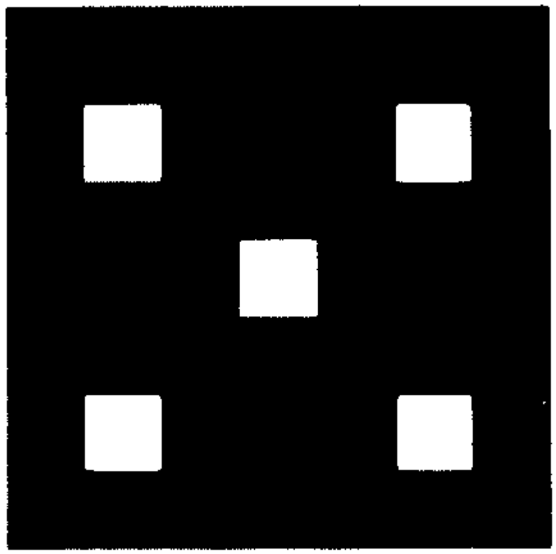
Questdata
Volume 2 Issue #3 ©

ELF  DIE

by
Ron Zoscak

Recently, the local computer club held a "mini-show" to attract new members. The meeting room, at the shopping mall where we meet, was packed with people and machines. You couldn't move without bumping a return key. There were dozens of expensive machines playing Star Trek, Chess, and Star Wars. Printers spat out posters and calendars. Lights blinked, disc drives clicked, and cassette tapes turned. And there I sat with my 256 byte Elf II, suffering the snide REM statements of the TRS-80 and the LSI-11 on either side of me. Talk about frustration! You and I know that the 1802 is the best eight bit microprocessor around, and that an Elf II or Super Elf can do things in a quarter of a kilobyte that other machines need 1K or more to do. But how do you explain that to someone who owns a 32K system with dual floppy drives and a line printer? By demonstrating an interesting program, of course. Unfortunately, at that time my software library and my own programming abilities were rather limited. Now, although my software skills aren't up to writing neat display routines like those of Paul C. Moews in his graphics booklet, I have managed to write an interesting (in my opinion) alternate main program that runs concurrently with a relocated version of his 64 byte display routine.

This program displays a die on the video screen. Rolling of the die is accomplished by pressing and releasing the input button. Upon release of the input button, the low byte of register 9 is copied into the low half of register 6. Since register 9 is incremented once every interrupt while the screen is being refreshed, or approximately 61 times a second, it provides a pseudo random number in the range of 0 to 255 to determine when the die will stop. Because the screen is still being refreshed while the program waits for you to press the input button, the resulting roll will be a random one.

Before loading the program, it is necessary to clear memory. The high bytes of all registers in the program are set to 00 for expanded memory. Punch in the opcodes using the hex keypad. Be sure to load in the codes that occupy locations A0 through DF, as these form the parts of the display that will not be changed by rolling.

Registers 0,1,2,5,C, and F are used by the display routine. Registers 3 and 4 are the program counters for the routines that cycle through the different die patterns and determine when to stop. Registers B,D, and E are used as pointers to the locations on the die face to be changed, and registers 8 and A, hold the patterns that will be written there. Registers 6 and 7 are used by the routine to determine when the die stops.

Elf Die Source Listing

Registers Used:

- X=2
- P=3
- 0=DMA
- 1=Interrupt
- 2=Stack Pointer
- 3=Program counter
- 4=Subroutine
- 5.0=Used
- 6.0=Counter
- 7.0=Delay Counter
- 8.0=Patterns
- 9.0=Keyboard Storage
- A.0=Patterns
- B=Pointer
- C.1=Used
- D=Pointer
- E=Pointer
- F=Counter

ADDR	CODE	COMMENT
0000	F8 12 A1 F8 3A A2 F8 3B	This is a relocated version of the 64 byte display routine by Paul C. Moews, in his booklet "Programs For The Cosmac Elf Graphics". This routine is Copyright and reprinted with permission.
0008	A3 F8 04 BC F8 A0 A5 D3	
0010	72 70 C4 22 78 22 52 9C	
0018	AF 85 BF 91 B0 9F A0 19	
0020	30 25 9F A0 2F A0 8F 32	
0028	2D 9F A0 30 22 9F A0 A0	
0030	9C 34 10 AF 80 BF 30 25	

0038 through 3A are used as the stack for the interrupt routine

ADDR	CODE	COMMENT
003B	E2 69	SET X=2, Turn on TV
003D	F8 0F AA	Store Patterns that will be written
0040	F8 FF A8	onto die face in registers 8 and A
0043	F8 87 A4	Point register 4 to subroutine
0046	3F 46 37 48	Wait here till input button pressed and released
004A	89 A6	Get byte from R9 to determine when to stop
004C	F8 AB AB	Point registers B,D, and E to
004F	F8 BB AD	first place on lines on die face
0052	F8 CB AE	where patterns will be written
0055	88 5B 1B 1B	Write pattern for one dot on die face
0059	5B 5D 1D 1D 5D 5E 1E 1E 5E 2D 8A 5D	
0065	D4	Go to subroutine
0066	8A 5B 2E 2E 5E	Write pattern for three dots on die face
006B	D4	Go to subroutine
006C	2B 2B 8A 5B 1E 1E 5E	Write pattern for five dots on die face
0073	D4	Go to subroutine
0074	88 5D 5B 5E	Write pattern for two dots on die face
0078	D4	Go to subroutine
0079	8A 5B 5E	write pattern for four dots on die face
007C	D4	Go to subroutine
007D	8A 1D 5D 2D 2D 5D	Write pattern for six dots on die face
0083	D4	Go to subroutine
0084	30 4C	Go back, not time to stop yet
0086	D3	Return from subroutine
0087	16 86 32 92	If time to stop, go to location 92
008B	A7 27 87 3A 8C	Not done: short delay
0090	30 86	Go to 86 to return from subroutine
0092	F8 43 A3 D3	Go to 43, hold present pattern, wait for next roll after resetting R4 to subroutine start

ADDR	CODE
00A0	E4 38 0F FF FF FF 00 00
00A8	84 20 0F FF FF FF 00 00
00B0	E4 38 0F FF FF FF 00 00
00B8	84 20 0F FF FF FF 18 E7
00C0	E7 20 0F FF FF FF 14 44
00C8	00 00 0F FF FF FF 14 47
00D0	00 00 0F FF FF FF 14 44
00D8	00 00 00 00 00 00 18 E7

Note from Editor:

Register initialization is required for expanded systems. Super Monitor owners should use the Execute option (00) at 00E0.

ADDR	CODE
00E0	93
00E1	B0
00E2	A0
00E3	B1
00E4	B2
00E5	B3
00E6	B4
00E7	B5
00E8	B6
00E9	B7
00EA	B8

ADDR	CODE
00EB	B9
00EC	BA
00ED	BB
00EE	BC
00EF	BD
00F0	BE
00F1	BF
00F2	E3
00F3	70
00F4	00

```
0000 F812 A1F8 3AA2 F83B A3F8 04BC F8A0 A5D3
0010 7270 C422 7822 529C AF85 BF91 B09F A019
0020 3025 9FA0 2FA0 8F32 2D9F A030 229F A0A0
0030 9C34 10AF 80BF 3025 8723 FFE2 69F8 0FAA
0040 F8FF A8F8 87A4 3F46 3748 89A6 F8AB ABF8
0050 BBAD F8CB AE88 5B1B 1B5B 5D1D 1D5D 5E1E
0060 1E5E 2D8A 5DD4 8A5B 2E2E 5ED4 2B2B 8A5B
0070 1E1E 5ED4 885D 5B5E D48A 5B5E D48A 1D5D
0080 2D2D 5DD4 304C D316 8632 92A7 2787 3A8C
0090 3086 F843 A3D3 0000 0000 0000 0000 0000
00A0 E438 0FFF FFFF 0000 8420 0FFF FFFF 0000
00B0 E438 0FFF FFFF 0000 8420 0FFF 0FFF 18E7
00C0 E720 0FFF FFFF 1444 0000 0FFF FFFF 1447
00D0 0000 0FFF FFFF 1444 0000 0000 0000 18E7
00E0 93B0 A0B1 B2B3 B4B5 B6B7 B8B9 BABB BCBD
00F0 BEBF E370
```

NIM

by
Richard Moffie

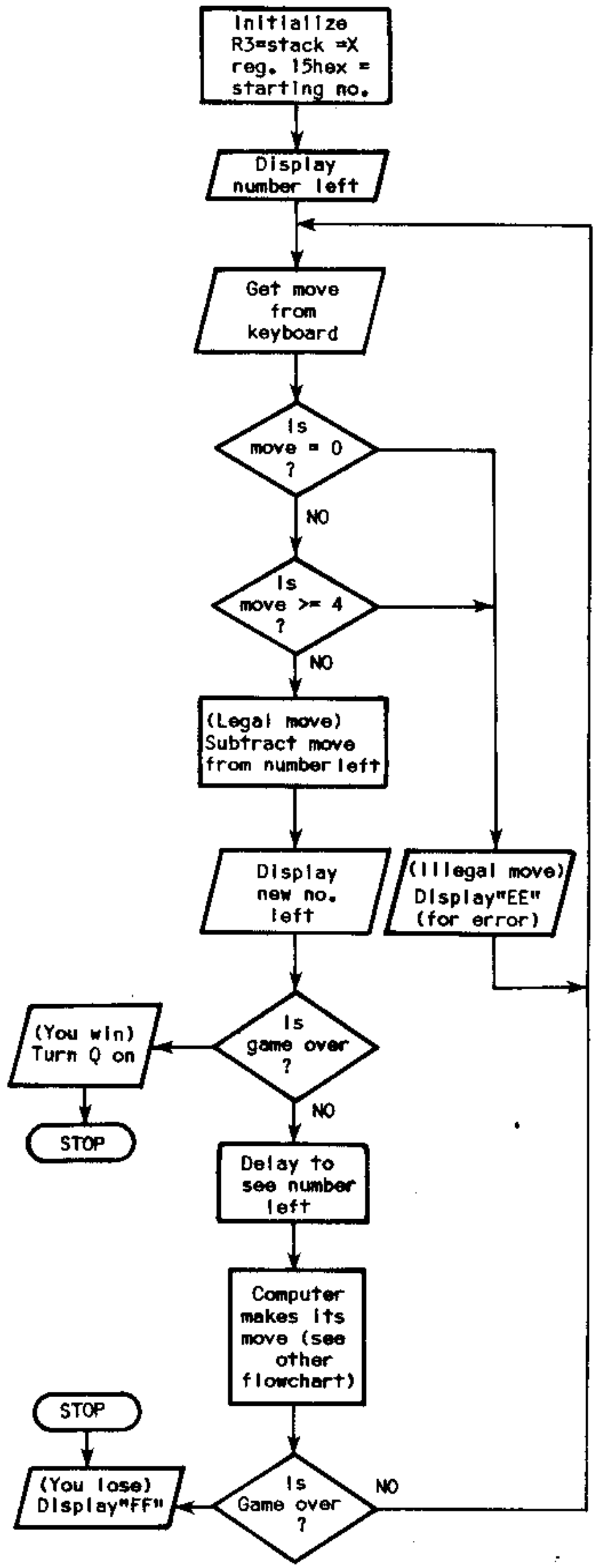
This is one version of an ancient game where play begins with a pile of objects (21 or 15 hex in this listing - it can be changed by putting the desired starting number in byte 05), and two players alternate removing 1, 2 or 3 from the pile. The object is to defeat your opponent, the computer, by removing the last object from the pile. The game is played in hex, and is one way to learn hex codes.

Play begins with the computer displaying the starting number and you get the first move. If you make an illegal move, anything besides 1, 2 or 3, the computer will display EE- for Error and loop back for you to make a valid move. When you do, it is subtracted from the pile and checked to see if you won the game (0 left). If so, the Q light is turned on and the computer stops. To play again, press Reset and Go. If you haven't won, there is a delay loop for you to see the result of your move and to give the impression the computer is trying to "think" of its next move. The computer then makes its move, and checks to see if it won. If so, FF is displayed and you lose the game. If not, the program loops back for your next move.

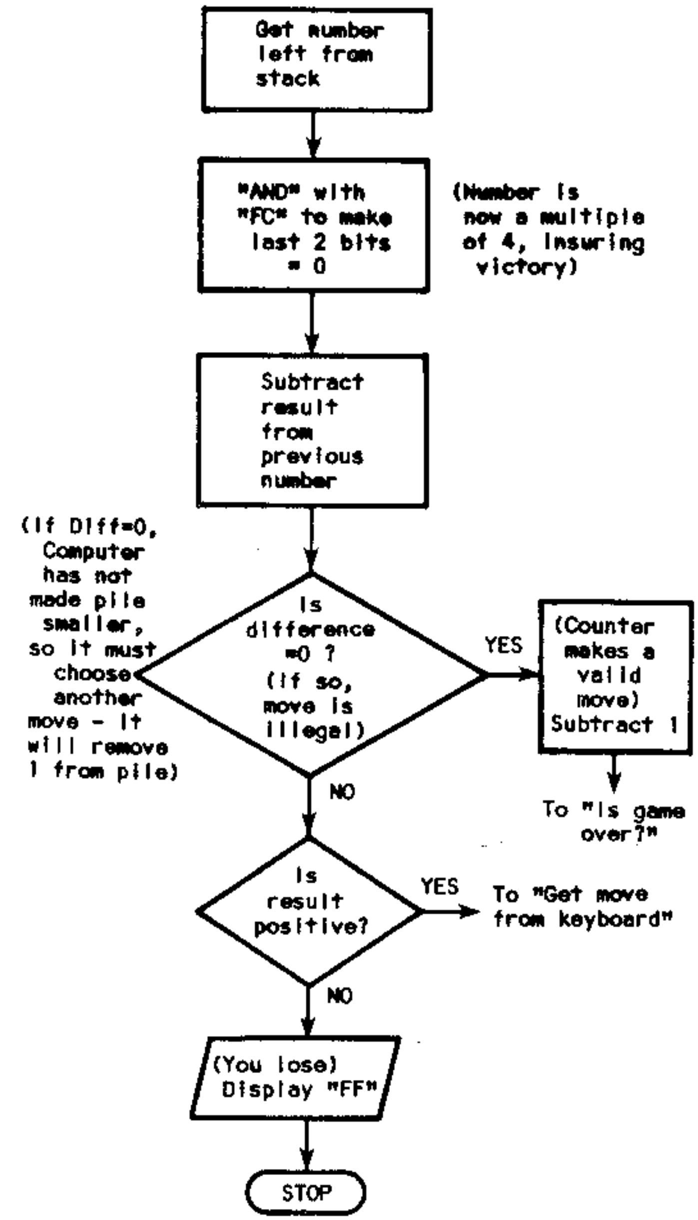
It's really best not to know how the computer makes its moves, since once you know how to win, there isn't much fun to the game, unless you wish to see friends beaten by your computer. However, if you really want to know, here are the details:

If the computer (or you) move so that after your move, there is a multiple of 4 left in the pile, you can continue reducing the pile by a multiple of 4 on each successive move until there are only 4 left in the pile. Then if your opponent removes 1, you take 3 and win. If he takes 2, you take 2 and win, and if he takes 3, you take 1 and win. The computer moves by and-ing the pile with FC hex so that the last two bits are 00 (for example if pile = 09=00001001, 00001001 and 11111100 = 00001000 =8) This will always be a multiple of 4, however if the pile was already a multiple of 4, the computer hasn't subtracted anything, so it will make a move of subtracting 1 (on the basis that the less it removes from the pile, the more turns it will have to try and win later). If there were less than 4 left, when the computer gets to move, the game is over with the computer winning, and if not the game continues.

It is really amazing to me that all this logic by the machine can be a part of a program which in its entirety needs only 66 bytes.



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




```
Registers Used:
     X=3
     P=0
     0=PC
     1=Delay Counter
     3=SP
     4.0=Computer's move
```

Nim Game Listing

```
ADDR CODE               COMMENTS
0000 30 42 A3 E3        Stack pointer
0004 F8 15              Change byte 05 for diff. number
0006 53 64              Display number
0008 3F 08 37 0A 6C     Get move
000D 32 3A
000F FF 0A 33 3A        Check for valid input
0013 F0 23 F5 53        Subtract move
0017 64 32 38
001A F8 B0 B1           Delay - change byte 1B for diff.
001D 21 91 3A 1D        delay
0021 23 F0 FA FC        Computer makes its move
0025 A4 F5 3A 35
0029 F0 FF 01
002C 53 3B 31           Check for end of game
002F 3A 06
0031 F8 FF 30 06        Display FF - you lose
0035 84 30 2C
0038 7B 00              Display 00 & turn Q on - you win
003A E0 64 EE E3        Display EE - invalid move
003E 30 08
0040 00 00              Stack area
0042 F8 00              Get 00 for high bytes of:
0044 B3                 R3 and
0045 B4                 R4
0046 F8 41              Get 41 for stack pointer
0048 30 02              Branch
```

```
0000 3042 A3E3 F815 5364 3F08 370A 6C32 3AFF
0010 0A33 3AF0 23F5 5364 3238 F8B0 B121 913A
0020 1D23 F0FA FCA4 F53A 35F0 FF01 533B 313A
0030 06F8 FF30 0684 302C 7B00 E064 EEE3 3008
0040 0000 F800 B3B4 F841 3002
```

JINGLE BELLS????

We at Questdata realize it's a little early to be thinking about Christmas, but it does seem to "pop up" before you know it. Therefore, this is a little reminder, to all you creative geniuses, that we need "holiday type" programs. We need them in the near future in order that we may review them for publication.

Many thanks,

QUESTDATA STAFF

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

Publisher ....Quest Electronics
Editor ....Paul Messinger
Assistant to Editor ...Jeanette Johnson
Associate Editor ....Allan Armstrong
Contributing Editors Ron Cenker
Van Baker
Art and Graphics ....Holly Olson
Proof Reading ....Judy Pitkin
Production ....John Larimer
Circulation ....Sue Orr

MODS FOR VIP TEXT EDITOR AND DISASSEMBLER

by
Ivan Dzombak

For only twenty dollars, you can have a text editor and disassembler running on your very own SUPER ELF. Of course, your first reaction is "But Stiv, you wiiiiiild and craaaaaaazzy programmer! How is this possible?!" Well, I'll tell you. A man named Tom Swan has written a 160 page booklet called **Pips for Vips**. This booklet contains programs written for the VIP. Among these programs are a text editor and a disassembler, both of which can be easily modified to run on the SUPER ELF (or ELF II, for that matter). No video board is necessary, because both programs contain a high-resolution character generator with full ASCII character set. A minimum of 3K of RAM is necessary, and an ASCII keyboard makes the text editor much more practical (routines for both ASCII and hex are included).

To use these great programs, they must first be handloaded (boo, hiss!); a tape is provided with the book, but this tape is in the VIP format. Next, load the character set and lookup tables (disassembler only). The format of these tables is discussed at length in the manual; the mnemonic lookup table is very flexible, in that the user can specify his own mnemonics (the table in this article conforms to RCA convention). The ASCII character set is also user definable. The manual contains excellent documentation for these programs, and it shows how and where to place the various tables. The text editor has 21 functions available to the user; they are:

KEY	FUNCTION
8	Cursor left
9	Cursor right
A	Scroll up
B	Scroll down
C	Control select
D	Carriage return
E	Cursor up
F	Cursor down

NOTE:

The booklet and tape (PIPS FOR VIPS) cost $19.95 together, or only $14.95 for the book alone. Send your order to:

ARESCO
P.O. Box 1142
Columbia, MD
21044

KEYS	FUNCTION
C/0	Escape
C/1	Page back
C/2	Page forward
C/3	Show page "N"
C/4	Cursor on/off
C/5	Reverse video (black on white, wh on bl)
C/6	Insert line
C/7	Delete line
C/8	Available for expansion
C/9	Available for expansion
C/A	Erase text buffer
C/B	Tape read (see note)
C/C	Home cursor
C/D	Erase to end of line
KEY	FUNCTION
C/E	Erase to end of page
C/F	Tape write (see note)

The disassembler has seven functions; they are:

KEY	FUNCTION
C	Page forward
D	Show from
E	"Start from" address
0	Write byte
5-6	Tape read/write (see note)
*	Add table entry

Note that the tape read/write routines will not work; this is because there are too many different monitors in use, and it would be almost impossible to include all of them here. The best way to effect cassette read/write is to reset and jump to the monitor.

These two programs are very useful (this article was composed with the text editor), and they are definitely worth the relatively small investment. Also included in the book are a few good CHIP-8 programs; note, however, that these will not run without modification to the Moews CHIP-8 interpreter (but it is possible). These other programs are:

CHIP-8 Program Editor
Character Designer
Messenger (display ASCII text in CHIP-8 programs)
Space Wars (high resolution)
Surround (high resolution game)

Text Editor and Disassembler Listing

TEXT EDITOR MODIFICATIONS

ADDR	CODE	COMMENT
0000	F8 0C	Load display page into RB.1
	BB	
0029	00 (03)	Change location of input routine (00 for ASCII kbd, 03 for hex input)
002C	51 (E1)	Change location of input routine (51 for ASCII kbd, E1 for hex input)
0030	FD 07	Change subtract instruction so it doesn't reference R(X)
	C4	NOP to use space

****** ASCII INPUT ROUTINE **********

ADDR	CODE	COMMENT
0050	D3	Return
0051	F8 00	Initialize input pointer
0053	BF	
0054	F8 7A	
0056	AF	
0057	EF	Set X to F
0058	31 6F	If Q on, branch to get second half of ASCII input from keyboard
005A	36 60	Branch if EF3 active
005C	37 6A	Branch if EF4 active
005E	30 5A	If not, do it again
ADDR	CODE	COMMENT
0060	7B	It's ASCII, so set Q
0061	6F	Get input
0062	F6 F6 F6 F6	Shift right to get high order nybble
0066	FA 0F	And with 0F
0068	30 50	Branch to return
006A	37 6A	Wait for "input" release
006C	6C	Get hex input
006D	30 66	Go and strip off high order nybble
006F	0F	Since Q was already set, just get low order nybble of ASCII byte in M(R(F))
0070	FA 0F	Strip off high order nybble
0072	7A	Reset Q
0073	30 50	Branch to return

********* HEX INPUT ROUTINE *****************

ADDR	CODE	COMMENT
03E0	D3	Return
03E1	F8 03	Initialize input pointer
00E3	BF	
00E4	F8 FF	
00E6	AF	
00E7	EF	Set X to F
00E8	3F E8	Wait for INPUT
00EA	37 EA	Wait for INPUT release
00EC	6C	Get hex input
00ED	FA 0F	Strip off high order nybble
00EF	5F	Put low order nybble into M(R(F))
00F0	30 E0	Branch to return

DISASSEMBLER MODIFICATIONS

ADDR	CODE	COMMENT
0000	F8 0C	Initialize display pointer
	BB	to top 4 pages
0036	00	New address for keyboard input routine
0039	44	New address for keyboard input routine

********** HEX INPUT ROUTINE ************

ADDR	CODE	COMMENT
0043	D3	Return
0044	E2	Set X to 2
0045	3F 45	Wait for INPUT
0047	37 47	Wait for INPUT release
0049	6C	Get hex input
004A	FA 0F	AND with 0F hex to strip off high order nybble
004C	AF	Put into RF.0
004D	30 43	Branch back to return

************ END INPUT ROUTINE *****************

ADDR	CODE	COMMENT
01EC	00	New address for keyboard input routine
01EF	44	New address for keyboard input routine
01FA	3F 3F 3F 00	ASCII data for error message "???"
024D	01	New address for error message
ADDR	CODE	COMMENT
0250	FA	New address for error message
02C7	01	New address for keyboard input routine
02CA	44	New address for keyboard input routine

MNEMONIC LOOKUP TABLE

ADDR	CODE	ADDR	CODE
0800	0F 4C 44 4E 00	0891	F5 53 44 00
	4F 4C 44 41 00		FD 53 44 49 00
	F0 4C 44 58 00		75 53 44 42 00
080F	72 4C 44 58 41 00	089F	7D 53 44 42 49 00
	F8 4C 44 49 00		F7 53 4D 00
	5F 53 54 52 00		FF 53 4D 49 00
081F	73 53 54 58 44 00		77 53 4D 42 00
	1F 49 4E 43 00	08B3	7F 53 4D 42 49 00
	2F 44 45 43 00		30 42 52 00
082F	60 49 52 58 00		32 42 5A 00
	8F 47 4C 4F 00	08C1	3A 42 4E 5A 00
	AF 50 4C 4F 00		33 42 44 46 00
	9F 47 48 49 00		3B 42 4E 46 00
0843	BF 50 48 49 00	08D0	31 42 51 00
	F1 4F 52 00		39 42 4E 51 00
	F9 4F 52 49 00		34 42 31 00
0851	F3 58 4F 52 00		3C 42 4E 31 00
	FB 58 4F 52 49 00	08E2	35 42 32 00
	F2 41 4E 44 00		3D 42 4E 00
0861	FA 41 4E 49 00		36 42 33 00
	F6 53 48 52 00		
	76 53 48 52 43 00		
0871	FE 53 48 4C 00		
	7E 53 48 4C 43 00		
	F4 41 44 44 00		
0881	FC 41 44 49 00		
	74 41 44 43 00		
	7C 41 44 43 49 00		

ADDR	CODE		
08EF	3E 42 4E 33 00	0940	C7 4C 53 4E 46 00
	37 42 34 00		CD 4C 53 51 00
	3F 42 4E 34 00		C5 4C 53 4E 51 00
	C0 4C 42 52 00	0951	CC 4C 53 49 45 00
0902	C2 4C 42 5A 00		00 48 41 4C 54 00
	CA 4C 42 4E 5A 00		C4 4E 4F 50 00
	C3 4C 42 44 46 00	0962	DF 53 45 50 00
	CB 4C 42 4E 46 00		EF 53 45 58 00
	C1 4C 42 51 00		7B 53 45 51 00
091E	C9 4C 42 4E 51 00	0971	7A 52 45 51 00
	38 53 4B 50 00		78 53 41 56 00
	C8 4C 53 4B 50 00		79 4D 41 52 4B 00
092F	CE 4C 53 5A 00	0981	70 52 45 54 00
	C6 4C 53 4E 5A 00		71 44 49 53 00
	CF 4C 53 44 46 00		61 49 4E 50 00
			6F 4F 55 54 00

ARGUMENT LOOKUP TABLE

```
ADDR CODE
09C0 F8 F9 FA FB FC FD FF 7C 7D 7F 30 31 32 33 34 35
09D0 36 37 39 3A 3B 3C 3D 3E 3F 00 C0 C1 C2 C3 CA CB
09E0 C9 00
```

ASCII CHARACTER SET (at 0A00 for 4K)

ADDR	CODE	ADDR	CODE
0A00	All 0's	0B38	57 77 50 00 N
.			75 55 70 00 O
.			75 74 40 00 P
0A80	00 00 00 00 sp		25 55 21 00 Q
	11 11 01 00 !		65 65 50 00 R
	55 00 00 00 "		74 71 70 00 S
	5F 5F 50 00 #	0B50	72 22 70 00 T
	23 63 62 00 $		55 55 70 00 U
	51 24 50 00 %		55 55 20 00 V
	24 25 70 00 &		55 77 20 00 W
0A9C	44 00 00 00 '		55 25 50 00 X
	24 44 20 00 (		55 22 20 00 Y
	42 22 40 00)	0B68	71 24 70 00 Z
0AA8	52 50 00 00 *		64 44 46 00 [
	02 72 00 00 +		44 21 10 00 \
	00 00 44 80 ,		62 22 26 00]
	00 70 00 00 -		25 22 20 00 ^
	00 00 44 00 .	cursor	00 00 00 F0
	11 24 40 00 /	0B80	42 00 00 00 ‾
0AC0	25 55 20 00 0		61 35 30 00 a
	26 22 70 00 1		44 75 70 00 b
	71 74 70 00 2		00 74 70 00 c
	71 71 70 00 3		11 75 70 00 d
	55 71 10 00 4		25 64 30 00 e
	74 71 70 00 5	0B98	25 46 40 00 f
09D8	74 75 70 00 6		00 25 31 60 g
	71 24 40 00 7		44 75 50 00 h
	75 75 70 00 8		20 22 20 00 i
	75 71 60 00 9		10 11 15 20 j
	44 04 40 00 :		44 56 50 00 k
	44 04 48 00 ;	0BB0	62 22 70 00 l
09F0	12 42 10 00 <		05 77 50 00 m
	07 07 00 00 =		00 65 50 00 n
	42 12 40 00 >		00 75 70 00 o
	25 12 20 20 ?		00 75 74 40 p
	35 25 30 00 @		00 75 71 10 q
	25 75 50 00 A	0BC8	00 74 40 00 r
0B08	65 65 60 00 B	0BCC	00 32 60 00 s
	74 44 70 00 C		22 72 30 00 t
	65 55 60 00 D		00 55 70 00 u
	74 64 70 00 E		00 55 20 00 v
	74 64 40 00 F		00 57 70 00 w
	74 45 70 00 G	ADDR	DATA
0B20	55 75 50 00 H	0BE0	00 52 50 00 x
	72 22 70 00 I		00 55 71 70 y
	11 15 20 00 J		00 72 70 00 z
	55 65 50 00 K		12 24 22 10
	44 44 70 00 L		22 20 22 20
	57 75 50 00 M		00 42 21 22
		0BF8	40 00 63 00
			00 00 4F 40 del

HEX TO DECIMAL

by
Paul J. Grech

A recent program I wrote involved converting Hexadecimal numbers to decimal. I would enter the data and convert to decimal as part of the main program. This, of course, slowed down the main program and occupied a lot of memory space.

I have thought of another way to convert Hex to Decimal and I would like to share it with you and perhaps your Questdata subscribers.

The following program will convert up to FF. It can also be modified slightly to do opposite conversions.

Registers Used:
X=2
P=0
0=Program Counter
1=Hex Number
2=Decimal Output

ADDR	DATA	COMMENT
10	90	Get R0.0.
11	B1 B2	Set up R1 & R2.
13	E2	Sex.
14	F8 0F	LD1. - 0F - Hex #.
16	A1	Put R1.0.
17	F8 30	LD1 - 25 - Decimal #.
19	A2	Put R2.0.
1A	81	Get R1.0.
1B	32 29	BZ -.
1D	21	Dec R1.
1E	02	LDN.
1F	FC 01	AD1 - 01.
21	52	STR.
22	FD 0A	SD1 - 0A.
24	3A 17	BNZ.
26	73	STRX.
27	30 1E	BR.
29	7B	SEQ.
2A	00	Stop.

```
0000 3010 0000 0000 0000 0000 0000 0000 0000
0010 90B1 B2E2 F8FF A1F8 30A2 8132 2921 02FC
0020 0152 FD0A 3A17 7330 1E7B 00
```

NOTE:
Locations 002E-0030 must be set to 00! The program receives hex input at location 0015 and outputs decimal at 002E-0030. (Use 30 10 at 0000 to run it).

ELF II CASSETTE TAPE READER

by
Van C. Baker

The routine whose hex listing appears below can be used by non-Elf II users to read cassette tapes generated by the Elf-II cassette driver. To use the routine, load the program into memory, noting that it may be located beginning on any page boundary. Also note that location XX01 (where "XX" represents the page number) must contain the byte defining the page in which the routine is located. The program listed below, for example, runs in page zero, hence, byte 0001 is "00".

The program assumes flag line EF3 is used for the cassette serial input; if your system differs from this convention, patch in the appropriate EFn conditional branch instructions at XX45,XX4C,XX5C,XX95,XX9C,XXA9,XXBA and XXBC.

To use the program, proceed as follows:

1. Execute the program using your monitor or other means. It does not matter what register is the program counter when the program is entered.

2. Note that "AA" will be displayed on the hex display. Enter the high byte of the starting address into which the tape contents are to be loaded. Press the "I" key on the hex keypad.

3. Enter the low byte of the starting address. Press the "I" key.

4. Enter the high byte of the end address. Press the "I" key.

5. Enter the low byte of the end address. Press the "I" key.

6. Enter file number to be read (01-FF). Press the "I" key.

7. Start the recorder (on playback).

As the tape advances, the current file number being skipped (if the file entered in 6 was greater than 1) will be displayed until the requested file is reached. As the contents of the tape are being loaded into the requested memory locations, the hex display will rapidly flicker. When the tape has been read, "AA" will appear on the hex display. At this point, you may load another tape by proceeding with step 2 above. Use caution not to load tape data over the cassette read program itself!

If "EE" should appear on the display while the tape is being read, it indicates that a read error (e.g., a parity error) occured. The address of the byte at which the error occured can be determined by examining locations XXFE and XXFF (High and low address bytes, respectively). To recover from a read error, press the "I" key and proceed from step 2 above.

```
     ;*********************************************
0000 ;*********************************************
0000 ;
0000 ;       ELF II CASSETTE LOAD ROUTINE
0000 ;
0000 ;              BY V C BAKER
0000 ;
0000 ;       * THIS ROUTINE READS A STANDARD "ELF II
0000 ;       * MONITOR" CASSETTE, LOADING THE DATA
0000 ;       * INTO USER-DESIGNATED MEMORY AREA.
0000 ;
0000 ;
0000 ;       * TO USE:
0000 ;       *
0000 ;       *  (1)  LOAD THE FOLLOWING ROUTINE INTO
0000 ;       *  MEMORY.  ALTHOUGH THE ROUTINE LISTED
0000 ;       *  HERE STARTS AT 0000 (HEX), IT MAY BE
0000 ;       *  LOCATED ANYWHERE AS LONG AS IT BEGINS
0000 ;       *  ON A PAGE BOUNDARY, I.E., AT XX00.
0000 ;       *  PATCH IN THE ACTUAL PAGE NUMBER AT
0000 ;       *  BYTE XX01.
0000 ;       *
0000 ;       *  (2)  EXECUTE THE PROGRAM USING ANY
0000 ;       *  REGISTER FOR THE PROGRAM COUNTER.
0000 ;
0000 ;       *  (3)  WHEN "AA" IS DISPLAYED ON THE
0000 ;       *  HEX DISPLAY, ENTER THE FOLLOWING,
0000 ;       *  USING THE HEX KEYPAD:
0000 ;       *
0000 ;       *       (A)  HIGH BYTE (MSH) OF STARTING
0000 ;       *       ADDRESS INTO WHICH TAPE CONTENTS
0000 ;       *       IS TO BE LOADED.
0000 ;       *
0000 ;       *       (B)  PRESS THE "I" KEY.
0000 ;       *
0000 ;       *       (C)  LOW BYTE (LSH) OF STARTING
0000 ;       *       ADDRESS.
0000 ;       *
0000 ;       *       (D)  PRESS THE "I" KEY.
0000 ;       *
0000 ;       *       (E)  HIGH BYTE OF END ADDRESS
0000 ;       *
0000 ;       *       (F)  PRESS THE "I" KEY.
0000 ;       *
0000 ;       *       (G)  LOW BYTE OF END ADDRESS
0000 ;       *
0000 ;       *       (J)  PRESS THE "I" KEY.
0000 ;       *
0000 ;       *       (K)  FILE NUMBER
0000 ;       *
0000 ;       *       (L)  PRESS THE "I" KEY.
0000 ;       *
```

```
0000 ;          *  (4)  START THE RECORDER ON PLAYBACK.
0000 ;          *
0000 ;          *  (5)  AS THE TAPE IS READ, ITS CONTENTS
0000 ;          *  WILL BE LOADED INTO THE DESIGNATED
0000 ;          *  MEMORY LOCATIONS.
0000 ;          *  NOTE THAT CAUTION MUST BE USED
0000 ;          *  TO ASSURE THAT THE CASSETTE ROUTINE
0000 ;          *  ITSELF IS NOT WRITTEN OVER AS THE
0000 ;          *  TAPE IS LOADED!
0000 ;          *        IF FILE 2 OR GREATER WAS
0000 ;          *  IN STEP 3-K, THE CURRENT FILE NUMBER
0000 ;          *  BEING SKIPPED WILL BE DISPLAYED UNTIL
0000 ;          *  THE REQUESTED FILE IS REACHED.  THE
0000 ;          *  DISPLAY WILL FLICKER AS THE TAPE IS
0000 ;          *  BEING READ.
0000 ;          *
0000 ;          *  (6)  WHEN THE TAPE HAS BEEN SUCCESS-
0000 ;          *  FULLY READ, "AA" WILL APPEAR ON THE
0000 ;          *  HEX DISPLAY.  IF "EE" APPEARS, A READ
0000 ;          *  ERROR OCCURED (E.G., A PARITY ERROR).
0000 ;          *  TO RECOVER FROM THE ERROR, REWIND THE
0000 ;          *  TAPE, PRESS THE "1" KEY, AND START
0000 ;          *  OVER FROM STEP 3.
0000 ;          *        THE ADDRESS OF THE BYTE BEING
0000 ;          *  READ WHEN THE ERROR OCCURRED CAN BE
0000 ;          *  DETERMINED BY INSPECTING MEMORY
0000 ;          *  LOCATIONS XXFE (FOR THE HIGH BYTE OF
0000 ;          *  ADDRESS) AND XXFF (FOR THE LOW BYTE).
0000 ;
0000 ;
```

```
ADDR CODE     LABEL    OPCODE OPERAND COMMENT
0000                   ORG    #0000   ; Start of program
0000          START:   EQL    #0000
0000          ONECT:   EQL    #000D   ; One-bit timing
0000                                  ; value
0000          LDRCT:   EQL    #000A   ; Leader one's
0000                                  ; timing
0000          ;
0000          ;
0000 F8 00    GO:      LDI    START   ; Load page number
0002 B3                PHI    R3      ; for this routine
0003 F8 07             LDI    INIT    ; Initialization
0003                                  ; address
0005 A3                PLO    R3
0006 D3                SEP    R3      ; SEP to PC = R3
0007 93       INIT:    GHI    R3      ; Initialize
0007                                  ; registers.
0008 BB                PHI    RB      ; RB = BOTM2 PC.
0009 B8                PHI    R8      ; R8 = DI PC.
000A B6                PHI    R6      ; R6 = ERR PC.
000B B5                PHI    R5      ; R5 = HEXIN PC.
000C B2                PHI    R2      ; R2 = STACK
000C                                  ; POINTER
000D          ;
000D          ;
000D          ;
000D F8 71             LDI    BOTM2   ; BOTM2 address
000F AB                PLO    RB
0010 F8 91             LDI    DI      ; DI address
0012 A8                PLO    R8
0013 F8 81             LDI    ERR     ; ERR address
0015 A6                PLO    R6
0016 F8 FF             LDI    #FF     ; Stack at XXFF
0018 A2                PLO    R2
0019 E2                SEX    R2      ; SP = R2
001A F8 78             LDI    HEXIN   ; HEXIN address
001C A5                PLO    R5
001D          ;
001D          ;        * BEGIN MAIN PROGRAM
001D          ;
001D          ;
001D          ;        * DISPLAY "AA" ON LED'S
001D          ;
```

```
ADDR CODE     LABEL    OPCODE OPERAND COMMENT
001D F8 AA    BEGIN:   LDI    #AA     ; Load "AA"
001F 52                STR    R2      ; Store it
0020 64                OUT    4       ; Output it.
0021 22                DEC    R2      ; Reposition SP.
0022          ;
0022          ;        * GET STARTING ADDRESS FROM
0022          ;        * KEYPAD
0022          ;
0022 D5                SEP    R5      ; Get MSH of
0022                                  ; address
0023 BE                PHI    RE
0024 D5                SEP    R5      ; Get LSH
0025 AE                PLO    RE      ; Start address in
0025                                  ; RE.
0026          ;
0026          ;        * GET ENDING ADDRESS FROM KEYPAD
0026          ;
0026 D5                SEP    R5      ; Get MSH of end
0026                                  ; address.
0027 73                STXD           ; Save it.
0028 D5                SEP R5         ; Get LSH of end
0028                                  ; address.
0029 52                STR    R2      ; Save it, too.
002A          ;        * CALCULATE NUMBER OF BYTES
002A 8E                GLO    RE
002B F5                SD
002C A6                PLO    R6      ; Low half of
002C                                  ; number of bytes
002D 9E                GHI    RE      ;
002E 60                IRX
002F 75                SDB            ; High half of
002F                                  ; number of bytes
0030 B6                PHI    R6
0031 3B 81             BM     ERR     ; Error if
0031                                  ; negative address
0033          ;
0033          ;        * GET FILE NUMBER FROM KEYPAD
0033          ;
0033 D5                SEP    R5      ; Get file number
0033                                  ; (1-FF).
0034 A4                PLO    R4      ; Save in R4.
0035          ;
0035          ;        * SET TIMING VALUE FOR A ONE BIT
0035          ;
0035 F8 0D             LDI    ONECT
0037 B9                PHI    R9      ; "1" timing value
0037                                  ; in R9.1
0038          ;
0038          ;
0038 F8 00             LDI    #00     ; Initialize
0038                                  ; current file no.
003A          ;
003A          ;
003A 52       FTST:    STR    R2      ; Save current
003A                                  ; file number
003B 64                OUT    4       ; Display it.
003C 22                DEC    R2
003D 84                GLO    R4      ; Get requested
003D                                  ; file number
003E F3                XOR            ; Check if equal
003E                                  ; to current.
003F 32 5C             BZ     RDF     ; Branch if so.
0041          ;
0041          ;        * CHECK FOR LEADER
0041          ;
0041 F8 0A    LDFND:   LDI    LDRCT   ; Load test value
0041                                  ; for leader.
0043 B7                PHI    R7      ; Save in R7.1
0044 DB       OZTST:   SEP    RB      ; Check for abort
0044                                  ; (BOTM2)
0045 36 44             B3     OZTST   ; Find data pulst
0045                                  ; transition
```

```
ADDR CODE    LABEL    OPCODE OPERAND COMMENT
0047 99               GHI    R9      ; Test pulse
0047                                 ; width
0048 FF 01   PTIM:    SMI    #01     ; Is it too long
0048                                 ; for a one?
004A 3B 54            BNF    ZER     ; JMP if so.
004C 3E 48            BN3    PTIM    ; Else loop till
004C                                 ; pulse is over.
004E 97               GHI    R7      ; This bit is a
004E                                 ; "1". Count the
004F 32 44            BZ     OZTST   ; Ones till LDRCT
004F                                 ; of them are
0051         ;                       ; found. (Verify
0051                                 ; leader located.)
0051 27               DEC    R7
0052 30 44            BR     OZTST
0054 97      ZER:     GHI    R7      ; The received bit
0054                                 ; must be a "0".
0055 3A 41            BNZ    LDFND   ; is leader done
0055                                 ; yet?
0057 02               LDN    R2      ; Yes. Increment
0057                                 ; file counter and
0058 FC 01            ADI    #01     ; then go check
0058                                 ; file number to
0058                                 ; see
005A 30 3A            BR     FTST    ; it is the one
005A                                 ; requested.
005C         ;
005C         ;        * REQUESTED FILE FOUND
005C         ;
005C 3E 5C   RDF:     BN3    RDF     ; Wait for end of
005C                                 ; leader bit.
005E         ;
005E         ;
005E         ;        * SET UP FOR DATA READ
005E         ;
005E 26      CNTSET:  DEC    R6      ; Set up byte
005E                                 ; counter
005F 96               GHI    R6      ; so that when
005F                                 ; R6.1 = 0,
0060 FC 01            ADI    #01     ; all bytes have
0060                                 ; been read.
0062 B6               PHI    R6
0063         ;
0063 D8      RDOPT:   SEP    R8      ; read a data byte
0064 5E               STR    RE      ; store it in
0064                                 ; memory
0065 1E               INC    RE      ; increment memory
0065                                 ; address
0066 8E               GLO    RE
0067 52               STR    R2
0068 64               OUT    4       ; Display low-
0068                                 ; order address.
0069 22               DEC    R2      ; Reposition SP
006A         ;
006A         ;        * CHECK IF FINISHED
006A         ;
006A 26               DEC    R6      ; Decrement byte
006A                                 ; counter
006B 96               GHI    R6
006C 3A 63            BNZ    RDOPT   ; Loop back if not
006C                                 ; done
006E 30 1D            BR     BEGIN   ; Else, go display
006E                                 ; "AA".
0070         ;
0070         ;
0070         ;
0070         ;****************************************
0070         ;****************************************
0070         ;
0070         ;****** BOTM2 ROUTINE -- ABORT THE READ
0070         ;
0070 D3      BTMRET:  SEP    R3      ; Return
```

```
ADDR CODE    LABEL    OPCODE OPERAND COMMENT
0071 3F 70   BOTM2:   BN4    BTMRET  ; Return if I-key
0071                                 ; is not in
0073 37 73   WAIT:    B4     WAIT    ; Software "De-
0073                                 ; Bounce" step.
0075 30 00            BR     GO      ; Abort. Start
0075                                 ; all over
0077         ;
0077         ;****************************************
0077         ;****************************************
0077         ;
0077         ;****** HEXIN ROUTINE -- HEX KEYPAD INPUT
0077         ;
0077 D3      HEXRET:  SEP    R3      ; Return
0078 3F 78   HEXIN:   BN4    HEXIN   ; wait for I-key
0078                                 ; in.
007A 6C               INP    4       ; Get input byte
007B 64               OUT    4       ; display it
ADDR CODE    LABEL    OPCODE OPERAND COMMENT
007C 37 7C   WAIT2:   B4     WAIT2   ; Wait till key
007C                                 ; released
007E 22               DEC    R2
007F 30 77            BR     HEXRET  ; return
0081         ;****************************************
0081         ;****************************************
0081         ;
0081         ;****** ERR ROUTINE-- ERROR "PROCESSOR"
0081         ;
0081 F8 EE   ERR:     LDI    #EE     ; Load "EE"
0083 52               STR    R2
0084 64               OUT    4       ; Display "EE"
0085 22               DEC    R2
0086 8E               GLO    RE      ; Get low address
0086                                 ; byte
0087 73               STXD
0088 9E               GHI    RE      ; Get high address
0088                                 ; byte
0089 52               STR    R2      ; Save error
0089                                 ; address on stack
008A 3F 8A   LOOP:    BN4    LOOP    ; Loop "Till I-key
008A                                 ; is pressed.
008C 37 8C   LOOP2:   B4     LOOP2
008E 3000             BR     GO      ; Start all over
008E                                 ; again
0090         ;****************************************
0090         ;****************************************
0090         ;******DI ROUTINE-- READ A BYTE FROM TAPE
0090         ;
0090 DB      DIRECT:  SEP    RB      ; Test for abort
0090                                 ; and return
0091 F8 08   DI:      LDI    #08     ; Set up counters
0093 A7               PLO    R7
0094 A9               PLO    R9
0095 36 95   LHTRAN:  B3     LHTRAN  ; Wait for end of
0095                                 ; current
0097         ;                       ; High input level
0097 99               GHI    R9      ; Test for bit
0097                                 ; value.
0098 FF 01   BTST:    SMI    #01
009A 3B A1            BNF    AHZER
009C 3E 98            BN3    BTST
009E         ;
009E 27               DEC    R7
009F 30 AB            BR     DNCHK   ; Got a "1" bit.
00A1 F8 00   AHZER:   LDI    #00     ; Got a zero bit.
00A1                                 ; Test
00A3 FC 01   TLTST:   ADI    #01     ; For excessive
00A3                                 ; pulse width.
00A5 3B A9            BNF    HLTRAN
00A7 30 81            BR     ERR     ; Error
00A9 3E A3   HLTRAN:  BN3    TLTST
00AB         ;
00AB 89      DNCHK:   GLO    R9      ; Got a byte?
00AC 32 B4            BZ     PARCHK  ; Check parity
00AC                                 ; if so.
```

```
ADDR CODE     LABEL     OPCODE OPERAND COMMENT
00AE          ;
00AE 02                 LDN    R2      ; Not full byte
00AE                                   ; yet.
00AF 7E                 RSHL           ; Insert bit into
00AF                                   ; buffer
00B0 52                 STR    R2
00B1 29                 DEC    R9
00B2 30 95              BR     LHTRAN
00B4          ;
00B4 87       PARCHK:   GLO    R7      ; Check for odd
00B4                                   ; parity.
00B5 F6                 SHR
00B6 02                 LDN    R2      ; Put data value
00B6                                   ; into D-Reg.
00B7 CF                 LSDF           ; Okay if parity
00B7                                   ; odd (Netronics)
00B8 3081               BR     ERR     ; Call error
00B8                                   ; routine
00BA 36 BA    FNLBIT:   B3     FNLBIT  ; Wait for next
00BA                                   ; Netronics
00BC 3E BC    FBIT:     BN3    FBIT    ; Start bit
00BC                                   ; transition.
00BE 30 90              BR     DIRET   ; Now return.
00C0
```

Note from the Editor:

The preceeding assembly listing is the output of the Quest Editor Assembler.

```
0000 F800 B3F8 07A3 D393 BBB8 B6B5 B2F8 71AB
0010 F891 A8F8 81A6 F8FF A2E2 F878 A5F8 AA52
0020 6422 D5BE D5AE D573 D552 8EF5 A69E 6075
0030 B63B 81D5 A4F8 0DB9 F800 5264 2284 F332
0040 5CF8 0AB7 DB36 4499 FF01 3B54 3E48 9732
0050 4427 3044 973A 4102 FC01 303A 3E5C 2696
0060 FC01 B6D8 5E1E 8E52 6422 2696 3A63 301D
0070 D33F 7037 7330 00D3 3F78 6C64 377C 2230
0080 77F8 EE52 6422 8E73 9E52 3F8A 378C 3000
0090 DBF8 08A7 A936 9599 FF01 3BA1 3E98 2730
00A0 ABF8 00FC 013B A930 813E A389 32B4 027E
00B0 5229 3095 87F6 02CF 3081 36BA 3EBC 3090
```

ALBERT BIEHL SONATINA

by
Ian Beer

This program uses the music algorithm printed in issue #13 of Questdata. The piece I chose was a sonatina by Albert Biehl, a piece I am very attached to and play often on the piano. The translation in the computer is very good. This program will run in an unexpanded Elf memory. The music can be heard by tuning an AM radio in between 13 and 14 kHz and placing it next to the Q light.

```
ADDR CODE
005C 05 02    008A 57 24    00B8 49 2D    00E6 24 2D
005E 31 1F    008C 52 27    00BA 24 2D    00E8 26 27
0060 26 27    008E 52 27    00BC 26 27    00EA 49 2D
0062 31 1F    0090 49 2D    00BE 24 2D    00EC 24 2D
0064 26 27    0092 49 2D    00C0 26 27    00EE 26 27
0066 41 33    0094 41 33    00C2 57 24    00F0 57 24
0068 41 33    0096 41 33    00C4 49 2D    00F2 49 2D
006A 41 33    0098 3E 37    00C6 52 27    00F4 52 27
006C 41 33    009A 3E 37    00C8 20 33    00F6 41 33
006E 6C 2D    009C 37 3F    00CA 62 2D    00F8 93 1F
0070 26 27    009E 37 3F    00CC 15 47    00FA 00 00
0072 57 24    00A0 53 37    00CE 18 3F
0074 49 2D    00A2 15 47    00D0 15 47
0076 3E 37    00A4 18 3F    00D2 18 3F
0078 3E 37    00A6 15 47    00D4 1B 37
007A 3E 37    00A8 18 3F    00D6 20 33
007C 3E 37    00AA 3E 37    00D8 1B 37
007E 61 33    00AC 3E 37    00DA 20 33
0080 24 2D    00AE 1B 37    00DC 3E 37
0082 52 27    00B0 20 33    00DE 1B 37
0084 26 27    00B2 1B 37    00E0 20 33
0086 31 1F    00B4 20 33    00E2 24 2D
0088 57 24    00B6 49 2D    00E4 26 27
```

TINY BASIC MUSIC

by
Richard Warner

Many of Questdata's readers are interested in playing music on their computers and enjoy using TINY BASIC language. Here is a program that will give them the best of both. With a slight change in register assignments of the Moews Music Algorithm published in Questdata Vol. 1 Issue 10 page 6, one can play music with Quest Tiny Basic.

I used R(F).0 in place of R(7).0 and R(1) in place of R(C). After these changes I assigned the starting address of the Music Algorithm at 0F00 hex (3840 decimal). Then using the Tiny Basic USR Function, Tiny Basic will play the music piece selected.

The first argument of the USR function is the starting address of the machine language program. The second argument is stored in R(8) and the third argument is stored in R(A). I used Moews "Mystery Tune" as an example, starting at 0F26 hex. The USR function arguments have been assigned as follows.

1st argument; Starting address of algorithm: 3840 decimal = 0F00 hex

2nd argument; Number of notes: 48 decimal = 30 hex

3rd argument; Starting Address of notes: 3878 decimal = 0F26 hex

A simple tiny basic program that will play the tune once is as follows:

```
10   LET P=USR(3840,48,3878)
20   END
```

Other programs can be written to repeat the tune. Other tunes can be added by changing the second argument to the number of notes and the third argument for its starting address.

```
Registers Used:
     P=3
     X=2
     1=Duration counter
     2=Stack Pointer
     3=Program Counter
     4=Call
     5=Return
     6=Linkage
     8.0=Number of Notes
     E=Delay counter
     F.0=Pitch

ADDR CODE      LABEL    OPCODE OPERAND COMMENT
0F00 EA                 SEX A
0F01 F8 01 BE  LOOP5:   LDI 01  PHI E
0F04 2E 9E     LOOP1:   DEC E   GHI E
0F06 3A 04              BNZ     LOOP1
0F08 F0 AF              LDX     PLO F
0F0A 64 28              OUT 4   DEC 8
0F0C 72 B1              LDX A   PHI 1
0F0E 72 A1              LDX A   PLO 1
0F10 8F        LOOP4:   GLO F
0F11 32 18              BZ      LOOP6
0F13 7B                 SEQ
0F14 FF 01     LOOP2:   SMI 01
0F16 3A 14              BNZ     LOOP2
0F18 7A        LOOP6    REQ
0F19 8F                 GLO F
0F1A FF 01     LOOP3:   SMI 01
0F1C 3A 1A              BNZ     LOOP3
0F1E 21 91              DEC 1   GHI 1
0F20 3A 10              BNZ     LOOP4
0F22 88                 GLO 8
0F23 3A 01              BNZ     LOOP5
0F25 D5                 SEP 5

0F00 EAF8 01BE 2E9E 3A04 F0AF 6428 72B1 72A1
0F10 8F32 187B FF01 3A14 7A8F FF01 3A1A 2191
0F20 3A10 883A 01D5
```


15
QUESTDATA
P.O. Box 4430
Santa Clara, CA 95054

ADDRESS CORRECTION REQUESTED

BULK RATE
U.S. Postage Paid
QUEST
Electronics
Permit No. 549
Santa Clara, CA

VEN . ..
.ET
.ER, MI 48063