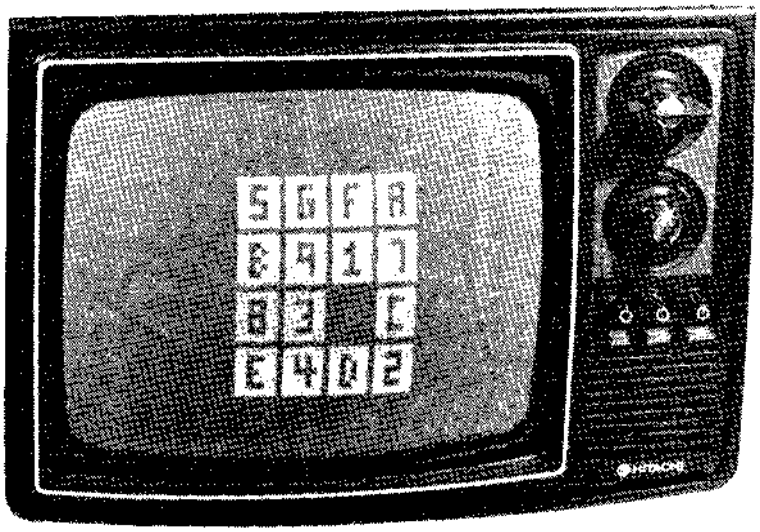
Questdata

Volume 2 Issue #1 ©

THE 15 PUZZLE

By Ray Tully



Probably everyone has at one time or another played the 15 Puzzle. In its standard form, it is a plastic or wooden box containing 15 numbered pieces and an empty space. The puzzle is solved by shifting the pieces around until they are in the proper numerical order. A hex version is as follows:

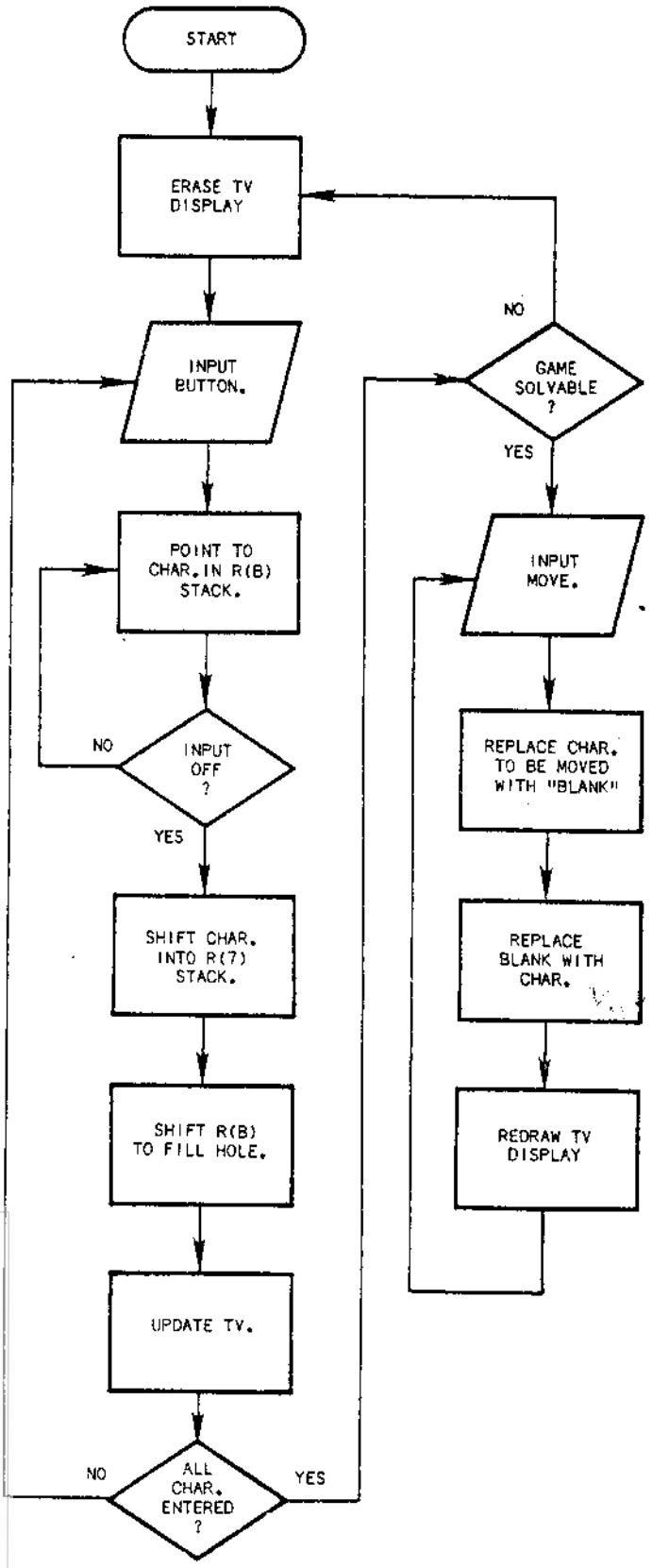
The program presented here is a video version of the 15 Puzzle, in which the pieces are moved by means of the keyboard. The program requires 1K of memory, with the program at pages 00-02, and the display on page 03.

The program begins by randomly generating a puzzle display. There are 16! = 2.092x10(13) possible board arrangements; however, only half of these are solvable. In "Game Playing with Basic" by D.D. Spencer (Hayden Books) a method is described to determine whether a given pattern is solvable or not, and is the method used in this program. First, moving from left to right and top to bottom, find how many numbers following a given square are smaller than the number of that square (count the blank as 16=hex 10). Keep this as a running total for all 16 positions. Next, referring to the above figure, if the blank occupies one of the shaded squares, add 1 to the total. The puzzle is solvable if the final sum is even (i.e., if shifting the sum right enters a 0 into the DF bit). If the computer finds that the pattern is unsolvable, then the screen is erased and the pattern generation is repeated. This continues until a solvable pattern is generated.

Run the program starting at 0100. The TV screen will be erased and the Input button monitored. While the Input is held down, the computer will rapidly cycle through the stack of pieces (pointed to by R (B)). When Input is released, the piece which is currently being pointed to will be deleted from its stack, the stack will be shifted down to fill the hole, and the piece will be entered into the display stack (pointer=R(7)). A beep will be sounded via the Q line, and the new contents of R(7) stack will be used to update the display. The piece will thus appear on the screen, drawn in the appropriate addresses as initialized by the addresses in the R(6) stack. Continued pushes of Input randomly shift the remaining characters into the display stack and TV display, until all the characters have been selected. The display stack is now tested to see if the order of the pieces is such that the puzzle is solvable. If it is not, a long beep is sounded and Q turns on. Pushing Input now resets the program and erases the screen. If the puzzle is solvable (no beep, no Q), then a given piece can be moved by pushing the corresponding hex key, followed by Input. The only legal moves, of course, are those where the piece to be moved is adjacent to the blank, and can be "mechanically slid" into place. The program does not check the legality of moves, however, so cheating is possible (when clear thinking fails). The puzzle is solved when the pieces are in their proper ascending numerical order.

A good addition to this game would be a routine to test if the puzzle has been solved. You could invent a subroutine to check for the ascending sequence 01,02,03...10 in the Square Contents Stack, locations 008B to 009A. The subroutine would be called at location 01E4, after making appropriate adjustments to the code there. If so, then perhaps a special message can be generated, or a sound made. Another possibility is to add a routine to test the legality of moves. The Piece Move Subroutine, locations 0249 to 0271, could be expanded. Within the subroutine, the location of the selected piece in the Square Control Stack (locations 008B to 009A) is compared to the location of the blank. If the locations are "adjacent" then the move is legal and the piece is moved. If the locations are not "adjacent" then just return without moving the piece (perhaps a long buzz, too?). Of course defining what is meant by "adjacent" is the real trick.

You must define the edges of the puzzle (no fair allowing sneak paths from one edge to the other), and define horizontal and vertical adjacency. Good Luck!



PROGRAM LISTING FOR 15 GAME

ADDR	CODE	LABEL	OPCODE	OPERAND	COMMENT
0100	E3				Entry from "Super Monitor"
0101	7023				
0103	93B1B2B4				R(3)=Main Program Counter
0107	F8FEA2				R(2)=Stack Pointer
010A	F816A1				R(1)=Interrupt Routine counter
010D	F800B7BB				To enter from LBR
0111	61				at Loc 0000,
0112	30CA				change "93" at Loc 0103 and LBR to 0103

VIDEO DISPLAY ROUTINE

ADDR	CODE	LABEL	OPCODE	OPERAND	COMMENT
0114	7270				P=3, Restore D, X, P
0116	22782252				R(2)-1, Save X and P,R(2)-1
011A	C4C4C4				Save D. 9 cycle NOP
011D	F803B0				R(0)=Display Area
0120	F800A0				
0123	80E2				
0125	E220A0				
0128	E220A0				
012B	E220A0				
012E	3C23				Go to refresh
0130	3014				Go to return

GENERATE RANDOM PATTERN

ADDR	CODE	LABEL	OPCODE	OPERAND	COMMENT
0133	C000AB				Go clear screen & set registers
0136	8D5C				Store R(D.0) at M(R(C))
0138	3F38				Push Input
013A	EC				X = C
013B	373F				Input off? If not keep looping
013D	304B				If yes, exit loop to draw square
013F	8BF3				D XOR M(R(C)) for equality check
0141	3246				Go reset R(B) If equal
0143	1B				R(B) + 1
0144	303B				Continue loop
0146	F89BAB				Reset R(B.0)
0149	303B				Continue loop

DISPLAY SQUARE

ADDR	CODE	LABEL	OPCODE	OPERAND	COMMENT
014B	0B5F				Enter character into Square Contents Stack
014D	D8DE				Go redraw dlspray, beep
014F	2D8D				Decrement counter R(D), place into D,
0151	FB9A				XOR with 9A for equality check
0153	3269				If equal, display is finished, exit loop

ADDR	CODE	LABEL	OPCODE	OPERAND	COMMENT
0155	9BB98BA9				Store R(B) in R(9)
0159	191F				Increment R(9) and R(F). R(B) and R(9) now point to 2 consecutive bytes
015B	095B				Shift M(R(9)) down to M(R(B))
015D	191B				Increment pointers
015F	89FBAB				XOR R(9).0 and AB for equality check
0162	3A5B				If not equal, keep looping
0164	F89BAB				Reset R(B) to 009B
0167	3036				Go select another random character

TEST IF PATTERN IS SOLVABLE

ADDR	CODE	LABEL	OPCODE	OPERAND	COMMENT
0169	F88BA7				Reset R(7) to 008B
016C	F88CAD				R(D) = 008C
016F	97BDA6				R(7) and R(D) now point to 2 consecutive characters. R(6) = counter
0172	F800F6				Shift 0 into DF
0175	E7				X = 7
0176	0DF5				Subtract M(R(X)) from M(R(D))
0178	337B				M(R(X))<M(R(D))?
017A	16				If not, R(6)+1
017B	1D				R(D)+1
017C	8DFB9B				R(D).0 XOR 9B for equality check
017F	3A72				If not equal, continue loop
0181	17				If equal, R(7)+1
0182	87AD1D				Store R(7).0 at R(D).0, increment. R(7) and R(D) point to 2 consecutive bytes
0185	FB9A				D(=R(7).0) XOR 9A
0187	3A72				If not equal, keep looping

TEST FOR BLANK (10) IN "SHADED BOX"

ADDR	CODE	LABEL	OPCODE	OPERAND	COMMENT
0189	F88CA7				Reset R(7) to 008C
018C	72FB10				M(R(X)) XOR 10, R(X)+1
018F	32BA				If equal, blank is in shaded square, go add 01 to total
0191	17				point R(7) to next square
0192	72FB1032BA				Repeat above testing process
0197	72FB1032BA				
019C	17				
019D	72FB1032BA				
01A2	1717				
01A4	72FB1032BA				
01A9	17				
01AA	72FB1032BA				
01AF	72FB1032BA				
01B4	17				
01B5	72FB103ABB				
01BA	16				Increment total
01BB	86F6				Shift total LSB into DF for odd/ even test
01BD	3BDE				If even (DF=0), go play game
01BF	DEDEDEDEDEDE				If odd, long buzz

RESET GAME IF PATTERN NOT SOLVABLE

ADDR	CODE	LABEL	OPCODE	OPERAND	COMMENT
01C5	7B				Q on-unsolvable game
01C6	3FC6				Push input to reset game
01C8	37C8				
01CA	F89BABA727				Reset square contents
01CF	F810				Stack and character stack
01D1	ACBC				
01D3	5B				
01D4	9C				
01D5	57				
01D6	27				
01D7	1B				
01D8	2C				
01D9	8C				
01DA	3AD3				
01DC	3033				

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

MOVE PIECES TO PLAY GAME

ADDR	CODE	LABEL	OPCODE	OPERAND	COMMENT
01DE	F802B6F84AA6				R(6) points to subroutine
01E4	3FE4				Enter move, push Input
01E6	D6				Go move piece within stack
01E7	F88BA7				Reset R(7) for use by sub-routine
01EA	D8DE				Go redraw display, beep
01EC	37EC30DE				Input off? Then return

DISPLAY DRAW SUBROUTINE

ADDR	CODE	LABEL	OPCODE	OPERAND	COMMENT
0200	D3				Return to main program
0201	F800B6B9				R(6)=address table pointer
0205	F8D6A6				R(9)=dot table pointer
0208	F803A9B5				R(5)=display pointer
020C	06A5				Store address pointer in R(5).0
020E	07FF01AA				Decrement M(R(7)) by 01, store in R(A).0
0212	321B				R(A).0=0? Exit loop
0214	89FC08A9				Increment R(9).0 by 08 to point to dot table of next character
0218	8A				Put R(A).0 back into D for decrementing
0219	300F				Continue loop
021B	0955				Copy dots into display to draw character
021D	85FC08A5				Move display pointer down one line
0221	19				Increment dot table pointer
0222	09FBFF3A1B				Dot table data= FF? Then finished
0227	1617				Move pointers to next address & data
0229	86FBE6				R(6).0 XOR E6
022C	3A08				If not equal, go finish drawing display
022E	F88BA7				Reset R(7) to 008B
0231	3000				Return to Main Program

BEEP SUBROUTINE

ADDR	CODE	LABEL	OPCODE	OPERAND	COMMENT
0233	D3				Return to Main Program
0234	F806A6				Set beep length
0237	7A				Q off
0238	F848A9				Set beep frequency
023B	29				R(9)-1
023C	893A3B				R(9).0 = 0? If not, continue decrement-ing
023F	3137				Q on?
0241	7B26				If not, turn Q on, decrement R(6)
0243	863A38				R(6).0 = 0? If not, continue looping
0246	7A3033				If yes, turn Q off, return

PIECE MOVE SUBROUTINE

ADDR	CODE	LABEL	OPCODE	OPERAND	COMMENT
0249	D3				Return to Main Program
024A	F88BA7				Reset R(7) to 008B
024D	72FB10				M(R(X)) XOR 10 for equality check, R(X)+1
0250	3A4D				Continue until equal
0252	27				R(7)-1, R(7) now points to "blank" (10)
0253	EC				X = C
0254	6C				Input bus data, store at M(R(C))
0255	FA0F				Erase 4 MSB's
0257	5C				Display input on
0258	642C				hex display
025A	3254				Loop if input is zero

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

```
ADDR CODE     LABEL   OPCODE  OPERAND   COMMENT

025C 3060
0260 F88BAB                             Reset R(B) to
                                        008B
0263 EB                                 X = B
0264 0CF31B                             M(R(X)) XOR
                                        M(R(C)), R(B)+1
0267 3A63                               Continue loop until
                                        M(R(X))=M(R(C))
0269 2B                                 R(B)-1. R(B) points
                                        to piece being
                                        moved
026A 0C57                               Copy M(R(C))=moved
                                        piece into M(R(7))
                                        =blank
026C F8105B                             Replace moved piece
                                        with "blank"
026F E7                                 Reset X to 7
0270 3049                               Return to main
                                        Program
```

CHARACTER DOT TABLE

0083 to 008A not used

008B to 009A Square Contents Stack

009B to 00AA Character Stack (program randomly shifts into above Contents Stack)

ERASE SCREEN AND SET REGISTERS

```
ADDR CODE     LABEL   OPCODE  OPERAND   COMMENT

00AB F803B5                             R(5) points to
                                        display area
00AE F800A5
00B1 B7BBBCBF                           Set miscellaneous
                                        registers
00B5 9755                               Load 00 into
                                        display
00B7 15                                 Point to next
                                        display byte
00B8 853AB5                             Erase until next
                                        page is reached
00BB F802B8BE
00BF F801A8                             R(8) points to
                                        Display Draw
                                        subroutine
00C2 F88BA7AF                           R(7) and R(F)
                                        point to Square
                                        Contents Stack
00C6 F89BAB                             R(B) points to
                                        Character Dot
                                        Table
00C9 F8FFAC                             M(R(C)) is for
                                        temporary data
                                        storage
00CC F834AE                             R(E) points to
                                        beep subroutine
00CF F8AAAD                             R(D) points to
                                        top of R(7) stack
00D2 7A                                 Turn Q off (used
                                        during program
                                        reset)
00D3 C00136                             Return to Main
                                        Program
```

TABLE OF SQUARE ADDRESSES

```
ADDR CODE     LABEL   OPCODE  OPERAND   COMMENT

00D6 0203040542434445                   These addresses
                                        point to the tops
00DE 82838485C2C3C4C5                   of the 16 squares
                                        of the display
```

```
0000 FFFF FF7F 7767 7777 637F FF7F 637B 636F
0010 637F FF7F 637B 737B 637F FF7F 6B6B 617B
0020 7B7F FF7F 636F 637B 637F FF7F 636F 636B
0030 637F FF7F 637B 7B7B 7B7F FF7F 636B 636B
0040 637F FF7F 636B 637B 7B7F FF7F 636B 636B
0050 6B7F FF7F 676B 676B 677F FF7F 636F 6F6F
0060 637F FF7F 676B 6B6B 677F FF7F 636F 676F
0070 637F FF7F 636F 676F 6F7F FF00 0000 0000
0080 0000 FFFF FFFF FFFF FFFF FFFF FFFF FFFF
0090 FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
00A0 FFFF FFFF FFFF FFFF FFFF FFF8 03B5 F800
00B0 A5B7 BBBC BF97 5515 853A B5F8 02B8 BEF8
00C0 01A8 F88B A7AF F89B ABF8 FFAC F834 AEF8
00D0 AAAD 7AC0 0136 0203 0405 4243 4445 8283
00E0 8485 C2C3 C4C5 FFFF FFFF FFFF FFFF FFFF
00F0 FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
0100 E370 2393 B1B2 B4F8 FEA2 F816 A1F8 00B7
0110 BB61 30CA 7270 2278 2252 C4C4 C4F8 03B0
0120 F800 A080 E2E2 20A0 E220 A0E2 20A0 3C23
0130 3014 FFC0 00AB 8D5C 3F38 EC37 3F30 4B8B
0140 F332 461B 303B F89B AB30 3B0B 5FD8 DE2D
0150 8DFB 9A32 699B B98B A919 1F09 5B19 1B89
0160 FBAB 3A5B F89B AB30 36F8 8BA7 F88C AD97
0170 BDA6 F800 F6E7 0DF5 337B 161D 8DFB 9B3A
0180 7217 87AD 1DFB 9A3A 72F8 8CA7 72FB 1032
0190 BA17 72FB 1032 BA72 FB10 32BA 1772 F810
01A0 32BA 1717 72FB 1032 BA17 72FB 1032 BA72
01B0 FB10 32BA 1772 FB10 3ABB 1686 F63B DEDE
01C0 DEDE DEDE DE7B 3FC6 37C8 F89B ABA7 27F8
01D0 10AC BC5B 9C57 271B 2C8C 3AD3 3033 F802
01E0 B6F8 4AA6 3FE4 D6F8 8BA7 D8DE 37EC 30DE
01F0 9831 0A34 9DB0 B425 F210 4252 E0AA 2349
0200 D3F8 00B6 B9F8 D6A6 F803 A9B5 06A5 07FF
0210 01AA 321B 89FC 08A9 8A30 0F09 5585 FC08
0220 A519 09FB FF3A 1B16 1786 FBE6 3A08 F88B
0230 A730 00D3 F806 A67A F848 A929 893A 3B31
0240 377B 2686 3A38 7A30 33D3 F88B A772 FB10
0250 3A4D 27EC 6CFA 0F5C 642C 3254 3060 642C
0260 F88B ABEB 0CF3 1B3A 632B 0C57 F810 5BE7
0270 3049
```


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

Publisher . . . . Quest Electronics
Editor . . . . . Bill Haslacher
Technical Consultant Paul Messinger
Art & Graphics . . . . . Holly Olson
Proofreading . . . . . Judy Pitkin
Photography . . . . Wayne Yamaguchi
Production . . . . . John Larimer

BLOCKADE

By Kevin Cutts

Blockade is a game played by two people. The object is to simply avoid all objects on the screen. The game starts out by putting up a simple border and two playing dots. Player one starts on the left side going right and player two starts on the right going left. As the game progresses the pieces move one square about every second.

Instead of erasing the old playing position the trail of each piece is left on the screen. These trails along with the borders provide the obstacles to be avoided. By using a hex keyboard and suitably labeling each key, the direction of the charging column can be changed. Strategy comes into play when each player moves his piece so that the trail left behind "boxes in" the opponent. The game is lost when your piece runs over something. (E.g. border, opponent's trail, your own trail, etc....)

I got the idea for this game at a local penny arcade. The only difference in the two games is that the commercial game uses a cute little pattern for each player. If this was used on the COSMAC 1K of memory would be needed to display and even this wouldn't allow for very complex patterns. Also, by using the hex keyboard, some confusion is created when two hands are reaching frantically for the keys.

Each player uses one register; RC for player one and RD for player two. The low order is used for the position of each piece. Since the high order location can only be 07, the computer simply assumes the real location to be 07 and the byte stored in RC.0 or RD.0. The high order of RC and RD are used for two things, the direction of travel and the location of the bit within the individual byte. This specific info is needed to do two things, first, so the computer has a counter to store any bit individually, and second, so the scanning of the next position will tell if that bit is occupied.

RX.0=Position
RX.1=Bit location (upper 4 bits), Direction (lower 4 bits)

The bit location code is 1 through 8 where the code refers to the bit position as shown:

8 7 6 5 4 3 2 1 bit locations 8=MSB 1=LSB

The Direction code is 0, 1, 2, or 4. These are derived fom the keypad entries and their meanings are:

Code	Meaning	Player 1 Keypad entries	Player 2 Keypad entries
0	Left	0	3
1	Right	4	7
2	Up	8	B
4	Down	C	F

Any other keypad entries are ignored. I simply place a piece of masking tape on each player's keys to show which key is which.

If the bit location becomes either 9 or 0 the byte position is incremented or decremented depending on which is needed. Of course, the bit local is reset as needed. I didn't use 0 1 2 3 for the direction codes because by using 0 1 2 4, I could simply shift right to decode. A typical value for, say RC would be 5237. This would mean that player one is going up from memory location 0737 and from the 5th bit. After moving the piece, the new location would be 5221.

TO PLAY

Step No.

1 Run at M(0400)
2 Play Game
3 When the tone sounds one player has lost. Depress and release Input to disable the tone.
4 Time is allowed to analyze the board.
5 To play again depress and release the Input. The game will automatically start over.

NOTE: Whatever code was last entered on the keypad will be used at the start. To get the preset starting positions, enter some code like "A" or "6".

Register Usage:

Reg. 0	T.V. Pointer
Reg. 1	Interrupt
Reg. 2	Stack
Reg. A	Program Counter
Reg. B	Game Speed Delay*
Reg. C	Player 1 Stats.
Reg. D	Player 2 Stats.
Reg. E	Memory Pointer for Game
Reg. F	Work Unit

*The value loaded into the high Reg. B is the timing value. To speed-up or slow-down the action, change the value in location 045D from the value given (04).

The code is set up for the Super Elf keypad but if you have a VIP system, overload the following program start at locationn 056B:

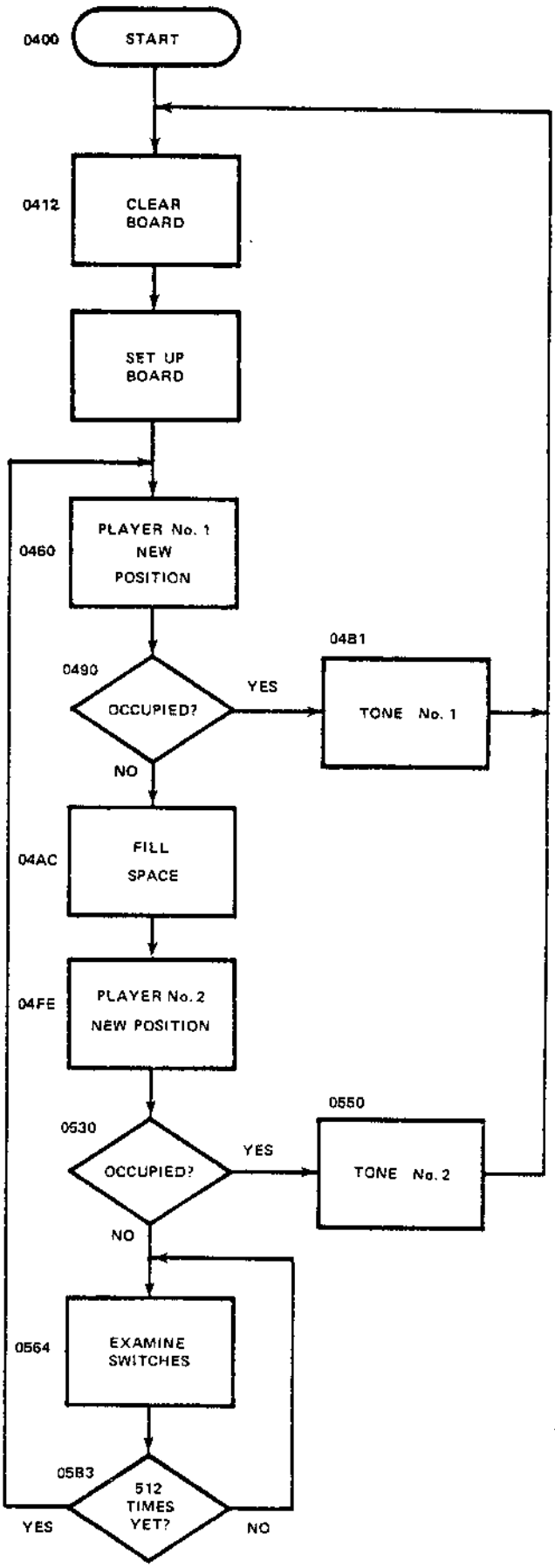
```
056B 62 3D74 9CFA
0570 F0F9 01BC 623D 7D9C FAF0 F900 BC62 3D86
0580 9CFA F0F9 02BC 623D 8F9C FAF0 F904 BC62
0590 3D98 9DFA F0F9 018D 623D A19D FAF0 F900
05A0 BD62 3DAA 9DFA F0F9 02BD 623D B39D FAF0
05B0 F904 BD
```

This has not been tried but should convert BLOCKADE for a scanning keyboard.

Some options I thought of were to have the setup changed so that obstacles would be put on the screen. Then the two players could fight it out among the obstacles. Also, by eliminating one player and making the whole screen white, a player could trace a path. Later, the other player could try to find his way through this maze. Of course, the games ending steps would have to be eliminated-to allow the first player to draw crosses, T's, etc....

The original game can be changed so that the borders are eliminated entirely. This would allow players to leave the board on one side and reappear on the other side. Another possiblity might be to "automate" (program) the path of the first player so that you play against the computer. There are lots of fun possiblities in the game.

```
0400 F804 BAF8 07AA DAF8 06B1 B2F8 02A1 F87F
0410 A261 F807 BEF8 00AE F8FF 5E1E 8EFF 083B
0420 188E 7633 3176 333C 7633 3CEE F880 5E30
0430 3F76 3B3C 763B 3CF8 015E 303F F800 5E1E
0440 8E3A 21F8 07BE F8F8 AEF8 FF5E 1E8E 3A49
0450 F881 ACF8 81BC F886 ADF8 10BD F804 BB9C
0460 7633 7776 3385 7633 8B9C FC10 BCFF 903B
0470 752C F810 BC30 8F9C FF10 BCFD 103B 831C
0480 F881 BC30 8F8C FF08 AC30 8F8C FC08 ACEE
0490 9CF6 F6F6 F6AF F801 BF2F 8F32 A29F FEBF
04A0 3099 8CAE F807 BEEE 9FF2 3AB1 9FF3 7330
04B0 FEF8 20AE 2E8E 3AB4 31BC 7B38 7A3F B137
04C0 BF3F C1EE 3012 0030 5C
04F0                                     9D76
0500 3316 7633 2476 332A 9DFC 10BD FF90 3B14
0510 2DF8 10BD 302E 9DFF 10BD FD10 3B22 1DF8
0520 81BD 302E 8DFF 08AD 302E 8DFC 08AD EE9D
0530 F6F6 F6F6 AFF8 01BF 2F8F 3241 9FFE BF30
0540 388D AEF8 07BE EE9F F23A 509F F373 3064
0550 F840 AE2E 8E3A 5331 5B7B 387A 3F50 375E
0560 3F60 30C1 F805 B9F8 B9A9 E96C FA0F F659
0570 7C00 F633 8D09 F659 FB03 3A7F F804 5933
0580 889C FAF0 F1BC 308D 9DFA F0F1 BD30 B3
05B0         28 9B3A 6430 C402 0408 0C03 2B9B
05C0 0FF8 04BA F804 BAD5 7017 F247 7285 74DF
05D0 F256 7415 7215 7A5D 7FD5 720C 7E04 79DF
05E0 780F 988D F28F F89D FA9B 7A8D B39C 3594
05F0 730B 729F 7388 BA89 7B1B 7A88 DA08 FA9F
0600 7270 2278 2252 C4C4 C4F8 07B0 F800 A080
0610 E2E2 20A0 E220 A0E2 20A0 3C0F 3000
```



BLOCKADE

ADDR	CODE	LABEL	OPCODE	OPERAND	COMMENT
0000	C00400				Jump to Program
0400	F804BA				
0403	F807AA				PC=RA
0406	DA				
0407	F806B1B2				Int. R1.1, R2.0
040B	F802A1				INterrupt
040E	F87FA2				Stack
0411	61				TV ON
0412	F807BE				Print
0415	F800AE				horizontal
0418	F8FF				line
041A	5E1E				on display
041C	8EFF08				
041F	3B18				End of line?
0421	8E76				
0423	3331				See if end line
0425	76				Branch to print
0426	333C				blanks. Start
0428	76				
0429	333C				
042B	EE				
042C	F8805E				Left border
042F	303F				
0431	76				
0432	3B3C				
0434	76				
0435	3B3C				
0437	F8015E				Right border
043A	303F				
043C	F8005E				
043F	1E8E				See if display
0441	3A21				full; if not go
					back
0443	F807BE				
0446	F8F8AE				Print bottom line
0449	F8FF5E				Keep going till
044C	1E8E				bottom full
044E	3A49				
0450	F881AC				INt. position
0453	F881BC				and direction
0456	F886AD				of player 1 & 2
0459	F810BD				(player 1=RC)
045C	F804BB				player 2=RD)
045F	9C				
0460	763377				GTO Right
0463	763385				GTO UP
0466	76338B				GTO Down
0469	9C				Assume Left
046A	FC10BC				
046D	FF90				
046F	3B75				
0471	2CF810				
0474	BC				
0475	308F				
0477	9CFF10				Right
047A	BC				
047B	FD10				
047D	3B83				
047F	1C				
0480	F881BC				
0483	308F				
0485	8CFF08AC				Up
0489	308F				
048B	8CFC08AC				Down
048F	EE				
0490	9C				See if
0491	F6F6F6F6				new position
0495	AF				is occupied
0496	F801BF				
0499	2F8F32A2				
049D	9FFEBF				
04A0	3099				
04A2	8CAE				
04A4	F807BE				
04A7	EE				
04A8	9FF2				
04AA	3AB1				If spot is
04AC	9FF3				occupied GTO
04AE	73				tone; otherwise
04AF	30FE				print new space
04B1	F820AE				
04B4	2E8E				Player 1
04B6	3AB4				has lost
04B8	31BC				tone
04BA	7B387A				
04BD	3FB1				
04BF	37BF				
04C1	3FC1				
04C3	EE				
04C4	3012				Restart
04C6	00				game;
04C7	305C				Move pieces

BREAK TO LOCATION

04FE

QUESTDATA COSMAC CLUB

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

```
ADDR CODE     LABEL   OPCODE  OPERAND   COMMENT

04FE 9D
04FF 76
0500 3316                               GTO Right
0502 763324                             GTO Up
0505 76332A                             GTO Down
0508 9DFC10                             Assume left
050B BD
050C FF90
050E 3B14
0510 2D
0511 F810BD
0514 302E
0516 9DFF10                             Right
0519 BD
051A FD10
051C 3B22
051E 1D
051F F881BD
0522 302E
0524 8DFF08AD                           Up
0528 302E
052A 8DFC08AD                           Down
052E EE
052F 9D
0530 F6F6F6F6                           See if
0534 AF                                 new position
0535 F801BF                             is occupied
0538 2F8F3241
053C 9FFE8F
053F 3038
0541 8DAE
0543 F807BE
0546 EE
0547 9FF23A50                           If spot is
                                        occupied
054B 9FF3                               generate tone;
054D 73                                 otherwise fill
                                        space
054E 3064
0550 F840AE                             Player 2 has
                                        lost
0553 2E8E3A53
0557 315B
0559 7B387A
055C 3F50
055E 375E
0560 3F60
0562 30C1
0564 F805B9                             Continue
0567 F8B9A9                             game
056A E9
056B 6C       HERE      INP4
056C FA0F               ANI
056E F6                 SHR
056F 59                 STR
```

```
ADDR CODE     LABEL   OPCODE  OPERAND   COMMENT

0570 7C00               ADCI    H'00'
0572 F6                 SHR
0573 338D               BNF     TSTINC
0575 09                 LDN
0576 F6                 SHR
0577 59                 STR
0578 FB03               XRI     H'03'
057A 3A7F               BNZ     TSTPLR
057C F804               LDI     H'04'
057E 59                 STR
057F 3388     TSTPLR:   BDF     PLRB
0581 9C                 GHI     C
0582 FAF0               ANI     H'F0'
0584 F1                 OR
0585 BC                 PHI     C
0586 308D               BR      TSTINC
0588 9D       PLRB:     GHI     D
0589 FAF0               ANI     H'F0'
058B F1                 OR
058C BD                 PHI     D
058D 3083     TSTINC:   BR      HERE
```

BREAK TO LOCATION 05B3

```
ADDR CODE     LABEL   OPCODE  OPERAND   COMMENT

05B3 2B9B                               Decrement
05B5 3A64                               the B and see
05B730C4                                If its time to
                                        move data
05B9 0004080C
05BD 032B9B0F
05C1 F804BA
05C4 F804BA
```

DISPLAY REFRESH

```
ADDR CODE     LABEL   OPCODE  OPERAND   COMMENT

0600 72
0601 70
0602 2278
0604 2252
0606 C4C4C4
0609 F807B0
060C F800A0                             Page 07=Display
060F 80E2
0611 E220A0
0614 E220A0
0617 E220A0
061A 3C0F
061C 3000
```

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

MUSIC FOR EXPANDED MEMORIES

By Allan Armstrong

The music algorithm presented in issue seven of QUESTDATA is a wonderful program, but will not run on an expanded memory system. This is due to random data appearing in R(5).1, at reset. The program presented in this article overcoms the difficulty and even has a few extra bonuses. The tempo byte and the delay byte are located in the data table for the particular piece of music being played. This leads to greater flexibility of the algorithm. I recommend a value of 06 for the tempo byte, location 005C, and 01 for the delay byte, locaion 005D. The basic program occupies locations 0000 through 005B. The data occupies locations 005C through to the end of memory. Data for James Hook's (1746-1827) "Sonata in G for Descant Recorder", and "Pianoforte" occupies locations 005C through 02E4. Try it, you'll like it! Press "I" to try it again.

Gavotte from J.S. Bach's Fifth French Suite

```
ADDR CODE
005C 06 (TEMPO)
005D 01 (PAUSE)
005E 7B17621F
0062 492D 2927 2E22 621F 5227
006C 7D37 5227 4133
0072 373F 1F37 2033 242D 1F37 2033 1B3F
0080 2033 1F37 1B3F 1847 7B17 621F
008C 5227 2927 2E22 621F 5227
0096 4530 242D 2927 2E22 242D 311F 2927
00A4 2E22 242D 6E1B 311F 2E22 2927 2E22
00B2 922D 373F 492D
00B8 BA22 311F 2E22 2927 242D
00C2 C41F 7B17 621F
00C8 5227 2E22 311F 6E1B 5D22
00D2 9C2A 1F37 2230 272A 2927
00DC 2E22 311F 2E22 2927 272A 2927 2E22 311F
00EC 371B 3E17 371B 311F 2E22 311F 371B 3E17
00FC 4115 371B 7B17 5227 2E22 272A
0108 A427 3147 3E37
010E 922D 2033 1F37 1B3F 1847
0118 2A24 242D 5227 2E22 1847 1B3F 1F37
0126 2033 1B3F 1F37 2033 242D 2033 242D 2927
0136 6D3F 8315 6E1B
013C 5D22 2E22 311F 6E1B 5D22
0146 492D 242D 2927 2E22 311F 371B 2E22
0154 311F 2927 6E1B 1722 181F 5D22 311F
0162 C41F
0164 00



0000 F800 B5F8 5CA5 45A6 45A7 F800 B4F8 29A4
0010 8654 F84C A487 54C4 C4C4 E5F0 3A22 3700
0020 301E A815 6425 F0A7 F807 A987 FC84 3338
0030 3135 7B30 387A 3038 87FF 013A 3989 FF01
0040 A93A 5488 FF01 A83A 287A 15F8 01B3 2393
0050 3A4E 301B C430 5830 5A30 2B
```

```
ADDR CODE     LABEL  OPCODE OPERAND COMMENT

0000 F800     RSC    LDI    00      Initialize Pointer
0002 B5              PHI    5       *to pick up quiet
0003 F85C            LDI    X'5C'   *data
0005 A5              PLO    5
0006 45              LDA    5
0007 A6              PLO    6
0008 45              LDA    5
0009 A7              PLO    7       Put data in R7.0
000A F800            LDI    00      Initialize tempo ptr
000C B4              PHI    4
000D F829            LDI    X'29'
000F A4              PLO    4
0010 86              GLO    6
0011 54              STR    4       Store tempo
0012 F84C            LDI    X'4C'   Pointer to short
0014 A4              PLO    4       *quiet
0015 87              GLO    7       Get quiet data
0016 54              STR    4       *and store
0017 C4C4C4          NOP's
001A E5              SEX    5       X=5
001B F0       NN     LDX
001C 3A22            BNZ    STDUR   Repeat score if
001E 3700            B4     RSC     *data is 00 and
0020 301E            BR     *-2     *INPUT depressed
0022 A8       STDUR  PLO    8       Store duration
0023 15              INC    5       *in reg 8
0024 64              OUT    4       Display pitch
0025 25              DEC    5
0026 F0              LDX            *store it
0027 A7              PLO    7       *in reg 7
0028 F800     ST     LDI    00      Store tempo; the
002A A9              PLO    9       *any loc. 29 data
                                    ok
002B 87       SQ     GLO    7       Stop alternating
002C FCB4            ADI    X'B4'   *Q if a rest
002E 3338            BDF    RP
0030 3135            BQ     TOQ     If Q on, turn
0032 7B              SEQ            *it off; if
0033 3038            BR     *+5     *off, turn it
0035 7A       TOQ    REQ            *on
0036 3038            BR     *+2
0038 87       RP     GLO    7       Repeat as often
0039 FF01            SMI    01      *as pitch
003B 3A39            BNZ    *-2     *indicates
003D 89              GLO    9       Repeat as often
003E FF01            SMI    01      *as tempo
0040 A9              PLO    9       *indicates
0041 3A54            BNZ    PI
0043 88              GLO    8       Repeat as often
0044 FF01            SMI    01      *as duration
0046 A8              PLO    8       *indicates
0047 3A28            BNZ    ST
0049 7A              REQ            When note done,
004A 15              INC    5       *turn off Q; insert
004B F800            LDI    00      *short quiet;loc.4C
004D BF              PHI    F       *can be any since
004E 2F              DEC    F       *loc. 5D is pause
004F 9F              GHI    F       *data
0050 3A4E            BNZ    *-2
0052 301B            BR     NN
0054 C4       PI     NOP            Delay to make
0055 3058            BR     *+3     *paths take
0058 305A            BR     *+2     *same time
005A 302B            BR     SQ
005C TEMPO DATA (06 is recommended)
005D PAUSE DATA (01 is recommended)
005E TABLE OF MUSIC BEGINS HERE
```

```
ADDR CODE

005C 06-TEMPO
005D 01-PAUSE
005E 3033 0C47 0D3F 1037
0066 3033 0C47 0D3F 1037
006E 2033 1033 242D 1037
0076 1033 122D 1427 2A24 122D
**************************
0080 2927 1033 0C47 0D3F 1037
008A 3033 0C47 0D3F 1037
0092 2033 1427 2A24 122D
009A 2033 1427 3033
**************************
00A0 3033 0C47 0D3F 1037
00A8 3033 0C47 0D3F 1037
00B0 2033 1033 242D 1037
0088 1033 122D 1427 2A24 122D
**************************
00C2 2927 1033 0C47 0D3F 1037
00CC 3033 0C47 0D3F 1037
00D4 2033 1427 2A24 122D
00DC 2033 1427 3033
**************************
00E2 491F 521B
00E6 181F 1427 1033 1847 181F
00F0 371B 1722 311F 1427
00F8 2033 1033 122D 1427 1524
**************************
0102 491F 521B
0106 181F 1427 1033 1847 181F
0110 2927 122D 1033 1037 0D3F
011A 0C47 1037 122D 311F 0C4C
**************************
0124 521B 2E22 122D
012A 181F 122D 1037 0C47 0D3F 1037
0136 2033 1033 2033 1033
013E 2F37 244C
0142 521B 2E22 122D
0148 181F 122D 1037 0C47 0D3F 1037
0154 2033 1033 2033 1033
015C 2F37 0D3F 0C47 1037
**************************
0164 3033 0C47 003F 1037
016C 3033 0C47 0D3F 1037
0174 2033 1033 242D 1037
017C 1033 122D 1427 2A24 122D
**************************
0186 2927 1033 0C47 0D3F 1037
0190 3033 0C47 0D3F 1037
0198 2033 1427 2A24 122D
01A0 2033 1427 3033
**************************
01A6 3A2A 362D
01AA 3033 362D
01AE 132A 181F 1524 132A 122D 1033
01BA 1033 1037 1033 122D 1037 0C47
**************************
01C6 3A2A 362D
01CA 3033 362D
01CE 132A 122D 1033 2033 1033
01C8 1037 122D 1037 1847 0C4C
**************************
01DC 272A 1033 242D 1037
01DA 2033 0C47 1847 0C47
01E2 1847 0C47 2033 1033
01EA 1033 1037 1033 122D 1037 0C47
**************************
01F6 272A 1033 242D 1037
01FE 2033 0C47 0C47 122D 152A
0208 132A 122D 1033 132A 122D 1033
0214 2F37 244C
**************************
0218 3033 0C47 0D3F 1037
0220 3033 0C47 0D3F 1037
0228 2033 1033 242D 1037
0230 1033 122D 1427 2A24 122D
**************************
023A 3033 0C47 0D3F 1037
0242 3033 0C47 0D3F 1037
024A 2033 1427 2A24 122D
0252 2033 1427 2033 1033
**************************
025A 521B 4024   •
025E 181F 1722 181F 1427 122D 1033
026A 521B 4024
026E 491F 2927 1033
**************************
0274 521B 4024
0278 181F 1722 181F 1427 122D 1033
0284 122D 1524 1427 122D 1033 1037
02A0 3033 244C
**************************
02A4 244C 4024
02A8 3D27 244C
02AC 244C 2F37
02B0 3033 181F 1524 122D
**************************
02B8 3033 0C47 0D3F 1037
02C0 3033 181F 1524 122D
02C8 3033 0C47 0D3F 1037
02D0 3033 181F 1524 122D
**************************
02D8 3033 3D27
02DC 3033 3D27
02E0 6033
02E2 6033
02E4 00
```

PATCH SQRT FUNCTION FOR ELF-II TINY BASIC

By Chuck Reid

A compact useful extension to TINY BASIC is always welcome. Questdata Issue #12 carried the article "Square Root Extraction for Tiny Basic" and thanks to the author, Mr. Liescheski, I now have the Square Root function on my ELF-II TINY BASIC.

Here are the "mods" as they fit on an ELF-II Netronics Terminal "TINY BASIC":

```
ADDR DATA

0020 0BB0
011C 0BB0
```

Load the Square Root hexadecimal code into memory from 0B87 to 0BAD. Refer to Page 13 of Questdata Issue #12. The memory locations given are 0F50 to 0F76. Having relocated the code, you must change these locations with the given data:

```
ADDR NEW DATA   (DATA IN ARTICLE)

0B8B 90         (59)
0BA4 A8         (71)
0BA7 95         (5E)
```

To use the Square Root function, its 'USR' statement changes as follows:

LET X = USR (2951, expr)

'expr' is already familiar to the users of the ELF-II TINY BASIC.

[EDITOR'S NOTE: We are printing this without verification in hopes that ELF-II owners can find them useful.]

PNR CALCULATOR

By Ron Zoscak

Most of the programs I write either blow themselves away or do something other than what I had intended them to do, because the computer, with stubborn literal mindedness, does what I tell it to do instead of what I meant it to do. On occasion, however, I do come up with something that not only works, but is actually useful. Such is the case with this program, which I call a two-banger hex calculator with PNR (Perverse Nolish Rotation).

It will add two bytes together or subtract one from another. To use it, load in the program then reset and run. For addition, key in the digits 11, then depress and release the input button. Next key in the first addend, and press and release the input button. The byte you just entered will be displayed on the LED readout. Then key in the second addend and depress the input button. The digits you entered will be displayed. When you release the input button, the sum of the two bytes will be displayed. If there was an overflow, the Q LED will light. The program is now back in the function mode.

To subtract, key in the digits 19, then depress and release the input button. Next key in the number you wish to subtract, and depress and release the input button. Then key in the number to be subtracted from, and depress the input button. When you release the input button, the difference between the bytes will be displayed. If a borrow occurred, indicating that you entered the larger number first, the Q LED will light. There is no need to reset and run again after an overflow or borrow, because there is a 7A instruction to make sure the Q is off if none of these two conditions occur after an operation. However, you must enter the code for the function every time.

This program works on the Basic Elf only. For the Expanded System try these patches:

```
LOC  CODES
0000 3041
0041 F800
0043 B2B3B4
0046 BEBF
0048 F8FF
004A 3002
```

LISTING FOR HEX CALCULATOR

```
ADDR CODE   LABEL   OPCODE OPERAND COMMENT

0000 F8FF   BEGIN   LDI    X'FF'  R1.0 points
                                  to work space
0002 A1             PLO           for function
0003 E1             SEX           determination
0004 3F04           BN4    *      Wait here if
                                  input button up
0006 3706           B4     *      Wait here if
                                  input button down
0008 6C             INP4          Input address
0009 A4             PLO           of function
                                  routine desired
000A F835           LDI    X'35'  Load address
                                  for math instruction
000C A2             PLO           IN R2.0
000D F839           LDI           Load Address
                                  for branch
                                  instruction
000F A3             PLO           IN R3.0
0010 D4             SEP           Go to chosen
                                  routine
0011 F8F4           LDI           Write add
                                  instruction
0013 52             STR           in location 35
0014 F833           LDI           Write BDF
                                  instruction
0016 53             STR           in location 39
0017 3020           BR            Go to main
                                  section
0019 F8F7           LDI           Write SM
                                  instruction
001B 52             STR           in location 35
001C F83B           LDI           Write BNF
                                  instruction
001E 53             STR           in location 39
001F C4             NOP           Filler
0020 F8FE           LDI           Put address
                                  of work space
0022 AF             PLO           in register F.0
0023 EF             SEX           RF.0 is pointer
0024 3F24           BN4    *      Wait here
                                  if input button up
0026 6C             INP4          Input first byte
0027 64             OUT4          and display it
0028 3728           B4     *      Wait here if
                                  input button down
002A 3F2A           BN4    *      Wait here if
                                  input button up
002C 6C             INP4          Input second byte
002D 64             OUT4          and display it
002E 372E           B4     *      Wait here if
                                  Input button down
0030 2F             DEC           Point RF.0 back
                                  to second byte
0031 8F             GLO           Point RE.0 to
0032 AE             PLO           second byte
0033 2F             DEC           Point RF.0 back
                                  to first byte
0034 0E             LDN           Put second byte
                                  in the D register
0035 00                           Put anything here.
                                  This location
                                  written by set up
                                  sections
```

```
ADDR CODE    LABEL   OPCODE  OPERAND   COMMENT
0036 5E              STR             Store result in
                                     location FF
0037 1F              INC             Point RF.0 to
                                     result
0038 64              OUT4            Display result
0039 00                              Put anything here.

                                     This location
                                     written by set up
                                     sections
003A 3E              X'3E'           If overflow or
                                     borrow, go to
                                     location 3E
003B 7A              REQ             No overflow or
                                     borrow, so make
                                     sure Q off
003C 3000            BR      BEGIN   Then go back and
                                     get ready
                                     for a new
                                     calculation
003E 7B              SEQ             Turn on Q to
                                     indicate overflow
                                     or borrow
003F 3000            BR      BEGIN   Then go back and
                                     get ready for a
                                     new caiculation
```

```
0000 3041 A1E1 3F04 3706 6CA4 F835 A2F8 39A3
0010 D4F8 F452 F833 5330 20F8 F752 F83B 53C4
0020 F8FE AFEF 3F24 6C64 3728 3F2A 6C64 372E
0030 2F8F AE2F 0EF4 5E1F 6433 3E7A 3000 7B30
0040 00F8 00B2 B3B4 BEBF F8FF 3002 CC1B B824
0050 6C6D
```

SUPER ELF CASSETTE FORMAT

The Super Elf Super Monitor and Super Basic use the following cassette format.

This approach allows automatic loading into memory since the starting address is on the tape. A test can be made to see if there is enough space in memory for the file prior to loading because the file also contains the number of bytes. Each byte includes a parity bit which is used for error checking. The rate of recording and playback is approximately 1200 bytes per second with a 1.79MH clock. Reliable operation has been achieved (with a recommended recorder) at 2000 bytes per second using a 3.0MH clock. The read software used in the Super Monitor and Super Basic allows starting the recorder with the cassette fully rewound. The plastic leader/tape splice does not cause problems and it is not necessary to advance the cassette past the leader prior to starting. In fact there are **no level adjustments or tape positioning required** (when using one of the recommended recorders). Just load, set volume and tone to max and go. This format is self clocking and insensitive to speed variations of over +-15%.

The Format Consists of:

1. A 10 second leader of all 'ones' terminated with a zero bit.
2. Data bytes.
3. A 5 sec trailer of all 'zeros'.

The Data Section Consists of:

1. Starting address 2 bytes Hi/Low.
   (0000 TO FFFF HEX)
2. Number of bytes 2 bytes Hi/Low.
   (0000 TO FF00 HEX)
3. Data bytes 8 bits with the most significant (MSB) first with an added parity bit. (For a total of 9 bits). Even parity is used, i.e. the sum of the ones in the data plus the parity bit is always an even number.

```
DATA 0111 01 01  PARITY=1      (6'1's)
DATA 0011 00 11  PARITY=0      (4'1'S)
```

A bit is defined as a high level followed by an equal length low level. A zero bit is three times the length of a one bit. The end of a bit is the beginning of the next bit.

The following bit times are for a 1.79 MH clock. The effects of other clock frequencies are discussed later.

1. A ONE bit has a 206 microsec ON time and a 609 to 627 microsec OFF time.
2. A ZERO bit has a 618 microsec ON time and a 609 to 627 microsec OFF time.

The standard Super Elf with a 1.79 MH clock and either Super Basic or the Super Monitor uses this format. If your system uses a different clock frequency, make the following changes to For Different Clock Rates, Solve this equation:

VALUE=8 X CLOCK RATE (MH)-1.3

Round off VALUE, convert to hexadecimal and patch into location 8193 in the Super Monitor V1.1 or 2.0; and location 2947 in Super Basic 1.4; QKOC is the location in the cassette read routine.

Super Elf owners may ignore this section. Non Super Elf owners must use the listed cassette read routine and may need to use the schematic for read hardware. Elf II owners usually can successfully read with their own hardware. The following table lists the changes required for the different sense lines.

	EF1	EF2	EF3	EF4
QK16	34	35	36	37
QK1D	3C	3D	3E	3F
QK2C	3C	3D	3E	3F
QK41	34	35	36	37
QK48	3C	3D	3E	3F
QK53	3C	3D	3E	3F

Where QK is the Quarter K of memory in which the cassette read routine is loaded. NOTE: Do not load the read routine in a page which will be written into by the read routine.

If you have trouble reading you may have the signal inverted (all recorders are not the same even the same brand and model). This software usually but not always accepts either normal or inverted data, it depends upon the amount of distortion introduced by the recorder. The following table inverts the data.

	EF1	EF2	EF3	EF4
QK16	3C	3D	3E	3F
QK1D	34	35	36	37
QK2C	34	35	36	37
QK41	3C	3D	3E	3F
QK48	34	35	36	37
QD53	34	35	36	37

To use the cassette read routine (non Super Elf owners only), follow the following instructions.

1. Load program per this load listing. QK refers to your page of memory where you want the program to reside:

2. Branch into the program from register 0 into location QK00. If you wish to branch into the program from any other register (but not register 2 or 5) then change the "90" instruction at location QK00 to "9P" where P is the same as the register designation which you are using. Then branch into QK00.

3. After starting the program, load the cassette at the beginning, start recorder in playback with maximum volume and treble on the tone control. There is a 10 second leader and then the low address of the current loaction of memory being loaded off the cassette will be displayed. The rate is about 1200 baud. At the end of loading, an "AA" will show up on the display (If you have a hex display at output port 64).

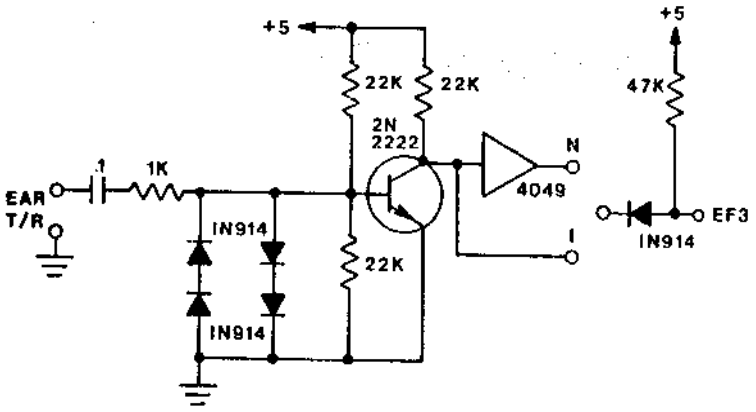
4. If a parity error is detected, an "EE" will be displayed (on the hex display). Check the program for loading errors and try re-running the playback. Also check your hardware setup.

Future articles will be discussed:

a. The Elf II format and how non Elf II owners can read Elf II cassettes.

b. The Super Elf Write/Read software source listings.

```
QK00 90B3 E2B5 B2F8 72A2 F812 A5F8 0DB6 F83E
QK10 A3D3 F808 A6A7 3616 96FF 013B 223E 1927
QK20 302E F800 FC01 3B2C F8EE 306F 3E24 8632
QK30 3797 7EB7 2630 1687 F633 2897 3011 F80A
QK40 B436 4196 FF01 3B50 3E44 9432 4124 3041
QK50 943A 3E3E 53D5 B8D5 A8D5 B9D5 A929 99FC
QK60 01B9 D558 8852 6422 1829 993A 62F8 AA52
QK70 6400
```



A TRICKY SOLUTION TO A CLEAR PROBLEM

By S. G. Grant

In QUESTDATA No. 8, Jay Mallin in his article **DOODLE PROGRAM** on page 8 issued a challenge of sorts, "It's possible to write a program that will clear all but one byte-try it sometime." Well, reader Arthur S. G. Grant has come up with a program which cleverly does the job in 12 machine language bytes. Do any readers have new challenges in search of solutions? If so, let us know. QUESTDATA readers can solve any challenge, right?

```
ADDR CODE    LABEL   OPCODE OPERAND COMMENT

0000 3030            BR
0001 00                             Hint- this
 ...                                later be-
 ...                                comes im-
 ...                                portant
 ...                                (this 00 in
 ...                                01) No more
 ...                                hints-tracing
 ...                                thru this
 ...                                program is
 ...                                part of the
 ...                                fun
0030 90              GHI    R0
0031 A1              PLO    R1
0032 B1              PHI    R1
0033 E1              SEX1
0034 F873            LDI    '73'
0036 73              STXD
0037 90              GHI    R0
0038 A0              PLO    R0
```

INDEX TO VOLUME 1

MATHEMATICS

MUSIC

MISCELLANEOUS

VIDEO GRAPHICS

Notes From the Publisher

The twelve issues of Volume I are now complete. All things considered we had a very good year with Questdata in a startup phase. We will be working very hard to increase the timeliness of the publication. Our goal is an issue every month and we will continue to stay at a subscription rate of 12 issues for $12.00 in spite of rising costs. At the same time our intent is to broaden the material to include hardware articles and tutorials as well as programs. Program content will be as broad as possible ranging from 256 bytes to 32K, Hex through full Basic, video, music, games, and hardware as well as serious routines of general interest. Issue 13 is our first issue completed on the word processor. Future issues will continue to be printed by the computer which will help us automate the process.

We have had excellent material sent in by our readers and it is incresing each month. We are seeing many large programs dedicated to expanded 1802 systems. Keep them coming. At the same time we need programs that run on 1/4k RAM as well as short routines and articles that will interest our users of unexpanded systems. Remember, we are one of the few dedicated publications that pay for material printed. We prefer programs to be documented with comments and a flowchart along with supporting articles. Short programs, however, can be accepted with a limited amount of additional material.

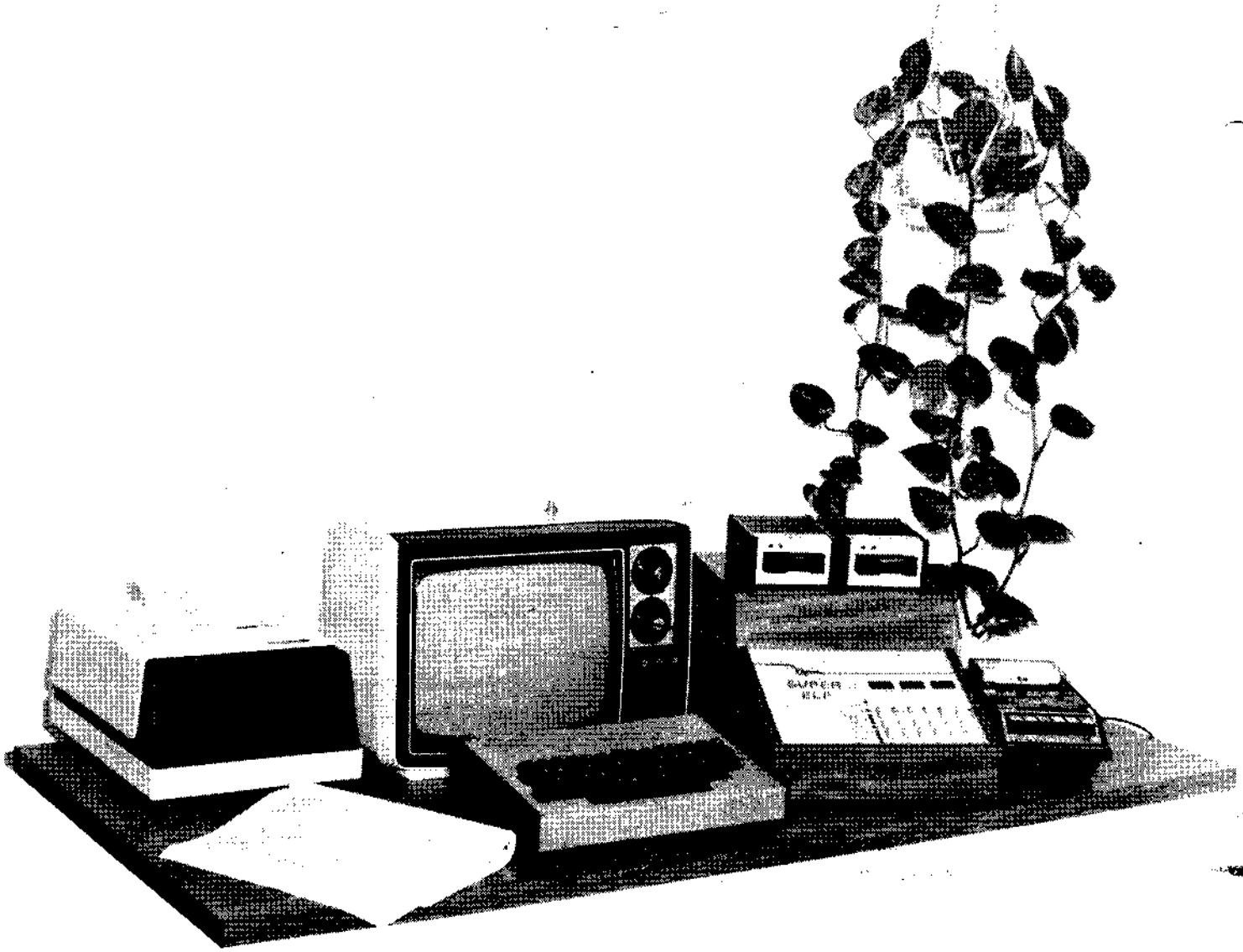
Our readership now numbers in the thousands from every state and many countries around the world. We are dedicated to the 1802 system user and will continue to expand the capabilities of the hardware and software to make the 1802 even more interesting and usable.

We welcome your comments and ideas for future issues. Thank you for your support.

A bound version of the first twelve issues with index issues of Questdata is now available for 16.50 plus 1.00 shipping. It is also available in a hard cover "gift edition" for 19.50 plus 1.00 shipping.

CONTRIBUTING AUTHORS

Bailey, Kirk	Seminole, FL
Binning, Ron	Abbot, TX
Davis, Reed	San Jose, CA
Dzombak, Ivan	Latrobe, PA
Garcia, Edgar	Lafayette, LA
Haslacher, Bill	San Jose, CA
Jackson, Eugene	San Jose, CA
Jenks, Jim	Mt. Prospect, IL
Krammer, Jack	N.Massapequa, NY
Lewis, Bobby	Waterford, CT
Liecheski, Phillip B. III	Houston, TX
Lindberg, D.J.	Marine on St.Croix, MN
Mc Cormick, Edward	Los Angeles, CA
Mallin, Jay	Coral Gables, FL
Mathew, Branden	Aptos, CA
Moews, Paul C	Storrs, CT
Moffie, Richard	North Hollywood, CA
Nicholson, James	Corte Madera, CA
Oats, Floyd	Rex, GA
Permenter, Terry O	Belling, WA
Petrou, Alexander	North Ryde, NSW, Australia
Ramirez, Ted	Fremont, CA
Richie, Bob	Champaign, IL
Smedstad, Gus	Grand Rapids, MI
Taylor, David	APO San Francisco, CA
Taylor, Patrick	Bloomsburg, PA
Tope, Mike	Canton, OH
Tyborski, Michael	Greendale, WI
Van Horn, Gerald	Junction City, OR
Wallace, Alan	Goldsboro, N.C.
Wendell, Mark	Los Angeles, CA



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

13

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

ADDRESS CORRECTION REQUESTED

BULK RATE
U.S. Postage Paid
QUEST
Electronics

Permit No. 649
Santa Clara, CA