 69 AC
   26  F8 FF A2
   29  F8 2D AF
   2C  DF
   2D  F8 34 A5
   30  94 53 13
   33  84 53 13
   36  14 14 14
   39  14 14 14
   3C  14 14 14
   3F  25 85
   41  32 45
   43  30 30
   45  F8 34 56
   48  C0 01 C0
```

INTERRUPT SUBROUTINE

```
01 00  72 70
   02  22 78
   04  22 52
   06  C4 C4 C4
   09  F8 03 B0
   0C  F8 00 A0
   0F  80 E2
   11  E2 20 A0
   14  E2 20 A0
   17  E2 20 A0
   1A  3C 0F
   1C  30 00
```

CARD FETCH SUBROUTINE

```
01 1E  DF
   1F  F8 34 A5
   22  F8 06 B3
   25  F8 04 A3
   28  37 32
   2A  13 13
   2C  25 85
   2E  32 1F
   30  30 28
   32  37 32
   34  43 B7
   36  43 A7
   38  03 23 23
   3B  53 13 13 13
   3F  03 23 23
   42  53 13 13 13
   46  25 85
   48  3A 38
   4A  06
   4B  FF 01
   4D  56
   4E  FB 0D
   50  3A 1E
   52  7B
   53  30 1E
```

DISPLAY SUBROUTINE

```
01 55  DF
   56  F8 08 B5
   59  07 58 95
   5C  FF 01 B5
   5F  32 55
   61  17 88
   63  FC 08 A8
   66  30 59
```

ACE/BUST/DRAW SUBROUTINE

```
01 68  DF
   69  17 8B
   6B  FB BE
   6D  32 9C
   6F  07 FB 0B
   72  3A 77
   74  F8 2B 38
   77  07 EB F4 5B
   7B  0B FA 1F
   7E  FF 16
   80  3B 8B
   82  0B FA E0
   85  3A 96
   87  1F 1F
   89  30 68
   8B  0B FA 1F
   8E  FF 11
   90  3B 89
   92  1F 1F
   94  30 87
   96  0B FF 2A 5B
   9A  30 7B
   9C  07 FB 0B
   9F  3A A4
   A1  F8 2B 38
   A4  07 EB F4 5B
   A8  0B FA 1F
   AB  FF 16
   AD  3B B6
   AF  0B FA E0
   B2  3A B8
   B4  1F 1F
   B6  30 68
   B8  0B FF 2A 5B
   BC  30 A8
```

MAIN PROGRAM

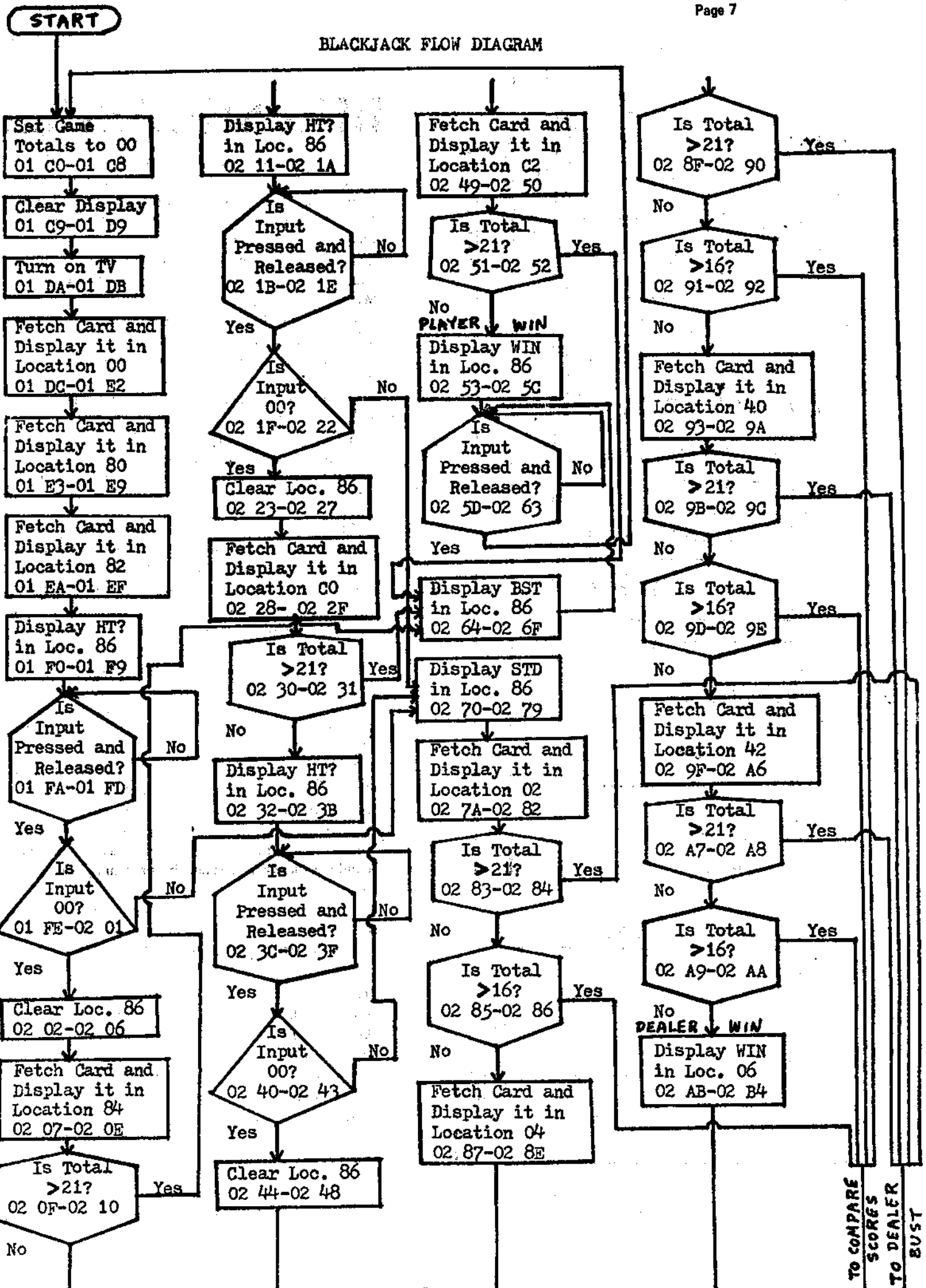
```
01 BE  00 00
   C0  F8 BE AB
   C3  F8 00 5B
   C6  1B 5B 2B
   C9  F8 03 B4
   CC  F8 FF A4
   CF  F8 00 54
   D2  84
   D3  32 DA
   D5  FF 01
   D7  A4
   D8  30 CF
   DA  E2 69
   DC  1B
   DD  D9
   DE  F8 00 A8
   E1  DA DC
   E3  2B
   E4  D9
   E5  F8 80 A8
   E8  DA DC
   EA  D9
   EB  F8 82 A8
   EE  DA DC
   F0  F8 05 B7
   F3  F8 D4 A7
   F6  F8 86 A8
   F9  DA
   FA  3F FA
   FC  37 FC
   FE  E2 6C
02 00  3A 70
   02  17
   03  F8 86 A8
   06  DA
   07  D9
   08  F8 84 A8
   0B  DA DC
   0D  30 11
   0F  30 64
   11  F8 05 B7
   14  F8 D4 A7
   17  F8 86 A8
   1A  DA
   1B  3F 1B
   1D  37 1D
```

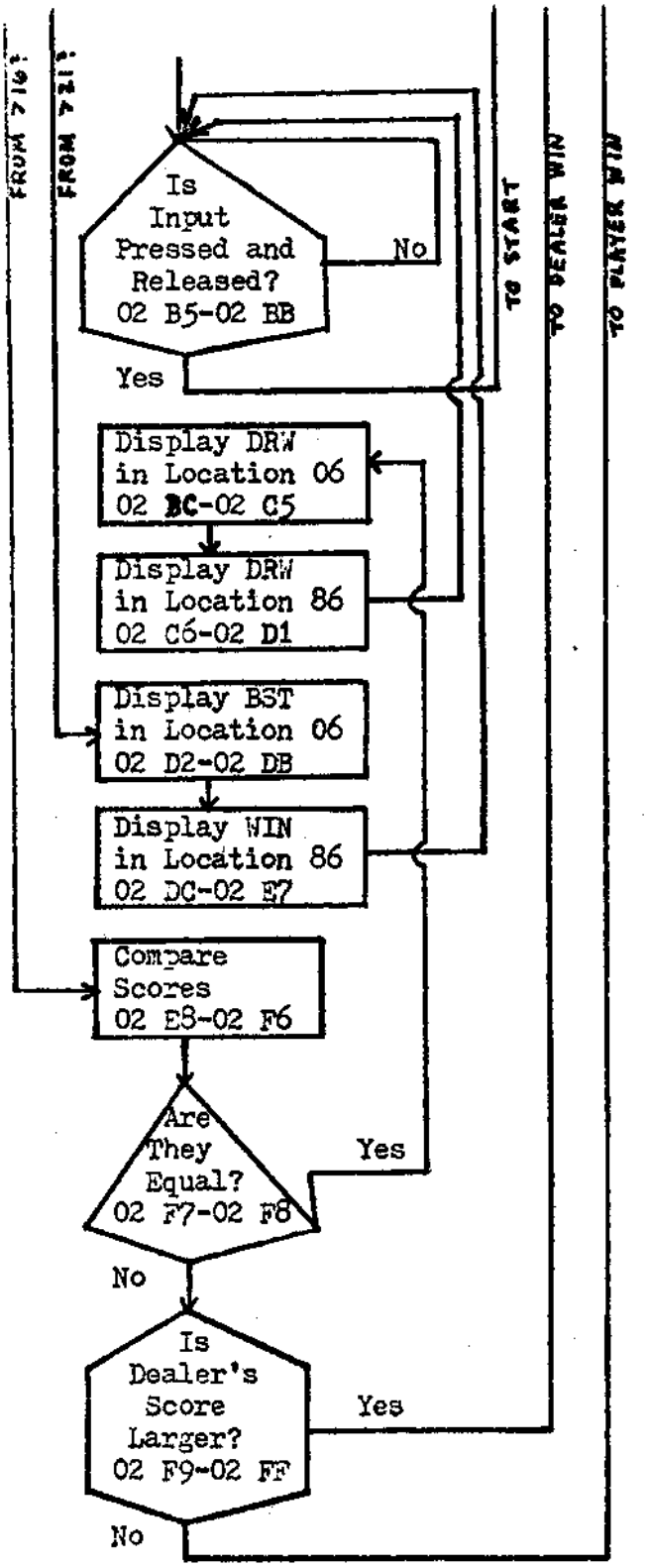
```
02 1F E2 6C
   21 3A 70
   23 17
   24 F8 86 A8
   27 DA
   28 D9
   29 F8 C0 A8
   2C DA DC
   2E 30 32
   30 30 64
   32 F8 05 B7
   35 F8 D4 A7
   38 F8 86 A8
   3B DA
   3C 3F 3C
   3E 37 3E
   40 E2 6C
   42 3A 70
   44 17
   45 F8 86 A8
   48 DA
   49 D9
   4A F8 C2 A8
   4D DA DC
   4F 30 53
   51 30 64
   53 F8 05 B7
   56 F8 E4 A7
   59 F8 86 A8
   5C DA
   5D 3F 5D
   5F 37 5F
   61 C0 01 C0
   64 F8 05 B7
   67 F8 EC A7
   6A F8 86 A8
   6D DA
   6E 30 5D
   70 F8 05 B7
   73 F8 F4 A7
   76 F8 86 A8
   79 DA
   7A 1B
   7B D9
   7C F8 02 A8
   7F DA DC
   81 30 87
   83 30 D2
   85 30 E8
   87 D9
   88 F8 04 A8
   8B DA DC

02 8D 30 93
   8F 30 D2
   91 30 E8
   93 D9
   94 F8 40 A8
   97 DA DC
   99 30 9F
   9B 30 D2
   9D 30 E8
   9F D9
   A0 F8 42 A8
   A3 DA DC
   A5 30 AB
   A7 30 D2
   A9 30 E8
   AB F8 05 B7
   AE F8 E4 A7
   B1 F8 06 A8
   B4 DA
   B5 3F B5
   B7 37 B7
   B9 C0 01 C0
   BC F8 05 B7
   BF F8 FC A7
   C2 F8 06 A8
   C5 DA
   C6 F8 05 B7
   C9 F8 FC A7
   CC F8 86 A8
   CF DA
   D0 30 B5
   D2 F8 05 B7
   D5 F8 EC A7
   D8 F8 06 A8
   DB DA
   DC F8 05 B7
   DF F8 E4 A7
   E2 F8 86 A8
   E5 DA
   E6 30 B5
   E8 F8 BE AB
   EB 0B FA 1F
   EE 5B 1B
   F0 0B FA 1F 5B
   F4 2B EB
   F6 F3
   F7 32 BC
   F9 0B 1B
   FB F7
   FC 33 53
   FE 3B AB

04 00 40 A0 E0 AA B5 11 0A 04 0B
   09 E0 20 E0 84 EA 11 0A 04 02
   12 E0 20 E0 2E FB 15 1B 04 03
   1B A0 A0 E0 24 2A 11 15 0A 04
   24 E0 80 E0 2A F5 11 0A 04 05
   2D E0 80 E0 A4 EA 11 0A 04 06
   36 E0 20 20 2E 3B 15 1B 04 07
   3F E0 A0 E0 A4 EA 11 15 0A 08
   48 E0 A0 E0 2A F5 11 0A 04 09
   51 E0 40 40 44 4A 11 0A 04 0A
   5A 20 20 20 AE FB 15 1B 04 0A
   63 E0 A0 A0 A4 EA 51 15 0A 0A
   6C A0 A0 E0 CA B5 11 0A 04 0A
   75 40 A0 E0 A4 AA 11 0A 04 0B
   7E E0 20 E0 8E FB 15 1B 04 02
   87 E0 20 E0 24 EA 11 15 0A 03
   90 A0 A0 E0 2A 35 11 0A 04 04
   99 E0 80 E0 24 EA 11 0A 04 05
   A2 E0 80 E0 AE FB 15 1B 04 06
   AB E0 20 20 24 2A 11 15 0A 07
   B4 E0 A0 E0 AA F5 11 0A 04 08
   BD E0 A0 E0 24 EA 11 0A 04 09
   C6 E0 40 40 4E 5B 15 1B 04 0A
   CF 20 20 20 A4 EA 11 15 0A 0A
   D8 E0 A0 A0 AA F5 51 0A 04 0A
   E1 A0 A0 E0 C4 AA 11 0A 04 0A
   EA 40 A0 E0 AE BB 15 1B 04 0B
   F3 E0 20 E0 84 EA 11 15 0A 02
   FC E0 20 E0 2A F5 11 0A 04 03
05 05 A0 A0 E0 24 2A 11 0A 04 04
   0E E0 80 E0 2E FB 15 1B 04 05
   17 E0 80 E0 A4 EA 11 15 0A 06
   20 E0 20 20 2A 35 11 0A 04 07
   29 E0 A0 E0 A4 EA 11 0A 04 08
   32 E0 A0 E0 2E FB 15 1B 04 09
   3B E0 40 40 44 4A 11 15 0A 0A
   44 20 20 20 AA F5 11 0A 04 0A
   4D E0 A0 A0 A4 EA 51 0A 04 0A
   56 A0 A0 E0 CE BB 15 1B 04 0A
   5F 40 A0 E0 A4 AA 11 15 0A 0B
   68 E0 20 E0 8A F5 11 0A 04 02
   71 E0 20 E0 24 EA 11 0A 04 03
   7A A0 A0 E0 2E 3B 15 1B 04 04
   83 E0 80 E0 24 EA 11 15 0A 05
   8C E0 80 E0 AA F5 11 0A 04 06
   95 E0 20 20 24 2A 11 0A 04 07
   9E E0 A0 E0 AE FB 15 1B 04 08
   A7 E0 A0 E0 24 EA 11 15 0A 09
   B0 E0 40 40 4A 55 11 0A 04 0A
   B9 20 20 20 A4 EA 11 0A 04 0A
   C2 E0 A0 A0 AE FB 55 1B 04 0A
   CB A0 A0 E0 C4 AA 11 15 0A 0A
   D4 A7 A2 E2 A2 1C 04 00 08
   DC 00 00 00 00 00 00 00 00
   E4 89 A9 A9 D9 24 34 2C 24
   EC C7 A4 C7 A1 C7 38 10 10
   F4 E7 82 E2 2C EA 0A 0A 0C
   FC C7 A5 A7 A6 C5 22 2A 36
```

BLACKJACK FLOW DIAGRAM





Back issues of QUESTDATA are available for $1.50 each, beginning with issue No. 1. Programming knowledge is cumulative.

BUG SQUASHER

[Alas, the following typographical error is more scary to an editor than 50K haunted houses. I really goofed, nightmare city. Mike Tyborski's fine program *Patterns* just rolls over and plays dead when you enter the object code on page 10 of issue no. 6. Mike has my apology for the typo in location 19—the object code at this location should be C4, not CA as listed. My thanks to Eugene Jackson, John Bradshaw, and others for the discovery of the goof. Everyone was very nice about it. Niel Wiegand solved the problem and commented, "I have enjoyed working with Mr. Tyborski's program. I consider the challenge of debugging a piece of code to make it run properly one of the most rewarding exercises leading to programming skill." Niel, the letter-writer below, showed innovation in coming up with even more "Patterns."

—Bill Haslacher]

I loaded the *Patterns* program shown on page 10 of issue number 6 and had two problems you should make your readers aware of. First, the object code shown for the NOP at location 19 hex should be C4 hex. The second problem will only show up on some systems depending on how R6 is used before entering a program. Mr. Tyborski turned on the video interface without first completely initializing Register 6. This will result in a byte in memory somewhere being changed to FF hex. To fix the problem merely rearrange the code beginning at location 36 hex to: F8EE A6 69.

I am an incurable modifier of programs. I made three changes to Mr. Tyborski's *Patterns* program to make a more interesting display. First, change the OR instruction at location 83 hex to XOR. Now, bits will toggle rather than coming on and staying on. Next, combine successive displays by branching to location 35 hex instead of 28 hex at location 92 hex. Finally, I randomized the initial random number by incrementing a register as part of the interrupt routine. These changes are summarized below:

LOC.	CODE
0083	F3
0092	3035
0010	F8B6
001F	1A

LOC.	CODE
0036	86
	52
	F8EF
	A6
	8A
	30A0

Thank You,
Neil Wiegand
Austin, TX

HUNT THE HERMIT

By Richard Moffie

HUNT THE HERMIT is a number guessing game where you try to locate a hermit in an 8x8 grid. Guess one of the numbered squares and the computer tells you how far away you are (but not which direction). The distance is the sum of horizontal and vertical distances, and not the diagonal distance (think of city blocks where you must go along streets and cannot cut across yards). You have nine tries to guess the number. If you are correct, a tone sweep will be heard, followed by a reset to play again. If you don't guess correctly in 9 guesses (can be changed in byte 18), the Q light comes on and the secret number is displayed.

The program begins by using the number in R9 as the secret number (the hermit's hiding place) and masking it with 77 hex to obtain a number between 00 and 77. Then 11 hex is added so that the number is between 11 and 88 (each digit is between 1 and 8). Because of this, the player does not need to think in terms of hex numbers. By ANDing with F0, and shifting, the two digits can be separated and stored in RB and RC. R4 holds the number of guesses allowed (9). R9 is made random by incrementing during each loop while the computer is waiting for a key to be pressed. Because of the speed with which this is done, there is no way to predict the number.

When a guess is made, its digits are also separated, and then each is subtracted from the corresponding digit in the secret number. If the result is negative, it is complemented to obtain a positive difference. The two differences are then added and the result is displayed. This tells you how far you are from the hermit, but not which direction to go. If the result is 0, you guessed the number and the program branches to the tone sweep subroutine (which appeared in Sept. 1976 *Popular Electronics*), before returning to the start of the program. If the result is not 0, the number of guesses left is decremented and – if you have guesses left – it loops back for the next one. If you are out of guesses, the Q light is turned on and the secret number is displayed. To play again, press RESET and GO.

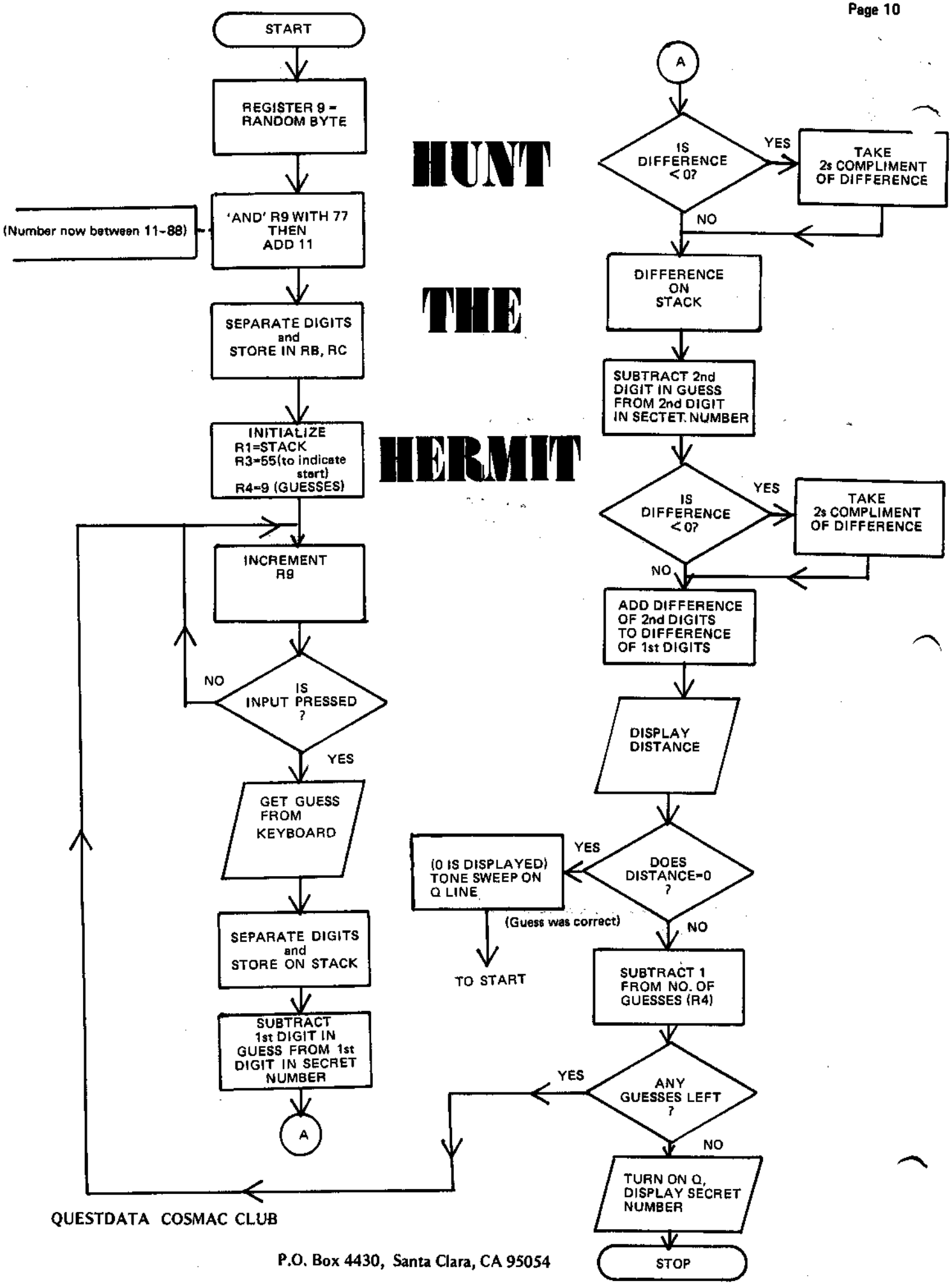
The display indicates 'SS' (actually '55') at the beginning to indicate START and show that the computer is ready to play. The program will run in an expanded memory Elf as written.

Address	Data	Comments
0000	30 67 C4	Jump to init. RX.1's
03	FC 11 AD	Mask to set boundaries 11-88
06	FA 0F AB	Separate digits
09	8D FA F0	
0C	F6 F6 F6 F6 AC	
11	F8 55 A3 A1	Stack pointer
15	53 E3	
17	F8 09 A4	No. of guesses
1A	64 23	
1C	29 3F 1C	Randomize
1F	37 1F 6C	Get guess
22	23 FA 0F 73	Separate digits
26	01 FA F0	
29	F6 F6 F6 F6 53	
2E	8C F7 33 36	Compare 1st digits
32	FB FF FC 01	
36	53 13	
38	8B F7 33 40	Compare 2nd digits
3C	FB FF FC 01	
40	23 F4 53	Combine distances
43	64 13	Display result
45	32 56	If guess correct, to tone
47	24 84	Decrement guesses left
49	3A 1C	Back for next guess
4B	7B 8D 53	Out of guesses, turn Q on &
4E	64 23 00	display number
51	00 00 00 00 00	Stack
56	F8 FF A6	Tone sweep subroutine
59	7A 86 A7	
5C	27 87 3A 5C	
60	31 59	
62	7B 26 86	
65	3A 5A	
67	F8 00 B1 B3 B4 B6	Init. RX.1's
6D	B7 89 FA 77 30 03	

Sample Grid

11	12	13	14	15	16	17	18
21	22	23	24	25	26	27	28
31	32	33	34	35	36	37	38
41	42	43	44	45	46	47	48
51	52	53	54	55	56	57	58
61	62	63	64	65	66	67	68
71	72	73	74	75	76	77	78
81	82	83	84	85	86	87	88

HUNT THE HERMIT



QUESTDATA COSMAC CLUB

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

MUSIC ALGORIGHM

QUESTDATA would like to greatfully acknowledge *Popular Electronics* permission to reprint Ed McCormick's algorithm for music. This program originally appeared in *Popular Electronics* February 1978 issue in the article "How to Upgrade a Basic ELF microcomputer." *Popular Electronics* is responsible for starting the 1802 on its way to winning the hearts of programmers and experimenters. They periodically publish excellent programs and information about the COSMAC. Owners of the 1802 are thankful to them for the work they have done.

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

Publisher . . . . . **Quest Electronics**
Editor . . . . . **Bill Haslacher**
Technical Coordinator . . . . . **Jane McKennon**
Proofreading . . . . . **Ken Brown**

ANSWER TO CARROLL LOGIC PROBLEM

(answer to puzzler on the bottom of page 3)

This is somewhat of a trick question.

According to Carroll the clock that has stopped is more accurate than the one that loses a minute a day. The first clock is exactly right twice every twenty-four hours, but the other clock is exactly right only once in two years. "You might go on to ask," Carroll adds, "'How am I to know when eight o'clock does come? My clock will not tell me.' Be patient: you know that when eight o'clock comes your clock is right; very good; then your rule is this: keep your eyes fixed on the clock and the very moment it is right it will be eight o'clock." There, now isn't that a good example of straightforward logical thinking? Well, sort of? Let's see a clock . . .

Loc.	Code	Mnem.	Comments
0000	E5	SEX 5	E=5
01	F8 45	LDI	Put first data
03	A5	PLO5	*in reg 5
04	F0	LDX	Repeat music
05	3A 09	BNZ	*if data is 00
07	30 00	BR	*
09	A8	PLO 8	Store duration
0A	15	INC 5	*in reg 8
0B	64 25	OUT 4	Display pitch
0D	F0 A7	LDX PLO	store it in reg 7
0F	F8 06	LDI	Store tempo
11	A9	PLO 9	*in reg 9
12	87	GLO 7	Stop alternating
13	FC B4	ADI	*Q if a rest
15	33 1F	BPZ	*
17	31 1C	BQ	If Q on, turn
19	7B	SEQ	*it off; if
1A	30 1F	BR	*off, turn it
1C	7A	REQ	*on
1D	30 1F	BR	*
1F	87	GLO 7	Repeat as often
20	FF 01	SMI	*as pitch
22	3A 20	BNZ	*indicates
24	89	GLO 9	Repeat as often
25	FF 01	SMI	*as tempo
27	A9	PLO 9	*indicates
28	3A 32	BNZ	*
2A	88	GLO 8	Repeat as often
2B	FF 01	SMI	*as duration
2D	A8	PLO 8	indicates
2E	3A 0F	BNZ	*
30	30 3A	BR	*
32	C4 C4	NOP's	Delay to make
34	30 36	BR	*paths take
36	30 38	BR	*same time
38	30 12	BR	*
3A	7A 15	REQ INC5	When note done,
3C	F8 0E	LDI	*turn off Q; insert
3E	B3	PHI 3	*short quiet
3F	23 93	DEC3 GHI3	*interval between
41	3A 3F	BNZ	*notes
43	30 04	BR	Get next note

TCHAIKOVSKY

By Ted Ramirez

Here is a program for classical music fans. The tempo byte (Hex location 10) should be 08 for this program. Also, byte 3D should be changed to 01 to give a more flowing sound (the entry F8 0E gives a staccato sound to this musical selection).Using the *Popular Electronics* lowered algorithm, the data starts at location 45. The theme from Piano Concerto #1 in B flat by Peter Tchaikovsky is the musical selection I chose. The score has been modified somewhat in order not to exceed the 256 byte memory of the Super Elf.

Loc.	Data
45	3A 15
47	3A 1B
49	3A 1F
4B	5D 24
4D	6E 1B
4F	3A 1F
51	6E 19
53	3A 1B
55	3A 27
57	3A 24
59	5D 2D
5B	6E 1F
5D	3A 1B
5F	6E 2D
61	31 1F
63	31 27
65	31 2D
67	32 33
69	6E 1F
6B	3A 1B
6D	6E 15
6F	3A 1F
71	3A 1F
73	3A 1B
75	6E 15
77	3A 1B
79	3A 12
7B	3A 15
7D	83 1F
7F	3A 27
81	3A 12
83	3A 15
85	3A 19
87	3A 1F
89	5D 24
8B	6E 1B
8D	3A 1F
8F	6E 19
91	3A 1B
93	3A 27
95	3A 24
97	5D 2D
99	6E 1F
9B	3A 1B
9D	6E 15
9F	3A 19
A1	3A 1B
A3	3A 19
A5	3A 1B
A7	3A 27
A9	3A 1B
AB	3A 22
AD	3A 1F
AF	3A 2D
B1	3A 1F
B3	3A 27
B5	3A 24
B7	27 3F
B9	27 2D
BB	27 27
BD	27 33
BF	27 2D
C1	27 27
C3	27 24
C5	31 1F
C7	31 1B
C9	31 1F
CB	31 1B
CD	6E 2D
CF	3A 27
D1	93 24
D3	3A 15
D5	3A 1B
D7	3A 1F
D9	5D 24
DB	6E 22
DD	3A 1B
DF	6E 19
E1	3A 24
E3	3A 14
E5	3A 19
E7	5D 1F
E9	6E 14
EB	3A 15
ED	6E 24
EF	3A 15
F1	3A 1B
F3	3A 1F
F5	5D 24
F7	6E 22
F9	3A 1B
FB	6E 19
FD	6E 27
FF	00

COSMAC CRICKET

By Mark Wendell

In order to achieve a surprisingly realisitic cricket "chirp" on a COSMAC Super Elf or other COSMAC computer with a speaker and amplifier, I modified the music program in *Popular Electronics* Feburary 1978 issue, page 68. The lowered Mc Cormick algorithm changes are as follows:

Location 00 to 0E remain the same, change:

Loc.	Data
0F	F8 02

Location 11 to 3B remain the same, change:

Loc.	Data
3C	F8 06

After this has been loaded into your Elf, load in the "chirp" starting at location 45:

Loc.	Data
45	11 03
47	11 03
49	11 03
4B	11 03
4D	11 03
4F	11 03
51	10 02
53	10 02
55	10 02
57	10 02
59	10 02
5B	10 02
5D	B0 4C
5F	00 00

To run the program all you need to do is make the modifications, load the data and run. Jimminy Cricket, eat your heart out!

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

BULK RATE
U.S. Postage Paid
QUEST
Electronics
Permit No. 649
Santa Clara, CA

ADDRESS CORRECTION REQUESTED

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