PROGRAMS

for the

COSMAC ELF

MUSIC

and

GAMES

for noncommercial use only
Oct. 20, 2010 PCM

Programs for the Cosmac Elf Music and Games

Paul C. Moews

List of Sections

List of Programs

Published January 1978 by Paul C. Moews.

Manufactured in the U.S.A.

2 3 4 5 6 7 8 9 10

Introduction

The following programs were written for the basic COSMAC 1802 "Elf" as described in the August and September, 1976 issues of Popular Electronics. However the programs will also run in expanded memory systems like the 1 1/4 K "Elf" described in the March, 1977 issue of Popular Electronics. While the programs in this booklet are recreational in nature it is hoped that they may also aid in learning programming. The programs are documented to make them easy to follow and some programming exercises are suggested with hints to aid in their solution.

The 1/4 K "Elf" is a very small machine and cannot accommodate an editor or assembler. Lacking an assembler one must work directly with the instruction set and RCA's assembly language mnemonics are not very helpful. Therefore mnemonics are not used; the machine code is listed together with enough information to make the programs understandable. One of the difficulties of writing directly in the instruction set is that programs can not be easily relocated in memory. To make this task somewhat easier the programs are located starting at memory address 00 which makes all memory addresses relative to 00. Thus if a program is to be placed in memory starting at location 30_{16} every instruction which is a relocatable memory address must have 30_{16} added to it.

Subroutines have been located in the high order portion of memory leaving low order memory for operating systems and programs. The subroutines are called by the "SEP register technique" (see RCA's User Manual for the 1802 Cosmac Microprocessor pp. 54-58) in which the program counter is changed to a register dedicated to a subroutine and back to the main program counter on return. Register 0 is used as the program counter and all of the programs can be run just as they are listed. However none of the code uses register 3 and it can be used as an alternate program counter. Remember to change any subroutine returns from D0 to D3 if you change the program counter from register 0 to register 3.

To make the programs work in expanded memory systems high order address bytes have to be initialized and all such bytes are set to 00. If an expanded memory system is available and one wishes to run in a 1/4 K memory block other than 00 the high order address bytes must be changed. If only a 1/4 K "Elf" is available these instructions can be replaced by do nothings (C4's) or the code rewritten to omit them.

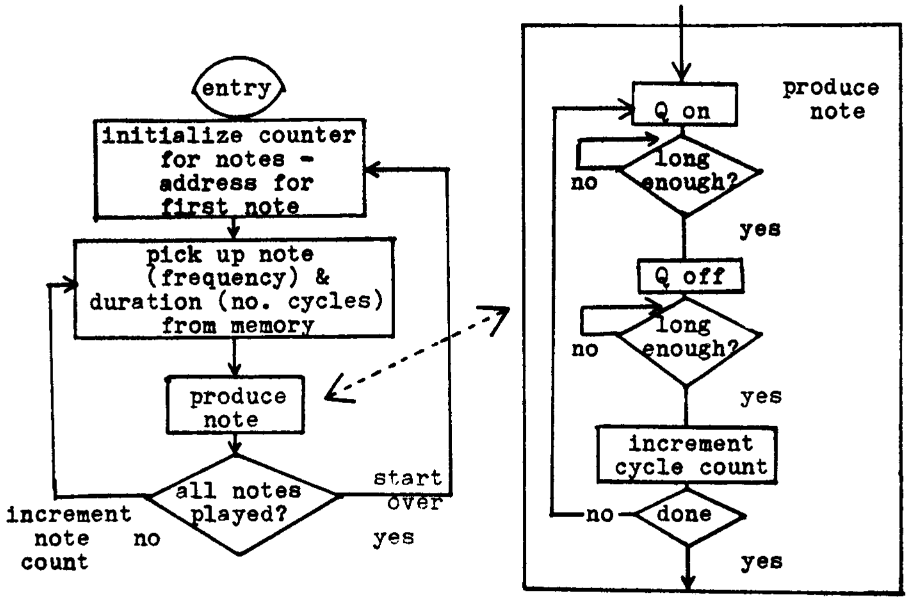
Music

The "Elf" can be programmed to play music by using the Q line, attached to a speaker, to produce tones of the proper frequency and duration. Lacking a speaker an AM radio can be brought close to your "Elf" and tuned between stations in order to hear the music. A difficulty to using a radio is that rests can not be obtained; any timing loop will produce a tone.

In the programs that follow the musical notes are stored in a table in memory; each note takes 3 bytes of store. The first byte designates one of 255 frequencies, if the byte is 00 silence (a rest) occurs. The duration of the note is specified by the next two bytes. The program plays music by picking up the notes one after the other from the table and producing the right tone for the proper period of time. The use of two bytes to time the note makes the program simpler and allows the program to be used as a timer by playing a series of long rests. As well, one of the hex digits in the four used to designate the length of the note could be used to code the timbre or quality of the note and a way to do so will be described.

A flow chart for the simple music program is shown on page 5 and the program listing is given on page 10.

Tones are produced by turning the Q line on and off to produce a square wave. The delays while Q is held on or off



Flow Chart - Music Program

are obtained by successively subtracting one from the D register i.e.

	7B AR(N)	Q on, load D from a register
address	FF 01	subtract one from D
	32 address	back to subtract one more if D is not equal to zero
	7A AR(N)	Q off, load D
	etc.	

For an "Elf" with a 2 Mhz crystal this pair of instructions takes 0.000016 seconds and if the D register is set to 71_{10} (47_{16}) for both Q on and Q off a note with a frequency of about 440 Hz is produced (A above middle C). By counting the number of times Q is turned on and off we can control the duration of the note.

In order to write music we have only to set up a table of values to be transferred to D to produce the proper frequencies and a table of the number of cycles to be counted to obtain notes of the right length. Such a table is given on page 9. Note that 256_{10} (100_{16}) is added to the number of cycles in the duration count because the high order byte is the one checked to see if the proper number of cycles has been produced.

To use this table music from any source is obtained (simple piano or recorder music is suitable) and coded from the table. Let's try it with the first few notes of Beethoven's Fifth symphony:

	Table Entry	
	note	duration
1/8 rest	00	01 3C
1/8 G	50	01 62
1/8 G	50	01 62
1/8 G	50	01 62
1/2 D#	64	02 37
1/8 rest	00	01 3C
1/8 F	5A	01 58
1/8 F	5A	01 58
1/8 F	5A	01 58
1 D	6A	03 6D

The music program as listed on page 10 will play these 10 notes. A number of tunes are given following the program. They can be played with the same program by changing the number of notes as explained in the program listing and loading the "music" into memory. (Note: If your "Elf" operates at 1.8 Mhz the music will sound just as good because it is the ratios between the frequencies of the notes that the ear is sensitive to. However the above table can be easily modified to suit most operating frequencies.)

One can of course use this music program to make the "Elf" a programmable doorbell or music box. The program is designed to run in ROM and a much simpler system would be sufficient for such uses. One can also use the program as a timer by coding a number of long rests (00 00 00 is the longest possible rest, ca. 267 seconds with a 2 Mhz crystal). The two notes 47 02 B8 and 00 39 3C would produce A notes 1 second long and rests 60 seconds long.

This simple music program can be modified in many interesting ways. Two will be described; a program is given for one and a way in which a second might be written is described.

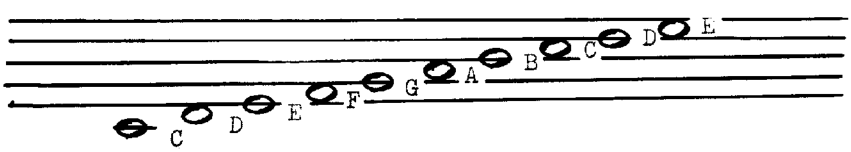
A modification of the music program containing a table of 8 entries that can be used to speed up or slow down the music in successive playings of a tune is given on page 11. In this version of the program the "speed" table specifies the number of times to decrement the duration count between cycles of the note. An entry of 01 in the speed table leaves the music unchanged; an entry of 02 shortens the notes by half, that is half notes become quarter notes. Entries in the speed table can be changed as wished to program the "speed" of the music. The listed tunes can be played with this program by placing the tables of music in memory starting at M(00 45) instead of M(00 30).

The second example is left to the reader to implement. In the simple music program the tones produced are square waves with the Q line being turned on and off for the same length of time. One can modify the program to change this by making the number subtracted from the D register when the Q line is on different from the number subtracted from the D register when the Q line is off. The timbre or quality of the note can be varied in this way to give more variety to the music. The most significant hex digit of the higher order byte used to control the duration of the note could be used to effect this change on a note to note basis. For example the high order byte might be the number (less one which would allow the music in the tables to run unaltered) to be subtracted from the D register when Q is held on. The location of the number to be subtracted could be held in a register (location 17_{16} in the simple music program as written). The high order byte of the cycle count would be picked up with a load via X (F0) instruction instead of a load via X, advance (72) and anded against 15_{10} (FA 0F) before being placed in the high part of register C. The high order byte could be loaded again, this time with the advance instruction (72), D shifted right four times (F6 F6 F6 F6), one added (FC 01) and the result written in the location corresponding to 17_{16} (5N where N indicates the register loaded with the address of the number to be subtracted). To make this program work properly the instruction at location 18 would have to be changed from 3A to 33. Can you see why?

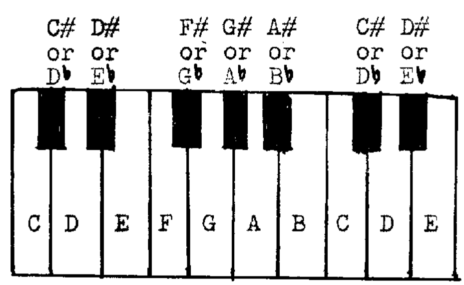
Note Table

note	Hz	machine cycles (10)	machine cycles (16)	Duration$_{16}$ 1/2 note	Duration$_{16}$ 1/4 note	Duration$_{16}$ 1/8 note
C	262	120	78	02 06	01 84	01 42
C#	277	113	71	02 15	01 88	01 45
D	294	106	6A	02 26	01 93	01 4A
D#	311	100	64	02 37	01 9B	01 4E
E	330	95	5F	02 4A	01 A5	01 53
F	349	90	5A	02 5D	01 AF	01 58
F#	370	85	55	02 72	01 B9	01 5C
G	392	80	50	02 88	01 C4	01 62
G#	416	75	4B	02 A0	01 D0	01 68
A	440	71	47	02 B8	01 DC	01 6E
A#	466	67	43	02 D2	01 E9	01 75
B	494	63	3F	02 ED	01 F7	01 7C
C	523	60	3C	03 0B	02 06	01 83
C#	554	56	38	03 2A	02 15	01 8B
D	588	53	35	03 4B	02 26	01 93
D#	622	50	32	03 6D	02 37	01 9B
E	664	47	2F	03 98	02 4B	01 A6
Rest	244	--	00	01 E4	01 7A	01 3C

Musical Scale



Piano Keyboard



Simple Music Program

Address	Code	Notes
00	F8 0A A8	load number of notes (here 0A (base 16) or 10 (base 10)) to R(8).0 change this number for other music
03 06	F8 00 BA F8 30 AA	starting address of note table loaded to R(A)
09	EA F0 A7	load first note to D and also to R(7)
0C	64 28	display first note and decrement number of notes
0E	72 BC 72 AC	load number of cycles for note to R(C)
12	87 32 1A	get note back, if it = 00 skip Q on loop
15	7B	turn Q on
16	FF 01 3A 16	waiting loop for Q on
1A	7A 87	turn Q off, load note again
1C	FF 01 3A 1C	waiting loop for Q off
20	2C 9C	decrement cycle count, load
22	3A 12	to 12 to check for rest and begin next cycle if note not over
24	88 3A 0A	here if note done, all notes done? no go to 0A
27	30 00	yes to 00 to start over
29 2D	xx xx xx xx xx xx xx	unused
30 33 36 39 3C 3F 42 45 48 4B	00 01 3C 50 01 62 50 01 62 50 01 62 64 02 37 00 01 3C 5A 01 58 5A 01 58 5A 01 58 6A 03 6D	table of music

Music which is Programmed to Change Speed

Address	Code	Notes
00	F8 FF AF	initialize R(F).0 to FF, this register is incremented to get successive speed table entries
03 05	1F 8F FA 07	increment R(F), load to D and to get a value between 0 and 7
07	FC 3C AB	add starting address of speed table and save in r(B).0
0A	F8 0A A8	no. notes to R(8).0, here 10 (base 10) to try with Beethoven's 5th, change for other music
0D	F8 00 BA BB	initialize hi order bytes
11	F8 45 AA	load starting address of notes
14 17 19	EA F0 A7 64 28 72 BC 72 AC	as in simple music program
1D	0B AD	get entry from speed table, put it in R(D).0
1F 22 23 27 29 2D	87 32 27 7B FF 01 3A 23 7A 87 FF 01 3A 29 2C 9C	as in simple music program
2F	32 37	if D = 0, note over go to 37
31	2D 8D	decrement R(D), load it
33	3A 2D	back to decrement R(C) again if R(D) not zero
35	30 1D	back to 1D to start next cycle
37	88 3A 15	here if note done, to 15 for next note
3A	30 03	here if all notes done, to 03 to get new speed table entry
3C 40	02 03 04 05 04 03 02 01	speed table entries, change to suit
44	xx	unused
45 - 62		Beethoven's 5th or put other music starting here

Music

Buttermilk Hill

22_{16} notes (34_{10})

Address	Code
30	3C 02 06
33	3C 02 06
36	3F 01 F7
39	3F 01 F7
3C	47 01 6E
3F	47 01 6E
42	47 01 DC
45	5F 02 4A
48	50 02 26
4B	5F 01 53
4E	50 01 C4
51	5F 01 A5
54	50 01 C4
57	50 01 C4
5A	78 01 84
5D	00 01 18
60	6A 01 4A
63	5F 01 A5
66	5F 01 A5
69	5F 01 A5
6C	6A 01 4A
6F	78 01 42
72	5F 01 A5
75	47 01 DC
78	3C 03 0B
7B	3C 01 83
7E	3C 01 83
81	3C 02 06
84	3F 01 F7
87	3C 01 83
8A	3F 01 7C
8D	47 02 B8
90	47 02 B8
93	00 01 70

Streets of Laredo

$2F_{16}$ notes (47_{10})

Address	Code
30	3C 02 06
33	3C 02 88
36	43 01 75
39	47 01 DC
3C	43 01 E9
3F	3C 02 06
42	43 01 E9
45	47 02 4A
48	50 01 62
4B	5A 01 AF
4E	5F 01 A5
51	78 01 84
54	78 01 84
57	5A 02 06
5A	5F 01 53
5D	5A 01 AF
60	50 01 C4
63	47 01 DC
66	43 01 E9
69	47 01 DC
6C	50 01 C4
6F	5A 01 AF
72	50 02 88
75	3C 02 06
78	3C 02 88
7B	43 01 75
7E	47 01 DC
81	43 01 E9
84	3C 02 06
87	43 01 E9
8A	47 02 4A
8D	50 01 62
90	5A 01 AF
93	5F 01 A5
96	78 01 84
99	78 01 84
9C	5A 02 06
9F	5F 01 53
A2	5A 01 AF
A5	50 01 C4
A8	47 01 DC
AB	43 01 E9
AE	47 01 DC
B1	5F 01 A5
B4	5F 01 A5
B7	5A 02 5D
BA	00 01 7A

Music

Kookaburra

26_{16} notes (38_{10})

Address	Code
30	47 01 6E
33	47 01 6E
36	47 01 6E
39	47 01 6E
3C	3F 01 F7
3F	3F 01 7C
42	3F 01 7C
45	47 01 DC
48	55 01 B9
4B	47 01 DC
4E	55 01 B9
51	55 01 5C
54	55 01 5C
57	55 01 5C
5A	55 01 5C
5D	50 01 C4
60	50 01 62
63	50 01 62
66	55 01 B9
69	6A 01 93
6C	55 01 B9
6F	6A 01 93
72	35 03 4B
75	3F 01 7C
78	38 01 8B
7B	35 01 93
7E	3F 01 7C
81	47 02 B8
84	55 01 5C
87	50 01 62
8A	47 01 6E
8D	50 01 62
90	55 01 B9
93	6A 01 93
96	6A 01 93
99	6A 01 93
9C	6A 02 B9
9F	00 01 7A

Run, Boys, Run

20_{16} notes (32_{10})

Address	Code
30	50 01 C4
33	50 01 C4
36	78 01 C5
39	78 01 42
3C	5F 01 53
3F	5F 01 53
42	6A 01 44
45	6A 01 44
48	5F 01 A5
4B	50 01 C4
4E	50 01 C4
51	50 01 C4
54	78 01 C5
57	78 01 42
5A	5F 01 53
5D	5F 01 53
60	6A 01 4A
63	6A 01 4A
66	78 02 06
69	2F 02 4B
6C	3C 02 06
6F	3C 03 0B
72	2F 02 4B
75	3C 02 06
78	3C 03 0B
7B	2F 02 06
7E	3C 02 06
81	3C 02 06
84	47 01 DC
87	47 01 DC
8A	50 01 C4
8D	50 02 88

Subroutines

While programs can be written without them, the use of subroutines has so many advantages that it is difficult to list them all. Most of the subroutines introduced in this section will be used in further programs and the advantages of writing and saving program segments as subroutines will become apparent. The COSMAC microprocessor has an easy way of calling and returning from a subroutine - the SEP register technique. The technique has the drawback that subroutines can only be called from and must return to the main program but in a 1/4 K system this in not an important disadvantage. To use this method one simply changes the program counter to one containing the starting address of the subroutine and on return changes the program counter back to the register used for the main program. It is convenient to reset the subroutine program counter back to the starting address of the subroutine before returning to the main program a bit of "housekeeping" which makes repeated use of the subroutine easy. As an example say register 3 is the main program counter and register 5 is a subroutine register which has been initialized to address s2:

main program			subroutine		
address	code		address	code	
m1	xx		s1	D3	
m2	xx		s2	xx	entry point
m3	D5		s3	xx	
m4	xx	return point	s4	30	
m5	xx		s5	s1	

On leaving the subroutine, a jump to address s1 occurs and the subroutine register is reset to its starting address.

In the following pages four subroutines will be described and each will be preceded by a short program to demonstrate its use. The subroutines will be: 1.) a display subroutine, 2.) a hex (binary) to decimal conversion routine, 3.) a binary coded decimal to hex (binary) conversion routine, and 4.) a routine to examine the contents of any register. These subroutines are written to be stored from memory address 00 93 to memory address 00 FE. It is suggested that an operating system like one of those described in the Popular Electronics articles be used to first place the subroutines in memory. The programs which use the subroutines can then be loaded starting at memory address 00 00.

1.) Display Subroutine

This subroutine displays a number of memory locations a specified number of times. To use it follow the call to the subroutine by two bytes; the first specifies (in hex) the number of times to repeat the display, the second the number of locations to display; return from the subroutine is to the location immediately after the last displayed byte. The program uses the subroutine to display EE 11 FF FF DD EE AA FF FF, ELF DEAF. The last letter of each word is repeated to give a suggestion of the end of a word. The listing for this subroutine in on page 17.

2.) Hex to Decimal Conversion Subroutine

This subroutine converts a byte from hex (binary) to decimal for display. The byte to be converted is passed to the subroutine in the D register and the answer is placed in the F register. The least significant part of the answer (00_{10} to 99_{10}) is placed in R(F).1 and also in the D register and the most significant part of the answer (00, 01 or 02) in R(F).0. The program given as an example uses the subroutine to convert a byte from binary to decimal and displays the least significant portion of the result. The listing for this subroutine is on page 18.

3.) Binary Coded Decimal to Hex Conversion Subroutine

This routine converts a byte from binary coded decimal to hex (binary). The byte to be converted is passed to the subroutine in the D register and the answer replaces it in the

D register. The sample program uses the subroutine to convert a byte from binary coded decimal to hex and displays it. The listing for this subroutine is on page 19.

4.) Subroutine to Examine the Contents of any Register

This subroutine can be used to determine the contents of a specified register. The number of the register to be examined is passed to the subroutine as the low order hex digit of a byte in the D register and the subroutine carries out a transfer. The high order portion of the specified register is transferred to the low order part of the F register and the low order byte of the register is loaded to D. The sample program uses the subroutine to display the low order byte of any of the registers. The listing for this subroutine is on page 20.

As a further illustration of the use of subroutines the program on page 21 is to be used with all four subroutines. If the program is entered with 00 on the switch byte it converts bytes from hex to decimal, if it is entered with 01 set in the switches it converts bytes from decimal to hex, and if it is entered with 02 on the switches it displays the contents of the register specified by the least significant hex digit of the entered byte. While the program was written mostly for fun it can be used as a hex to decimal or decimal to hex converter. As an exercise try to work out what will be displayed when registers 0, 2, 4, 5, 6, 7, 8, 9, E, and F are shown by the subroutine. It is difficult to get them all right.

Another exercise which uses these subroutines is to write a program which takes two binary coded decimal numbers between 00 and 99_{10}, adds them and displays them both as well as their sum. The display might show something like BCD number 1, AD (for add), BCD number 2, EE (for equals), and sum (sum should be displayed as two bytes). One way to add the numbers (after they have been converted to binary) is to store one in M(R(X)), bring the other to the D register, and execute an F4 (add) instruction. Convert the result to decimal and store it for display.

Display Subroutine

Program

Address	Code	Notes
00	F8 00 B5	initialize hi order bytes
03	F8 DE A5	address for display subroutine to R(5)
06	D5 03 09	call display subroutine, display message 3 times, message is next 9 bytes
09 0D	EE 11 FF FF DD EE AA FF FF	message
12	C4	do nothing, return is to here
13	D5 01 01	call routine again to blank display
16	00 00	display 00 and stop

Subroutine

Address	Code	Notes
DC	E2 D0	make R(2) X and return
DE	E0	entry to routine, set R(0) to X
DF	72 AC	times to repeat to R(C).0
E1	72 AA AB	no. bytes to R(A).0 and R(B).0
E4	64	display byte
E5 E8 E9	F8 80 BD 2D 9D 3A E8	delay loop for display
EC	2B	decrement R(B), no. words
ED	8B 3A E4	more bytes? yes back to display
F0	2C	no - decrement R(C), no. times
F1	8C 32 DC	more times? no exit
F4	8A AB	yes, restore no. bytes to R(B).0
F6	2B 20	decrement R(B) and R(0), note: R(0) is program counter
F8	8B 3A F6	loop over no. bytes
FB	8A AB	restore no. bytes to R(B).0
FD	30 E4	repeat message

Hex to Decimal Subroutine

Program

Address	Code	Notes
00	F8 00 B2 B6	initialize hi order bytes
04	F8 BA A6	address for hex to decimal subroutine to R(6)
07	F8 FF A2 E2	X is R(2), M(00 FF) for work
0B	3F 0B 37 0D	wait for in on, off
0F	7B 6C 64 22	Q on, display switch byte
13	D6	call subroutine
14	3F 14 37 16	wait for in on, off
18	7A 52 64 22	Q off display least significant part of byte in decimal
1C	30 0B	go to wait for next byte

Subroutine

Address	Code	Notes
B8	9F D0	least significant part of answer to D, return to main
BA	BF	enter here, byte to R(F).1
BB	F8 00 AB AF	initialize counters
BF	9F	bring byte back to D
C0	FF 64	subtract 100_{10}
C2	3B C7	to C7 if answer less than 0
C4	1F	increment R(F), ends as 0, 1, or 2
C5	30 C0	back to subtract another 100_{10}
C7	FC 64	add 100_{10} and start least significant part
C9	FF 0A	subtract 10_{10}
CB	3B D0	to D0 if answer less than 0
CD	1B 30 C9	increment R(B), no. of tens and back to subtract another
D0	FC 0A	add 10_{10} if we overdid it
D2	BF	save result
D3 D6 D9 DA	8B 32 B8 9F FC 10 2B 30 D2	Add 10 (base 16) to result for every time 10 (base 10) occurred in least significant part - exit to B8 when finished

Decimal to Hex Subroutine

Program

Address	Code	Notes
00	F8 00 B2 B7	same program as hex to
04	F8 A4 A7	decimal subroutine except
07	F8 FF A2 E2	address of subroutine is
0B	3F 0B 37 0D	00 A4 and register used to
0F	7B 6C 64 22	hold subroutine address is R(7)
13	D7	
14	3F 14 37 16	
18	7A 52 64 22	
1C	30 0B	

Subroutine

Address	Code	Notes
A3	D0	return to main
A4	AA	enter here, byte to R(A).0
A5	FA F0	and against F0 to get most significant hex digit
A7	F6 F6 F6	shift right 3 times, now have most significant times 2
AA	73 60	store via X, restore X register
AC	F4 F4 F4 F4	add the number to D 4 times to get 10 times most significant digit in D
B0	73 60	store via X, restore X register
B2	8A FA 0F	bring back byte, and against 0F to get least significant digit
B5	F4	add what we have so far to get final result in D
B6	30 A3	go to exit

Examine Register Subroutine

Program

Address	Code	Notes
00	F8 00 B2 B8	same program as hex to
04	F8 94 A8	decimal subroutine except
07	F8 FF A2 E2	address of subroutine is
0B	3F 0B 37 0D	00 94 and register used to
0F	7B 6C 64 22	hold subroutine address is R(8)
13	D8	
14	3F 14 37 16	
18	7A 52 64 22	
1C	30 0B	

Subroutine

Address	Code	Notes
93	D0	return to main
94	FA 0F	enter here, and against 0F to get least significant hex digit
96	BF	save in R(F).1
97	FC 90	add 90
99	58 xx	R(8) is the program counter, instruction 58 writes the D register to the next location, xx becomes 90 thru 9F
9B	AF	save the result in R(F).0
9C	9F	get the least significant digit back
9D	FC 80	add 80
9F	58 xx	one of 80 thru 8F to location A0
A1	30 93	go to exit

Program to Use All Subroutines

Address	Code	Notes
00 05	F8 00 B2 B4 B5 B6 B7 B8 B9	initialize hi order bytes
09	F8 FF A2 E2	X is 2, M(00 FF) for work
0D	F8 3B A4	address of location which will call subroutine, will be D6, D7, or D8
10	F8 DE A5	address of display subroutine
13	F8 BA A6	address, hex to decimal routine
16	F8 A4 A7	address, decimal to hex routine
19	F8 94 A8	address, examine register
1C	6C FC 01	read switch byte, add 01
1F	FA 03	and against 03
21	32 00	wait for legal switch byte
23	FC D5 54	add to D5 to get D6, D7 or D8 in location 3B
26	F8 48 A9	initialize R(9) to point to M(00 48), last of displayed bytes
29	F8 00 AF	initialize R(F).0 to zero
2C	3F 2C 37 2E	wait for in on, off
30	6C 64 22	read and display switch byte
33 36	F8 80 BE 2E 9E 3A 36	delay loop for display
3A	F0	bring back switch byte
3B	xx	will contain D6, D7, or D8
3C	E9	R(9) becomes X, points to last location of displayed bytes
3D	73 73	write D in locations 48 and 47
3F	8F 73	get low R(F), write in 46
41	E2 7B	R(2) back to X, turn on Q
43	D5 03 03	call display routine to show 3 bytes, 3 times
46	xx xx xx	will contain displayed bytes
49	7A	turn Q off
4A 4D	D5 01 01 00	blank display
4E	30 26	go to wait for next byte
93 - FE		the four subroutines

Random Numbers

Random numbers are used in many areas of mathematics and several uses for a random number generator will be described in this section. In the "Elf" random numbers can be generated by counting up in a register, originally zero, until a definite value is reached, resetting the register to zero and counting up again, etc. If the value of the register is displayed when the in button is pushed a random number between zero and the upper limit of counting will be obtained (the time spent at each possible value in the register must be the same). One can be added to the random number before it is displayed to get a range of numbers from 1 onwards. A program which generates random numbers in the range 1 to 6 (a die throw) is given below.

A Die Throw

Address	Code	Notes
00	F8 00 B2	initialize X register, 2
03	F8 FF A2 E2	M(00 FF) for work
07	F8 00 AA	00 to R(A).0, the register in which counting will occur
0A	37 0A	wait for in off
0C	37 1A	to 1A if in button pushed
0E	1A 8A	increment R(A), load to D
10	FF 06	subtract 06
12	32 17	go to 17 if D equals zero
14	E2	makes all paths equal
15	30 0C	to 0C to check for in on
17	AA	load 00 to R(A).0
18	30 0C	go to 0C to check for in on
1A	8A FC 01	get R(A).0 and add 01
1D	52 64 22	display random no. 01 to 06
20	30 07	back for another go

The preceding program simulated the roll of a single die, the program on page 24 simulates the roll of a pair of dice. On entry, CD is displayed and the Q light comes on. When the in button is pushed once 00 is displayed and the Q light goes off at the second push a roll of dice is displayed and the Q light comes on. Subsequent pairs of pushes on the in button produce additional rolls of the dice.

One advantage of a program of this type over real dice is that modifications can be easily made. For example the program can be changed to simulate 4, 5, 7, 8, or 9 sided dice. It is easier to work out the probable outcome for throws of 4 sided die and confirm the probabilities by repeated simulations using this program than it is to use 6 sided die. If two sided dice are used the program simulates the simultaneous throw of two coins.

There is another way to program the throw of dice and it is suggested that the reader write such a program as an excercise. The 36 possible outcomes for the throws are stored in a table in memory. The simple program on page 22 is altered to generate a number between 00 and 35_{10} (23_{16}). Instead of displaying the number it is added to the starting address of the table of moves and a table entry is loaded to the D register and displayed. Using this method the "odds" can be altered at will by changing the number of table entries to suit.

Random number generators can be used to play games in which the optimal strategy is mixed. Mixed strategy is an idea due to von Neumann in which a player randomly selects alternatives at each move with the aid of a table of probabilities. The use of a random number generator to select moves for two such games will be outlined.

The first game which employs a random number generator in the range 1-100 is called "Guess-It". It was devised by Rufus Isaacs and was first described in the American Mathematical Monthly (Vol. 62, pp. 99-108, 1955). A detailed

Pair of Dice Program

Address	Code	Notes
00	F8 00 B1 B2	initialize hi order bytes
04	F8 FE A1	M(00 FE) work for R(1)
07	F8 FF A2 E2	M(00 FF) work for R(2), X
0B	F8 CD	load CD to D register
0D	52 64 22	display it
10	7B	Q on
11	F8 00 AA	00 to R(A).0, counting register
14	37 14	wait for in off
16	37 24	to 24 if in button pushed
18	1A 8A	increment R(1), load it to D
1A	FF 06	subtract 06, change this number for different sides to dice
1C	32 21	go to 21 if D equals zero
1E	E2	equalize path lengths
1F	30 16	go to 16 to check in on
21	AA	00 to R(A).0
22	30 16	go to 16 to check in on
24	31 31	go to 31 if Q is on
26	E1	otherwise make R(1), X
27	8A F4	get this no., add last one
29	E2 FC 01	X back to 2, add 01 to last no.
2C	52 64 22	display roll of dice
2F	30 10	back for 2 more pushes of in button and next roll of dice
31	8A FC 01	comes here if Q is on, gets random no. and adds one
34	FE FE FE FE	shifts it left 4 times
38	51 7A	stores it, turns Q off
3A	F8 00	display 00
3C	52 64 22	
3F	30 11	back for 2nd push of in button

description is given in the December, 1967 issue of Scientific American (pp. 129-131). Briefly:

It is a game for two players. A shuffled deck with an odd number of cards is used, 11 in the original version, although any number of cards may be employed. The tables given in this section allow up to 13 cards to be used. The cards should be easily recognized (for example ace to jack or a complete suit) and the same number are dealt to each of the players.

The extra card is dealt face down and the object of the game is to identify this card.

During his move a player may either guess which card is hidden or ask his opponent if he has a certain card. A guess at the hidden card ends the game, the player who guesses wins if his guess is correct and loses if it is wrong.

If he asks his opponent a question, e.g. "Have you a ten?", the opponent must reply truthfully. It is then the next players turn to move and he has the same choices. However the same card may not be asked about twice.

The bluffing part of the game is that a player can ask about a card which he holds in his own hand so as to mislead his opponent. If a player never bluffed his opponent would know that if he didn't have a card asked for it must be the hidden card.

Cards that are identified are turned face up on the table, i.e. if a player has a card asked for he plays it face up on the table and if a player was bluffing he turns the card he bluffed about face up on the table when his turn comes again.

The game was completely analyzed by Isaacs, his solution depends on the fact that after each move the game can be considered to be starting over again with a different number of cards. This is reflected in the way cards are turned face up as they are identified.

Isaacs' strategy involved two dials which were spun to serve as random number generators. One, referred to by

Isaacs as the b dial, told player 1 when to bluff and when to guess a possible card. The other dial the c_1 dial, was used by player 2 when he had responded "no" to a question. This dial told him whether to guess that the hidden card is the one just asked about or not.

To use a random number generator we can replace the dials by tables calculated from the equations given in Isaacs' paper, these tables are given on page 27. To use the first table, which replaces the b dial, generate a random number between 1 and 100 (see program below). If it is less than or equal to the table entry corresponding to the current game situation we bluff otherwise we ask in earnest.

Random Number Generator (1-100)

Address	Code	Notes
00	F8 00 B2 B6	initialize hi order bytes
04	F8 FF A2 E2	M(00 FF) for work, R(2) is X
08	F8 BA A6	address of hex to decimal routine
0B	F8 00 AA	as for simple random number generator except that range is 00 to 99_{10}
0E	37 0E	
10	37 1E	
12	1A 8A	
14	FF 64	
16	32 1A	
18	E2	
19	30 10	
1B	AA	
1C	30 10	
1E	8A FC 01	
21	D6	call hex to decimal conversion
22	52 64 22	display result
25	30 0B	back for another try
B8 - DB		hex to decimal conversion subroutine

In this program 100_{10} is displayed as 00.

The second table (equivalent to the c_1 dial) is used in the same way. Generate a random number between 1 and 100. If it is less than or equal to the table entry corresponding to the current game situation, guess that the hidden card is the one just asked about. If we are not to guess the hidden card and opponent has only one card left we guess the remaining unknown card, if he has more than one card we must use the b dial or table to guess or bluff.

The player of course must guess when he has determined the hidden card through his questions.

B-Dial (When to Bluff)

opponents number cards	my number of cards 1	2	3	4	5	6
1)	33	50	50	56	57	60
2)	25	33	37	39	43	44
3)	20	27	29	31	33	35
4)	17	21	24	25	27	28
5)	14	18	20	21	22	24
6)	13	16	17	18	19	20

C_1 Dial (Should I Guess)

opponents number cards	my number of cards 1	2	3	4	5	6
1)	50	50	40	37	33	31
2)	33	33	29	25	23	21
3)	38	33	28	25	22	20
4)	33	31	26	23	21	19
5)	33	29	26	23	20	19
6)	31	28	25	22	18	18

The second mixed strategy game which will be described is the game of "Morra" or the "three fingers game". In this game two players simultaneously show one, two or three fingers and call out one, two or three, the call is a guess of the number of fingers their opponent is holding out. If both call out the correct number, or if both call out the wrong number, the round is drawn. However if one player guesses the number of fingers his opponent has extended and the other guesses wrong, the loser pays his opponent as many dollars as the total of fingers shown by the two players. The optimum strategy for this game was worked out by von Neumann. It is to show one finger and say three 5/12 of the time, to show two fingers and say two 4/12 of the time, and to show three fingers and say one 3/12 of the time. If one hex digit is used to show the number of fingers and the other the guess for the number of fingers the computers opponent is holding up, this can be easily programmed using a table with 12 entries and a random number generator.

The program on the next page will play a game of "Morra" against an opponent. On entry the Q light comes on and when the in button is pushed the computer extends one, two, or three fingers (the more significant hex digit) and says one, two, or three (the less significant hex digit). Its opponent should do the same. However instead of displaying its move the computer stores the result and turns off the Q light. The human player should enter his move to the switch byte and push the in button again. After displaying its opponents move the computer reveals its move and determines the result. If the game is drawn DD 00 is displayed. If the computer wins CC (amount won) is displayed and if its opponent wins AA (amount won) is displayed. After displaying this message 3 times the Q light comes on and the computer is ready for another move. Remember the "Elf" is deaf and blind so it won't cheat you. However using the optimum strategy it can't lose either; the best one can do against it in the long run is to draw. It is suggested that the player and computer start the game with $25.00 each and play until one or other has lost all his money.

"Morra"

Address	Code	Notes
00	F8 00	initialize hi order bytes
02	B2 B5 B6 B7	
06	F8 FF A2 E2	M(00 FF) work for R(2), X
0A	F8 DE A5	address of display routine
0D	F8 9A A6	address for computer wins answer
10	F8 A3 A7	address for opponent wins answer
13	F8 00 AA	initialize random no. register
16	52 64 22 7B	blank display, turn Q on
1A	37 1A	random number generator
1C	37 2A	in range 0 to 11 (base 10)
1E	1A 8A	
20	FF 0C	
22	32 27	
24	E2	
25	30 1C	
27	AA	
28	30 1C	
2A	37 2A	comes here if in pushed, waits for in off
2C	8A FC BA	get random no. add starting address of table
2F	AA 0A	fetch table entry
31	AF	save it in R(F).0
32	7A	Q off
33	3F 33 37 35	wait for in on, off
37	6C 64 22	read opponents move and display it
3A	BF	save move in R(F).1
3B	FA 0F	check is move legal?
3D	32 AD	no go to AD and display
3F	FF 04	message CADD
41	33 AD	

Address	Code	Notes
43	9F	bring back opponents move
44	FA F0	check is move legal?
46	32 AD	no go to AD and display
48	FF 40	message CADD
4A	33 AD	yes - go on
4C	F8 B0 BE	delay to show opponents
4F	2E 9E	move
51	3A 4F	
53	7B	Q on
54	8F	bring back computers move
55	52 64 22	and display it, remember hi byte - no fingers shown lo byte - guess of fingers
58	F8 B0 BE	delay to show computers move
5B	2E 9E	
5D	3A 5B	
5F	7A	Q off
60	F8 00 AB	initialize R(B).0 a marker for win-lose situation R(B).0 will become 0 computer right & opponent right 1 computer wrong & opponent right 2 computer right & opponent wrong 3 computer wrong & opponent wrong
63	8F FA 0F	get computers move, and for guess
66	73 12	save in work
68	9F	get opponents move
69	F6 F6 F6 F6	shift to get no. fingers
6D	AE	save in R(E).0
6E	F7	subtract computers guess
6F	CE 1B C4	skips increment instruction if computer right
72	9F	get opponents move
73	FA 0F	and off guess
75	73 12	save in work
77	8F	get computers move
78	F6 F6 F6 F6	shift to get no. fingers
7C	BE	save in R(E).1
7D	F7	subtract opponents guess
7E	CE 1B 1B	skips increments if opponent right, marker is now set

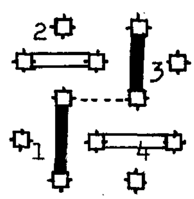
Address	Code	Notes
81	8B	load marker to D to see what result is
82	32 A6	drawn if D=0, go to A6
84	FF 03	subtract 03
86	32 A6	drawn if D=0, go to A6
88 8B 8C	8E 73 12 9E F4 73 12	sum number of fingers and save in work
8F	8B	bring back marker again
90	FF 01	subtract 01
92	32 9D	opponent wins if D=0, go to 9D
94	F0 56	here only if computer wins, load no. fingers and store for display
96 99	D5 03 02 CC xx	display result if computer wins answer in 9A
9B	30 13	return for next round
9D 9F A2	F0 57 D5 03 02 AA xx	store and display result if opponent wins answer in A3
A4	30 13	return for next round
A6 A9 AB	D5 03 02 DD 00 30 13	here if round drawn, display DD 00 and return for next round
AD B0 B4 B7	D5 02 04 CC AA DD DD D5 01 01 00 30 33	here if illegal move, display CADD, blank display and go back for another try at a legal move
BA BD BF C3	13 13 13 13 13 22 22 22 22 31 31 31	table entries for the 12 randomly chosen computer moves
DC - FE		display subroutine

Bridg-it

Bridg-it*, as it was sold some years ago, is a game in which two people tried to build bridges of plastic rods of different colors across a board. One person's bridge makes it impossible for another person to build a bridge in the same place. The winner is the first person to build a bridge completely across the board. An example on a 2 x 2 board is shown below:

black goal

white



white goal

black

The consecutive moves by the two players are numbered 1, 2, 3, and 4 with black playing first. Black will win if he next makes the move indicated by the dotted line.

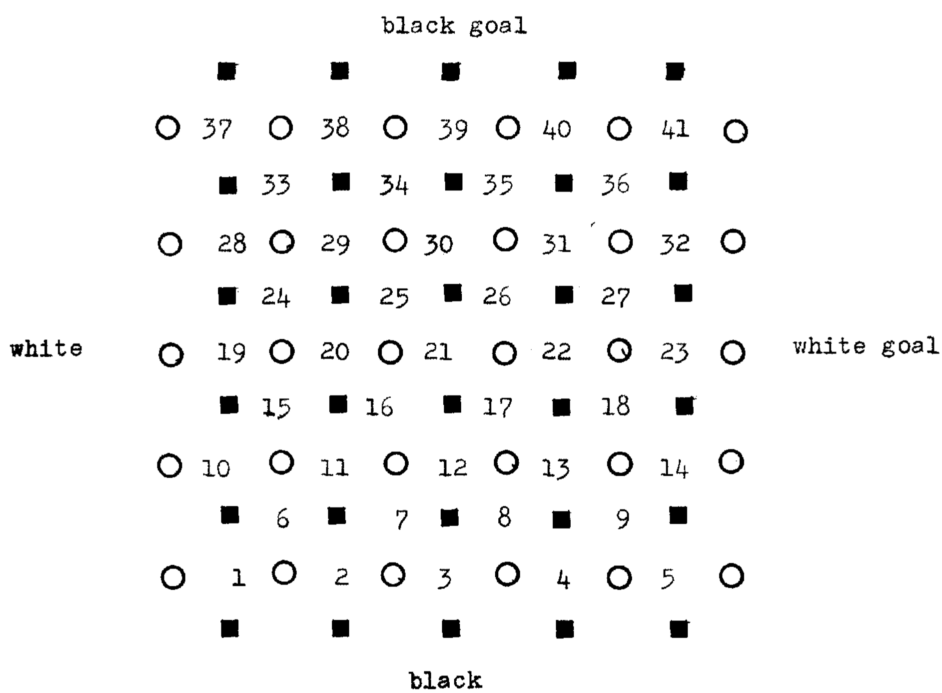
The game as sold was played on a 5 x 5 board which is illustrated on the next page. To make it easier to program the board should be numbered as indicated. Two people can play the game by copying the board to a piece of paper and making moves with different colored pens. The game was analyzed by Oliver Gross (see Martin Gardner's column in the July, 1961 Scientific American) and if both players play perfectly the first player should always win.

The program at the end of this section uses Oliver Gross's winning strategy to play a perfect game. The computer plays black and moves first. On entry to the program the

*Bridg-it is a trade mark of Hasbro, Inc.

computer shows it first move, it invariably chooses 01, and turns on the Q light. White enters his move (moves are entered just as they appear on the board) the Q light goes off and the move is displayed briefly followed by blacks (the computers) move. The program works by finding the proper reply for any possible play in a look-up table.

Although the outcome of the game is not in doubt it is still interesting to follow the computer's strategy. As Gross pointed out, the computer will play badly against a poor opponent and well against a good opponent, but in any case it will always win.



Bridg-it Playing Program

Address	Code	Notes
00	F8 00	initialize hi order bytes
02	B2 B7 BA	
05	F8 FF A2 E2	M(00 FF) for work, R(2) is X
09	F8 A4 A7	address of decimal to hex routine
0C	F8 01	01 to D, computers 1st move
0E	7B 52 64 22	Q on, display D, computers first and subsequent moves
12	3F 12 37 14	wait for in on, off
16	6C 64 22	read and display switch byte
19	AA 7A	save byte in R(A).0, Q off
1B	F8 80	delay for display
1D	BF	
1E	2F 9F	
20	3A 1E	
22	8A	get back switch byte
23	D7	convert it to binary
24	FC 28	add 2 less than starting address of table of moves
26	AA 0A	fetch table entry
28	30 0E	go to display computers move and turn Q on
2A	06 07 08 09	table of replies
2E	02 03 04 05	
32	11 10 16 17	
36	18 19 12 13	
3A	14 15 21 20	
3E	26 27 28 29	
42	22 23 24 25	
46	31 30 36 37	
4A	38 39 32 33	
4E	34 35 41 40	
A3 - B6		decimal to hex subroutine

Reaction Time

This is a program that uses two of the subroutines introduced earlier, the hex to decimal conversion subroutine and the display subroutine.

On entry to the program 00 is displayed. When the in button is pushed, FF is displayed and after a time, which varies randomly, the Q light comes on. The person whose reaction time is being tested pushes the in button as quickly as possible and the time required to do so is displayed in hundreths of a second, i.e. if 30 is displayed 30/100 of a second elapsed between the time the Q light came on and the button was pushed. If the reaction time is 1 second or greater, BB AA DD DD, BAD is displayed; this occurs in any case if the in button is not pushed within 2.56 seconds. If the in button is pushed before the light comes on, CC AA DD DD, CAD is displayed. When the program has finished with one of the above messages, an unchanging 00 is displayed and the program is ready for another try.

The reader might like to modify this program to give some kind of audible or visible reward when a person's reaction time is especially good. This could be done as follows: After displaying the answer instead of reentering the program bring the answer to the D register with an 08 instruction (remember the answer is now in decimal). Subtract some test number from the reaction time (FF xx). If the result is positive do nothing (33 14) but if it is negative give your reward before reentering the program.

Reaction Time Program

Address	Code	Notes
00 04	F8 00 B2 B5 B6 B8 BF	set high order bytes
07	F8 FF A2 E2	M(00 FF) for work, R(2) is X
0B 0E	F8 DE A5 F8 BA A6	address for display routine address of hex to decimal routine
11	F8 64 A8	address for reaction time
14	F8 00 AB	initialize counter for time
17	52 64 22	blank display
1A	7A	Q off
1B	4F FA 0F	pick up random byte and and against 0F
1E	FC 01 AE	add 01, save in R(E).0
21	3F 21 37 23	wait for in on, off
25 27	F8 FF 52 64 22	load FF to D and display it
2A 2D 2E 30 33 35	F8 40 BA 2A 37 4F 9A 3A 2D 2E 8E 3A 2A	start variable delay loops out to 4F if in pushed too soon loop till done no. times thru loops depends on number in R(E).0
37	7B	now done with variable delay turn Q on and start timing
38	F8 E3 AA	load constant for 100 Hz loop E3 for a 2 Mhz crystal CB for a 1.79 Mhz crystal 71 for a 1 Mhz crystal
3B	2A C4	decrement loop counter, no op for additional delay
3D	37 58	out to examine results if in is pushed
3F	8A 3A 3B	done yet? no back to 3B
42	1B	yes, increment 100 Hz loop counter
43	8B 3A 38	and go back for another pass, however go on if 2.56 seconds have elapsed

Address	Code	Notes
46	D5 02 04	use display routine to show
49	BB AA DD DD	BAD
4D	30 14	reenter for another try
4F	D5 02 04	comes here to display message
52	CC AA DD DD	CAD if in button pushed too soon
56	30 14	reenter for another try
58	8B FF 64	here to examine result,
5B	33 46	subtract 100 (base 10) and go to message BAD if 1 or more seconds taken
5D	8B	otherwise get result back again
5E	D6	and convert it to decimal
5F	58	store it in location 64
60	D5 03 02	display result
63	00	
64	xx	result is stored here
65	30 14	reenter program for another try
B8 - DB		hex to decimal subroutine
DC - FE		display subroutine

Tic-Tac-Toe

This version of tic-tac-toe uses an algorithm due to A. G. Bell, "the incomplete defense algorithm". The method is a defensive one, and the computer makes no attempt to win although it will do so on occasion against a clumsy player. There is one defect in the algorithm and if the defect is exploited a knowledgeable player can win one game in three against the machine. A player not familiar with the algorithm seldom beats the machine.

The cells of the tic-tac-toe board are numbered as shown below:

1	2	3
8	9	4
7	6	5

The human opponent plays first. If he plays 1, 2, 3, 4, 5, 6, 7, or 8 the machine replies 9; if he plays 9 the machine replies 1, 3, 5, or 7. Additional machine replies are made using two tables, the first to be used if the machine has the center, the second if the opponent has the center. Each table has 64, 2 bit entries and occupies 16 bytes (see the program listing). In order to determine the response to its opponents move, the computer adds rows in one of the tables which correspond to the squares occupied by its opponent. It then looks at the result and chooses the move which corresponds to the sum with the highest total, if two or more moves are possible one of them is chosen at random. The computer keeps track of occupied squares and will only move in a square if it is vacant.

After loading the program into memory the computer is placed in reset mode. The player sets his first move on the switch byte and changes to run. The player's move is shown and the Q light comes on; on pushing the in button the Q light goes off and the computer displays its reply. The player enters his second move to the switch byte, pushes the in button and his move is displayed with the Q light on; a second push of the in button gives the computers reply, etc. To play a second game reset the computer.

The flaw in the computers play occurs if the player enters 1, 3, 5, or 7 as his first move. The computer invariably replies 9 and the player answers 4 or 6 if his first move was 1; 6 or 8 if his first move was 3, etc. The computer makes its mistake on its reply to the players second move. One out of three times it replies as shown below with c_2, the players winning move is indicated with a w.

p_1		w
c_2	c_1	p_2

p_1	c_2	
	c_1	
w	p_2	

	c_2	p_1
	c_1	
	p_2	w

w		p_1
p_2	c_1	c_2

Tic-Tac-Toe

Address	Code	Notes
00 04	F8 00 B2 B7 BB BC BF	initialize hi order bytes, R(2) is X, R(7).1 is hi order address for update routine, R(B) and R(C) will be used as scratch memory addressing registers, R(F) address of a random byte for random no.s
07 0A	F8 00 A4 AA BA	R(4).0 marker, 0 if player has center, 1 computer has center R(A).0 keeps track of players move, set bit if cell used R(A).1 keeps track of all moves
0C	F8 FF A2 E2	M(00 FF) for work, R(2) is X
10	F8 C0 A7	address of update routine
13	7B 6C	Q on, read switch byte
15	64 22	display players move
17	3F 17 37 19	wait for in on, off

Address	Code	Notes
1B	F8 09 F5	was move 09?
1E	32 2A	yes - go to 2A
20	D7 7A	no, call update routine to record move, turn Q off
22 25	F8 09 52 64 22	reply 09 to move and display it
27 28	14 30 35	01 to R(4).0, computer has center go to wait for players next move
2A	7A	comes here if player has center Q off
2B 2E 31	4F FA 03 FE FC 01 52 64 22	reply 1, 3, 5, or 7 and display it
34	D7	call update routine to record move, renter here in future
35	3F 35 37 37	wait for in on, off
39 3A	7B 6C 64 22	Q on read switch byte and display it
3D	3F 3D 37 3F	wait for in on, off
41	D7 7A	record players move, then turn Q off
--		here starts algorithm to determine computers move
43	84 32 4A	get mark, will be 01 if computer has center
46 48 4A	F8 D7 30 4C F8 E7	load Table 1 address (D7) if computer has center load Table 2 address (E7) if player has center
4C	AB	table address to R(B)
4D	F8 F6 AC	load one less than starting address of sums table in R(C).0
50 53 54 57 59	F8 08 AD 1C F8 00 5C 2D 8D 3A 53	clear the 8 locations of sums table, leave R(C) pointing to last sums table entry
5B	F8 08 AD	set counter to loop over players moves
5E	8A B6 96	transfer record of players moves to R(6).1 and D
61	FA 01	and record against 01

Address	Code	Notes
63	32 92	if player hasn't moved here go to 92 we don't need this entry
65	9A B4	here is player did move, load D with record of total moves
67	EC	R(C) to X, location for sums
68	F8 02 AE	set counter to loop over no. bytes per table row
6B	F8 04 A1	set counter to loop over no. table entries per byte
6E	4B BE 9E	load table entry, transfer to R(E).1, load from R(E).1
71	FA 03 B1	and to strip off single entry, save in R(1).1
74 77 79	94 FA 01 FB 01 32 7D	a move corresponding to this cell? if so set D=0 and go to 7D to zero out this sum
7B 7D	91 F4 73	else get partial sum, add it and store zero or sum
--		now for housekeeping
7E	94 F6 B4	shift record of moves & replace
81	9E F6 F6 BE	shift table entries & replace
85	21 81	increment counter, entries per byte
87	3A 70	if not done go to 70
89	2E 8E	increment counter, bytes per row
8B	3A 6B	if not done go to 6B
8D 90	F8 FE AC 30 94	here if done with row corresponding to this cell, reload sums address and jump to 94
92	1B 1B	skip two bits of table, here if no move made in cell
94	96 F6 B6	shift players record right and save
97	2D 8D	increment main loop over cells
99	3A 60	back to examine next cell if not done
9B	EB	comes here when sums have been collected - now find largest sum, if more than one of same size chose one at random, first set R(B) to X
9C	4F A6	load random no. to R(6).0

Address	Code	Notes
9E	F8 20 AD	load 32 (base 10) to R(D).0 number to be subtracted from sums
A1	2D	decrement R(D), this is test no.
A2	F8 08 AE	load 08 to R(E).0 for loop
A5	16 86	increment random no., load to D
A7	FA 07	and it to get no. 0 to 7
A9	FC 01	add 01 to get no. 1 to 8
AB	52	save it, this is answer if test condition is met
AC	FD FF AB	FF is one more than last sums table address
AF	8D F7	get test no., subtract M(R(B))
B1	32 B9	if result = 0, we have answer go to B9
B3	2E 8E	decrement loop counter
B5	32 A1	if done and no answer back to decrement test no. and start over
B7	30 A5	if not done back for another table entry
B9	E2	here if answer found, R(2) is X
BA	64 22	display answer and go back to
BC	30 34	34 to record move and wait for players next move
--		update subroutine starts here keeps track of moves
BE	E2 D0	comes here to return to main
C0	F8 CE F4	enters here, adds move to CE CE is one less than address of update table
C3	AB EB	save result in R(B), R(B) to X
C5	39 CA	to CA if Q = 0, this is computers move
C7	8A F1 AA	record players move in R(A).0
CA	9A F1 BA	record all moves in R(A).1
CD	30 BE	to BE to return to main
CF	01 02 04 08	update table entries
D3	10 20 40 80	

Address	Code	Notes
D7	6D E6	Table 1, to be used if computer
D9	77 59	has center
DB	DE 66	each line corresponds to a
DD	75 97	row of the table
DF	E6 6D	
E1	59 77	
E3	66 DE	
E5	97 75	
E7	65 67	Table 2, to be used if player
E9	55 5D	has the center
EB	56 76	
ED	55 D5	
EF	67 65	
E1	5D 55	
E3	76 56	
E5	D5 55	
F7	xx xx xx xx	sums table, locations where
FB	xx xx xx xx	sums are stored
FF	xx	work for X register

The tables are the basis of A. G. Bell's algorithm. To determine the computers move look at the player's moves. For example: If the player has moved in cells 3 and 5 add rows 3 and 5 to obtain 8 sums each corresponding to a possible move. Examine the sums and make the move corresponding to the largest sum, if a choice needs to be made chose at random.

i.e. from Table 2

Row 3 is 56 76 and Row 5 is 67 65.

possible moves

	4	3	2	1	8	7	6	5	
Row 3	01	01	01	10	01	11	01	10	(56 76)
Row 5	01	10	01	11	01	10	01	01	(67 65)
Sums	10	11	10	101	10	101	10	11	

The computer would reply with a move in cell 1 or cell 7. Note that sums which correspond to cells where moves have been made are set to zero.

01	01	52	34	103	67	154	9A	205	CD
02	02	53	35	104	68	155	9B	206	CE
03	03	54	36	105	69	156	9C	207	CF
04	04	55	37	106	6A	157	9D	208	D0
05	05	56	38	107	6B	158	9E	209	D1
06	06	57	39	108	6C	159	9F	210	D2
07	07	58	3A	109	6D	160	A0	211	D3
08	08	59	3B	110	6E	161	A1	212	D4
09	09	60	3C	111	6F	162	A2	213	D5
10	0A	61	3D	112	70	163	A3	214	D6
11	0B	62	3E	113	71	164	A4	215	D7
12	0C	63	3F	114	72	165	A5	216	D8
13	0D	64	40	115	73	166	A6	217	D9
14	0E	65	41	116	74	167	A7	218	DA
15	0F	66	42	117	75	168	A8	219	DB
16	10	67	43	118	76	169	A9	220	DC
17	11	68	44	119	77	170	AA	221	DD
18	12	69	45	120	78	171	AB	222	DE
19	13	70	46	121	79	172	AC	223	DF
20	14	71	47	122	7A	173	AD	224	E0
21	15	72	48	123	7B	174	AE	225	E1
22	16	73	49	124	7C	175	AF	226	E2
23	17	74	4A	125	7D	176	B0	227	E3
24	18	75	4B	126	7E	177	B1	228	E4
25	19	76	4C	127	7F	178	B2	229	E5
26	1A	77	4D	128	80	179	B3	230	E6
27	1B	78	4E	129	81	180	B4	231	E7
28	1C	79	4F	130	82	181	B5	232	E8
29	1D	80	50	131	83	182	B6	233	E9
30	1E	81	51	132	84	183	B7	234	EA
31	1F	82	52	133	85	184	B8	235	EB
32	20	83	53	134	86	185	B9	236	EC
33	21	84	54	135	87	186	BA	237	ED
34	22	85	55	136	88	187	BB	238	EE
35	23	86	56	137	89	188	BC	239	EF
36	24	87	57	138	8A	189	BD	240	F0
37	25	88	58	139	8B	190	BE	241	F1
38	26	89	59	140	8C	191	BF	242	F2
39	27	90	5A	141	8D	192	C0	243	F3
40	28	91	5B	142	8E	193	C1	244	F4
41	29	92	5C	143	8F	194	C2	245	F5
42	2A	93	5D	144	90	195	C3	246	F6
43	2B	94	5E	145	91	196	C4	247	F7
44	2C	95	5F	146	92	197	C5	248	F8
45	2D	96	60	147	93	198	C6	249	F9
46	2E	97	61	148	94	199	C7	250	FA
47	2F	98	62	149	95	200	C8	251	FB
48	30	99	63	150	96	201	C9	252	FC
49	31	100	64	151	97	202	CA	253	FD
50	32	101	65	152	98	203	CB	254	FE
51	33	102	66	153	99	204	CC	255	FF