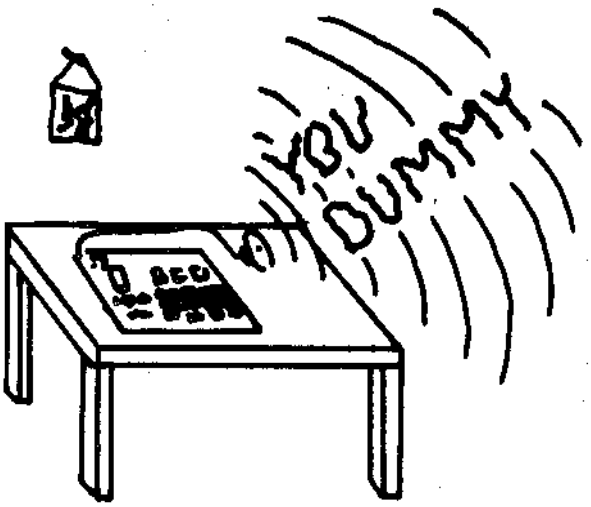
Questdata
Volume 2 Issue #2 ©

THE TALKING ELF



by
Bobby R. Lewis

The following program will allow RCA-1802 users to digitize voice information from a microphone input, display the information graphically, and reproduce it for output via a speaker. The only additional hardware required is a microphone and mini-speaker amplifier.

The program, as written, will run on systems configured as follows:

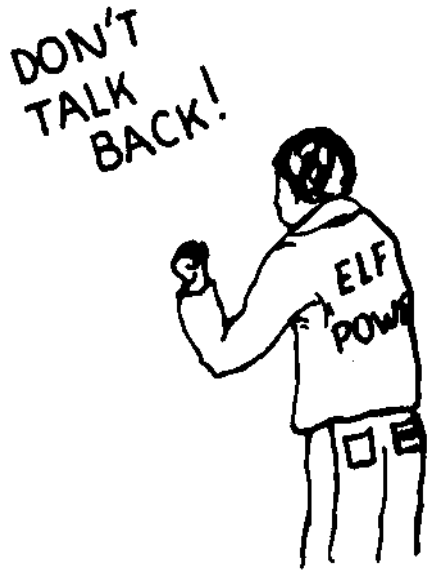
1. EF3 - cassette input
2. Q - speaker output
3. EF4 - input switch
4. Continuous RAM memory from address 0000 to OFFF.
5. The amplifier will be attached as follows:
   a. Microphone to input
   b. Ext speaker output attached to the cassette input.
6. Non Super Elf owners can connect the Q output to a speaker via a transistor driver or the cassette out line (Q) can be fed into the phono input of a audio amplifier for optimum results.

The program can be tested without the amplifier by playing a normal music cassette into the cassette input line while the program is running. This will still allow reproduction and output of the information on the tape. You could also record you own voice on a tape and play it through the cassette input line.

The following changes must be made to the program for use on an Elf II.

Location	Contents
0016	61
001D	35
001F	3D
0027	35
0029	3D
005D	61

In addition, a speaker and amplifier must be connected to the Q output line.



OPERATING INSTRUCTIONS

1. Load the program into the required address space.

2. Place the computer in the run mode.

3. The graphics display will show pages zero through three on the screen with the program at the top of the display.

4. Press input and immediately start talking into the microphone. You can usually say from 7 to 12 words. The screen will go blank while the computer is inputting the voice information.

5. The display coming back on signals the end of the voice input cycle, and will be displaying pages 0C, 0D, 0E, and 0F.

6. Press input and pages 08, 09, 0A and 0B will be displayed.

7. Press input and pages 04, 05, 06 and 07 will be displayed.

8. Press input and pages 00, 01, 02 and 03 will be displayed.

9. At this point, press input again and the information will be output to the Q line.

10. Pressing input again restarts the sequence at step 4 above.

```
U000 ;                          0001..
0000 ;                          0002..
0000 ;                          0003..
0000 ;                          0004..VOICE INPUT
0000 ;                          0005..GRAPHICS
0000 ;                          0006..AND VOICE
0000 ;                          0007..OUTPUT
0000 ;                          0008..
0000 ;                          0009..BY:  BOBBY
0000 ;                          0010..R. LEWIS
0000 ;                          0011..
0000 ;                          0012..1980
0000 ;                          0013..
0000 ;                          0014..REGISTER
0000 ;                          0015..EQUATES
0000 ;                          0016..
0000 ;                          0017 DMA=0
0000 ;                          0018 INT=1
0000 ;                          0019 SP=2
0000 ;                          0020 PC=3
0000 ;                          0021 BPTR=4
0000 ;                          0022 TMP2=#0E
0000 ;                          0023 TMP3=#0C
0000 ;                          0024 VIN=#0F
0000 ;                          0025 TVON=1
0000 ;                          0026 TVOF=2
0000 ;                          0027 BLOC1=#0C00
0000 ;                          0028 BLOC2=#0800
0000 ;                          0029 BLOC3=#0400
0000 ;                          0030 BLOC4=#0000
```

ADDR	CODE	LABEL	OPCODE	OPERAND	COMMENT
0000	;		ORG	0	0031
0000	F800;	STRT:	LDI	A.1(STRT)	0032
0002	B1;		PHI	INT	0033
0003	B2;		PHI	SP	0034
0004	B3;		PHI	PC	0035
0005	B4;		PHI	BPTR	0036
0006	F896;		LDI	A.0(INTP)	0037
0008	A1;		PLO	INT	0038
0009	F8FF;		LDI	#FF	0039
000B	A2;		PLO	SP	0040
000C	F811;		LDI	A.0(GRFX)	0041
000E	A3;		PLO	PC	0042
000F	E2;		SEX	SP	0043
0010	D3;		SEP	PC	0044
0011	69;	GRFX:	INP	TVON	0045
0012	3712;		B4	*	0046
0014	3F14;		BN4	*	0047
0016	62;		OUT	TVOF	0048
0017	F80F;		LDI	#0F	0049
0019	BF;		PHI	VIN	0050
001A	F8FF;		LDI	#FF	0051
001C	AF;		PLO	VIN	0052
001D	361D;		B3	*	0053
001F	3E1F;		BN3	*	0054
0021	F800;	LOP1:	LDI	#00	0055
0023	BE;		PHI	TMP2	0056
0024	F808;		LDI	#08	0057
0026	AE;		PLO	TMP2	0058
0027	362B;	LOP:	B3	ZERO	0059
0029	3E30;		BN3	ONE	0060
002B	F800;	ZERO:	LDI	#00	0061
002D	FE;		SHL		0062
002E	3033;		BR	DISP	0063
0030	F880;	ONE:	LDI	#80	0064
0032	FE;		SHL		0065
0033	0F;	DISP:	LDN	VIN	0066
0034	76;		SHRC		0067
0035	5F;		STR	VIN	0068
0036	2E;		DEC	TMP2	0069
0037	8E;		GLO	TMP2	0070
0038	3A27;		BNZ	LOP	0071
003A	2F;		DEC	VIN	0072
003B	9F;		GHI	VIN	0073
003C	F800;		XRI	#00	0074
003E	3A21;		BNZ	LOP1	0075
0040	F80C;		LDI	A.1(BLOC1)	0076
0042	B4;		PHI	BPTR	0077
0043	69;		INP	TVON	0078
0044	3744;		B4	*	0079
0046	3F46;		BN4	*	0080
0048	F808;		LDI	A.1(BLOC2)	0081
004A	B4;		PHI	BPTR	0082
004B	374B;		B4	*	0083
004D	3F4D;		BN4	*	0084
004F	F804;		LDI	A.1(BLOC3)	0085
0051	B4;		PHI	BPTR	0086
0052	3752;		B4	*	0087
0054	3F54;		BN4	*	0088
0056	F800;		LDI	A.1(BLOC4)	0089
0058	B4;		PHI	BPTR	0090
0059	3759;		B4	*	0091
005B	3F5B;		BN4	*	0092
005D	62;		OUT	TVOF	0093
005E	F800;		LDI	A.1(TMP)	0094
0060	BC;		PHI	TMP3	0095
0061	F8A4;		LDI	A.0(TMP)	0096
0063	AC;		PLO	TMP3	0097
0064	F80F;		LDI	#0F	0098
0066	BF;		PHI	VIN	0099
0067	F8FF;	:	LDI	#FF	0100
0069	AF;		PLO	VIN	0101

```
ADDR CODE      LABEL   OPCODE  OPERAND  COMMENT

006A F807;     AGN:    LDI     #07      0102
006C AE;               PLO     TMP2     0103
006D F800;             LDI     #00      0104
006F 76;               SHRC             0105
0070 0F;               LDN     VIN      0106
0071 5C;               STR     TMP3     0107
0072 0C;       LOPV:   LDN     TMP3     0108
0073 76;               SHRC             0109
0074 3378;             BDF     ONES     0110
0076 3B86;             BNF     ZERS     0111
0078 7B;       ONES:   SEQ              0112
0079 5C;               STR     TMP3     0113
007A 2E;               DEC     TMP2     0114
007B 8E;               GLO     TMP2     0115
007C 3A72;             BNZ     LOPV     0116
007E 2F;               DEC     VIN      0117
007F 9F;               GHI     VIN      0118
0080 FB00;             XRI     #00      0119
0082 3A6A;             BNZ     AGN      0120
0084 3011;             BR      GRFX     0121
0086 7A;       ZERS:   REQ              0122
0087 5C;               STR     TMP3     0123
0088 2E;               DEC     TMP2     0124
0089 8E;               GLO     TMP2     0125
008A 3A72;             BNZ     LOPV     0126
008C 2F;               DEC     VIN      0127
008D 9F;               GHI     VIN      0128
008E FB00;             XRI     #00      0129
0090 3A6A;             BNZ     AGN      0130
0092 3011;             BR      GRFX     0131
0094 42;       EXIT:   LDA     SP       0132
0095 70;               RET              0133
0096 C4;       INTP:   NOP              0134
0097 22;               DEC     SP       0135
0098 78;               SAV              0136
0099 22;               DEC     SP       0137
009A 52;               STR     SP       0138
009B E2;               SEX     SP       0139
009C E2;               SEX     SP       0140
009D 94;               GHI     BPTR     0141
009E B0;               PHI     DMA      0142
009F F800;             LDI     #00      0143
00A1 A0;               PLO     DMA      0144
00A2 3094;             BR      EXIT     0145
00A4 00;       TMP:    ,0               0146
00A5 ;                 END              0147
0000
```

```
0000 F800 B1B2 B3B4 F896 A1F8 FFA2 F811 A3E2
0010 D369 3712 3F14 62F8 0FBF F8FF AF36 1D3E
0020 1FF8 00BE F808 AE36 2B3E 30F8 00FE 3033
0030 F880 FE0F 765F 2E8E 3A27 2F9F FB00 3A21
0040 F80C B469 3744 3F46 F808 B437 4B3F 4DF8
0050 04B4 3752 3F54 F800 B437 593F 5B62 F800
0060 BCF8 A4AC F80F BFF8 FFAF F807 AEF8 0076
0070 0F5C 0C76 3378 3B86 7B5C 2E8E 3A72 2F9F
0080 FB00 3A6A 3011 7A5C 2E8E 3A72 2F9F FB00
0090 3A6A 3011 4270 C422 7822 52E2 E294 B0F8
00A0 00A0 3094
```

A COOL DISPLAY

By William Carnes

This is a short program that uses Quest Super Basic. The program generates a screen full of asterisks, with one randomly located on each line. After four passes, the program lists and names itself before restarting. The program is unique in that no GOTOs are used in the loops. The RUN statement takes care of jumps.

NOTE: Before typing this in, type IMODE: DEFINT Z to avoid floating point errors.

```
10 IMODE:REM        This is a COOL display
20 FOR J=1 TO 4
30 CLS: FOR N=1 TO 15
40 PR TAB (RND(62));"*"
50 NEXT
60 WAIT(450): NEXT
70 CLS:LIST
80 WAIT (1500): RUN
```

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

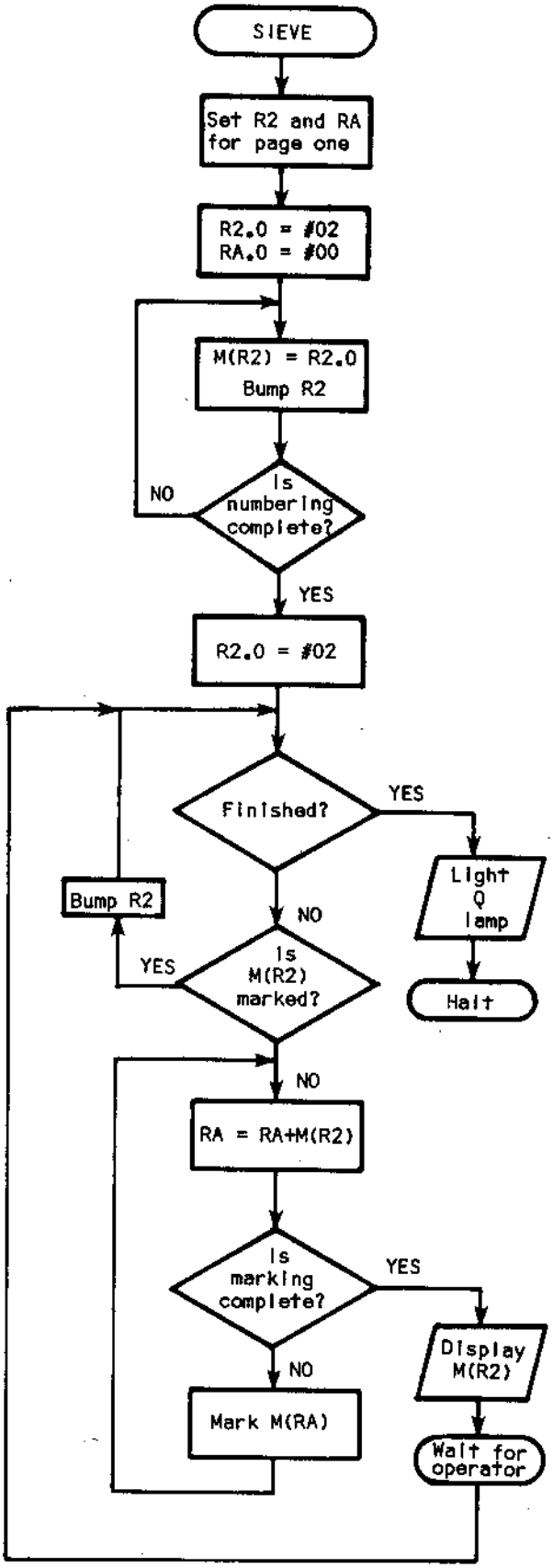
THE SIEVE OF ERATOSTHENES

by
Phillip B. Liescheski III

The Sieve of Eratosthenes is a method for obtaining a sequence of prime numbers. This algorithm is simple and one of the best methods for generating this sequence. There are other methods and formulas for performing this task, but they are more complex and do not produce a complete list, in other words, some prime numbers are left out. Eratosthenes of Kyrene who lived between 276 - 194 B.C. is the author of this algorithm. He inscribed upon a sheet a list of numbers starting with two. The non-prime numbers were cut out, thus leaving the prime numbers. The many holes in the sheet gave it the appearance of a sieve, thus this algorithm received its name.

Prime numbers are those numbers whose factors are itself and one (1). The number one is not considered a prime number; therefore, two is the first prime number whose only factors are one (1) and two (2). These numbers are important in the sense that all numbers can be represented as a product of them. For example, fifteen (15) may be represented as the product of five (5) and three (3) which are prime numbers. This fact is very handy when one must factor a natural number.

This classic algorithm is very simple to use and often taught in elementary arithmetic. To find the prime numbers between two and one-hundred (100), one must first produce a table or list of the natural numbers from two to one-hundred. Next, one begins with two and marks out every second number after it, thus removing all even numbers except two from the list. Now the next unmarked number which should be three is taken as the starting point, and every third number after it is marked out, thus removing the multiples of three from the list. The process is continued with the next unmarked number, and its multiples are removed, until no more numbers can be marked out. The unmarked numbers in the list are the prime numbers from two to one-hundred.



This method can be implemented by a computer such as the COSMAC 1802. Instead of using a sheet or paper, the list of numbers can be stored in a page of memory. With a page (256 bytes) of memory, one can obtain a list of prime numbers which are contained in the set of natural numbers which range from two to 255. In this 1802 machine-code program, the list is prepared on the first page of memory after the base page (locations: 0100-01FF). The program is contained in the base page (locations: 0000-00FF). First, the program prepares the page by generating the sequence of numbers between 02 and FF and storing them in the corresponding locations, in other words, 02 is stored at location 0102, 03 is stored at location 0103 and so on. After this, the program starts with 02 and stores a zero in every second location after 0102. The program marks out numbers by storing zeros at their locations. With this, the machine displays the number 02 on the hexadecimal readout and waits for the operator to push the input key for the next prime number. After the depression of the input key, the program searches for the next unmarked (non-zero) location. Zeros are stored at locations which are multiples of it, and this number is displayed. The process continues until all of the prime numbers that are contained in the page have been displayed, and the machine signals the operator that the process is complete by lighting its Q lamp.

This algorithm is quite slow and tedious for large lists of numbers, but since this program is in machine-code, the prime numbers are generated very quickly. Also, the prime numbers which are displayed are in hexadecimal notation, but this tends to give this ancient algorithm a contemporary twist. Finally, it should be stated that this program is designed for a 4K expanded Super Elf, but any COSMAC machine with at least two whole pages of memory should be able to execute this algorithm with slight modifications to the program's register initialization.

Bibliography

Gellert, W.; Kustner, H.; Hellwich, M.; and Kastner, H., ed. "The VNR Concise Encyclopedia of Mathematics". New York: Van Nostrand Reinhold Company, 1975.

Hutton, E.L. University of St. Thomas, Houston, Texas. Interview, 26 December 1979.

An Example Sieve

2	3	(4)	5
'(6)'	7	(8)	'9'
(10)	11	'(12)'	13
(14)	'15'	(16)	17

( )-Every second deletion

' '-Every third deletion

Unmarked numbers are the prime numbers from two to seventeen.

```
ADDR CODE            COMMENTS

0000 E2              Set X to 2
0001 F8 01           Set R2 and RA to point at page 1
0003 B2
0004 BA
0005 F8 02           Initialize R2
0007 A2
0008 52              Store the list of numbers in
0009 12              location pointed to by R2 and
                     bump R2
000A 82
000B 3A 08           Test if numbering is complete
000D 22              Numbering is complete
000E F8 02           Initialize R2
0010 A2
0011 92
0012 FB 01           Test if finished
0014 3A 2D
0016 F0              Not yet
0017 3A 1C           Find next unmarked location
                     (prime number)
0019 12
001A 30 11
001C F4              Calculate the unmarked location's
                     multiple
001D 33 26           Test if marking is complete
001F AA              Not yet; Push multiple in RA
0020 F8 00           Mark the location pointed at by
0022 5A              RA with a zero
0023 8A
0024 30 1C           Find the next location's multiple
0026 64              Display the prime number
0027 3F 27 37 29     Wait for the operator
002B 30 11           Do it again
002D 7B              Finished; Turn on the Q lamp
002E 00              Halt
```

```
0000 E2F8 01B2 BAF8 02A2 5212 823A 0822 F8B2
0010 A292 FB01 3A2D F03A 1C12 3011 F433 26AA
0020 F800 5A8A 301C 643F 2737 2930 117B 00
```

USING SUPER BASIC

by
Ron Cenker

SUPER BASIC is being enjoyed by many 1802 users today. It was written to be as compatible as possible with most popular BASIC's currently on the market. To a large extent this goal has been met. But certain features have been compromised in order to maintain reasonable speed performance. Most of these "exceptions to the rule" are really quite minor when looked at in more detail. This is a first in a limited series of articles to spell out differences between SUPER BASIC and the so called "standard" BASIC. The inherent differences in SUPER BASIC will first be listed. Examples of programs will be taken from commercial publications and any necessary modifications spelled out in more detail.

So what isn't "standard" in SUPER BASIC?

SUPER BASIC has only 26 variables (A-Z). Most BASIC's have 260 variables (A0-A9, B0-B9, ...Z0-Z9). The fact is that even the most complex BASIC programs can be written with 26 variables or less. In the unlikely event that more are needed a one-dimensional array can be set aside to get as many additional storage locations as possible. So when converting programs using two character variables into SUPER BASIC format, simply list all of the variables (I'll bet the list is less than 26 long) and re-assign them to single character variables. For example:

A0	A
A1	C
B	B
D1	D
D2	E

When entering the program one simply makes the necessary substitutions when a variable is encountered.

IF/THEN structures are slightly different in SUPER BASIC. IF A=B THEN 200 is a legitimate statement in most BASIC's. In fact, some BASIC's permit only this form of an IF/THEN. That is to say that the "THEN 200" is a branch to line number 200. SUPER BASIC would expect IF A=B GOTO 200 (more descriptive and with the same number of keystrokes). SUPER BASIC could also understand IF A=B THEN GOTO 200 but the "THEN" is unnecessary. In fact, any executable instruction could be used in place of the GOTO 200 including another conditional statement. When converting any BASIC program to SUPER BASIC one simply changes all lines of the following form:

IF A<B THEN 500

to:

IF A<B GOTO 500

Random number generation is probably the most disputed "standard" in BASIC. The RND function in SUPER BASIC can be either integer or floating point depending upon the existence of an argument as described in the SUPER BASIC users manual. It suffices to say that when a random number is to be used care should be taken to use the proper form of the function. In most cases where SUPER BASIC is in the full floating point mode, RND, as such, simply returns a floating point random number between 0 and 1. Therefore, 10*RND will return a floating point random number between 0 and 10. Furthermore, INT(10*RND) will return a floating point whole random number between 0 and 10.

Dimensioning of string arrays is unnecessary in SUPER BASIC. So when encountering a DIM statement for strings in any commercial program simply delete it to avoid any problems.

PRINT statements have different meanings assigned to the semi-colon and the comma. SUPER BASIC treats the semi-colon as a pure delimiter, i.e. the next outputted character will immediately follow the last with no spaces in a printed line. The comma will cause the next entry to be printed in the next eighth column increment. Either a semi-colon or a comma will inhibit a carriage return / line feed if used as the last character in a PRINT statement.

Some BASIC programs may permit premature exiting of a FOR/NEXT loop or a subroutine via a GOTO statement. This most often has the effect of walking down the working stack. SUPER BASIC provides an EXIT statement to be used in exactly the same way as a GOTO (an unconditional branch) to prematurely exit a FOR/NEXT loop or subroutine gracefully without disturbing the stack. Further explanation of the EXIT statement can be found in the SUPER BASIC users manual. Examples of its use in specific programs will be found in the next article.

A book entitled "Game Playing with Basic" by Donald D. Spencer, published by Hayden, is available at many bookstores. In it are many examples of simple programs and the logic that went into writing them. A small number guessing program appears on page 24 and is duplicated here as an example:

```
10 REM A NUMBER GUESSING GAME
15 PRINT "PLAYER 1 - GUESS IS";
20 INPUT P1
25 PRINT "PLAYER 2 - GUESS IS";
30 INPUT P2
35 LET C=INT(RND(1))+1
40 PRINT "COMPUTER SELECTED";C
45 IF ABS(C-P1) <> ABSC(C-P2) THEN 60
50 PRINT "BOTH PLAYERS WERE EQUAL"
55 GOTO 80
60 IF ABS(C-P1) < ABS(C-P2) THEN 75
65 PRINT "PLAYER 2 WAS CLOSEST"
70 GOTO 80
75 PRINT "PLAYER 1 WAS CLOSEST"
80 END
```

The first modification found necessary to run this program in SUPER BASIC is to rename the variables since it makes use of two character variables:

Variables used	Variables assigned
P1	A
P2	B
C	C

Next it should be recognized that line number 45 and 60 must be changed in such a way as to replace the "THEN" with "GOTO". Finally, line #35 generates a random number between 1 and 50. Consideration must be given to the method of generating that random sequence of numbers. Assuming that SUPER BASIC will be in the powered up, full floating point mode, the following will represent the modified program:

```
10 REM A NUMBER GUESSING GAME
15 PRINT "PLAYER 1 - GUESS IS";
20 INPUT A
25 PRINT "PLAYER 2 - GUESS IS";
30 INPUT B
35 LET C=INT(50*RND)+1
40 PRINT "COMPUTER SELECTED ";C
45 IF ABS(C-A) <> ABS(C-B) GOTO 60
50 PRINT "BOTH PLAYERS WERE EQUAL"
55 GOTO 80
60 IF ABS(C-A) < ABS(C-B) GOTO 75
65 PRINT "PLAYER 2 WAS CLOSEST"
70 GOTO 80
75 PRINT "PLAYER 1 WAS CLOSEST"
80 END
```

Note also that in line #40 a space is inserted after the word SELECTED since the value of C will be printed directly after the preceeding string enclosed in quotes. The following is an example of the program execution:

```
RUN

PLAYER 1 - GUESS IS?6
PLAYER 2 - GUESS IS?36
COMPUTER SELECTED 9.
PLAYER 1 WAS CLOSEST
```

Note that when line #40 was executed, a floating point nine (9.) was printed. If the decimal is to be inhibited, the C in line #40 can be replaced by INUM(C).

If this same program were to be executed in the integer mode in the interest of improved speed or accuracy (this example requires neither more speed nor accuracy) the following changes would be made:

```
Add line #5          5 DEFINT Z
Change line #35     35 LET C=1+RND(50)
```

Finally, SUPER BASIC would allow this program to be typed in a condensed format as follows:

```
5  DEFINT Z
10 ! A NUMBER GUESSING GAME
15 INPUT"PLAYER 1 - GUESS IS";A
25 INPUT"PLAYER 2 - GUESS IS";B
35 C=1+RND(50)
40 PR "COMPUTER SELECTED ";C
45 IF ABS(C-A)A=ABS(C-B) PR"BOTH PLAYERS WERE
   EQUAL":END
60 IF ABS(C-A)<ABS(C-B) PR"PLAYER 1 WAS
   CLOSEST":END
65 PR"PLAYER 2 WAS CLOSEST"
```

Note the ability to use conditional "statements", concatenated commands on the same line, commented INPUT statements, imbedded END statements, and an implied END at the end of the program.

The above example demonstrates the majority of changes which must be made to run almost any BASIC program on SUPER BASIC. These programs might be found in any of the computer magazines or BASIC game-playing books. The future issues of Questdata will give more examples of more complex programs found in some of the commercial publications.

HARMONIOUS SEQUENCER

By Don Stevens

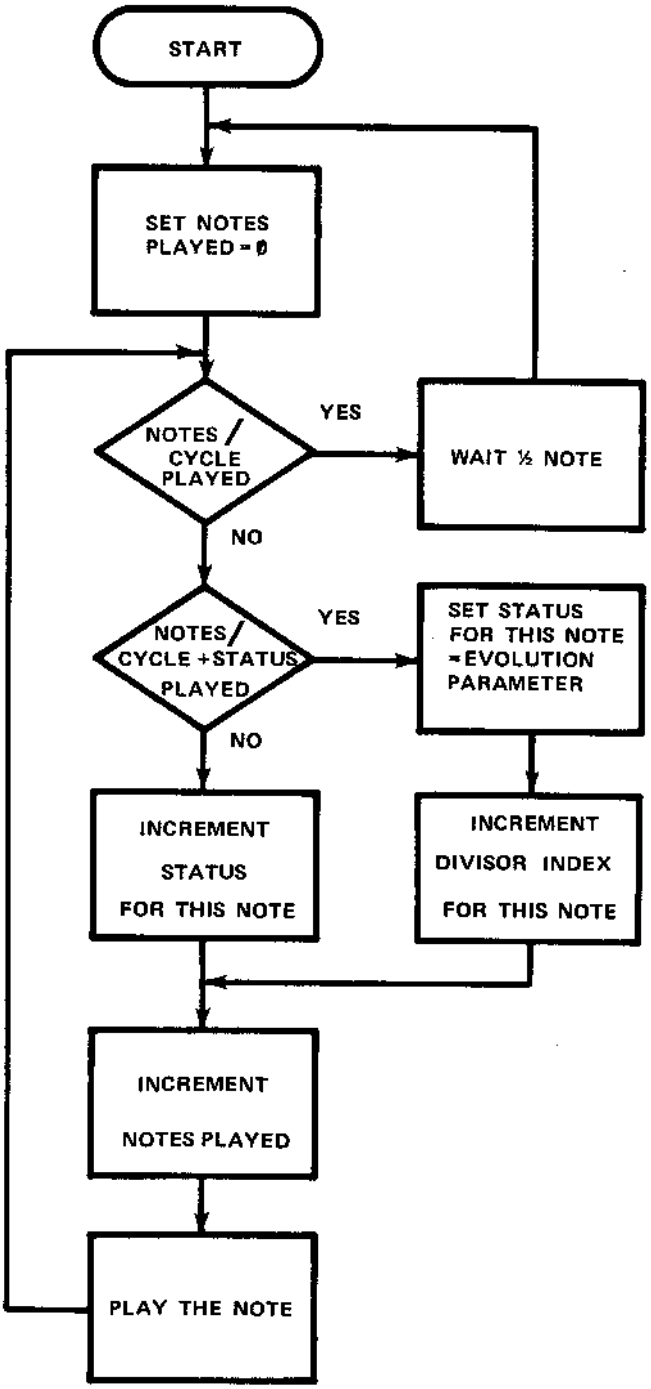
This harmonious sequencer program plays sequences of harmonically related tones, and it was inspired by Bob Richie's Sounder (issue #5), Ed McCormic's algorithm (issue #7), and Paul Moews' algorithm (issue #10). The sequences evolve gradually or rapidly depending on the choice of the evolution parameter (location 65). All sequences have the same name number of notes (determined by location 26) which can be chosen to be from 1 to 8. All the notes are played the same length of time, which is determined by the tempo parameter (location 15). The suggested value (38) gives notes about 1 second long.

Music which sounds somewhat like Telemann recorder music is produced with evolution=FF (LOC 0065) notes=03, (LOC 0026) and tempo=0C, (LOC 0015).

The frequencies of the notes are determined by a table of divisors (located from 16 to 25), with the frequency produced=3579545/(64* divisior). The divisors suggested in the listing are chosen so that the tones produced are all harmonics of 77.68Hz.; respectively the 3, 4, 16, 5, 6, 8, 4, 9, 10, 12, 15, 16, 8, 18, 20, 24th harmoics. Another selection of divisors is suggested for musical experimenters, namely: D2, A8, 3C, 8C, 78, 69, 54, 46, 1E, 3C, 38, 2A, 28, 78, 23, 1E. These yield, respectively, the 4, 5, 14, 6, 7, 8, 10, 12, 28, 14, 15, 20, 21, 7, 24, 28th harmonics of 66.58Hz. I find the 7th harmonics somewhat disharmonious.

The pattern of sequences will eventually repeat, the period is more than 3 days for sequences of length 4 and more than 100 years for sequences of length 8. Particular sequences will repeat much more often, of course.

The program works as follows. A status value is kept for each note position in the cycle. For simplicity suppose there are 4 notes per cycle and now we consider the 3rd note in the cycle. Each time this note is to be played, its status value is checked to see whether it is equal to 1 (=4-3); if not, the status value is incremented and the third divisor index determines the divisor used for this note. If the status value were =1, the status value is set equal to the evolution parameter, the 3rd divisor index is incremented, and the note played. If the divisor for a note were say 24, then exactly 36 (=24 hex) instructions are executed between succesive reversals of Q.



HARMONIOUS SEQUENCER

```
ADDR CODE    LABEL   OPCODE OPERAND COMMENT

0000 F8              NOP
0001 00              NOP
0002 B3              NOP
0003 93              GHI 3
0004 B4              GHI 0
0005 B8              PHI 8
0006 B9              PHI 9
0007 BA              PHI A
0008 BC              PHI C
0009 BD              PHI D
000A F827            LDI    (L4)
000C A4              PLO 4          R4 Points to
                                    Scratch
000D F826            LDI    (L3)
000F A9              PLO 9          R9 Points to
                                    notes played
0010 F815            LDI    (L1)
0012 AA              PLO A          RA Points to
                                    Tempo
0013 3038            BR     (L7)    GOTO (L7)
0015 38      (L1)                   tempo
0016 F0      (L2)                   divisor 1
0017 B4                             divisor 2
0018 2D                             divisor 3
0019 90                             divisor 4
001A 78                             divisor 5
001B 5A                             divisor 6
001C B4                             divisor 7
001D 50                             divisor 8
001E 48                             divisor 9
001F 3C                             divisor 10
0020 30                             divisor 11
0021 2D                             divisor 12
0022 5A                             divisor 13
0023 28                             divisor 14
0024 24                             divisor 15
0025 1E                             divisor 16
0026 04      (L3)                   notes per cycle
0027 00      (L4)                   Scratch
0028 01      (L5)                   divisor index for
                                    position 1
0029 02                             divisor index for
                                    position 2
002A 03                             divisor index for
                                    position 3
002B 04                             divisor index for
                                    position 4
002C 05                             divisor index for
                                    position 5
002D 06                             divisor index for
                                    position 6
002E 07                             divisor index for
                                    position 7
002F 08                             divisor index for
                                    position 8
0030 00      (L6)                   position 1 status
0031 00                             position 2 status
0032 00                             position 3 status
0033 00                             position 4 status
0034 00                             position 5 status
0035 00                             position 6 status
0036 00                             position 7 status
0037 00                             position 8 status
0038 F800            LDI    X'00'   Begin a new cycle
003A A5              PLO 5          R5.0=Notes Played
                                    =0
003B 0A      (L8)    LDN A          Continue cycle,D=
                                    tempo
003C BB              PHI B          RB.1=tempo
003D F8FF            LDI    X'FF'
003F AB              PLO B          RB.0=FF
0040 85              GLO 5          D=Notes played
0041 E9              SEX 9
0042 F3              XOR
0043 3A4D            BNZ    (L10)   If(notes played do
                                    not equal notes
                                    per cycle)GOTO (L10)
0045 9B              GHI B          D=tempo
0046 BB              PHI B          RB.1=tempo
0047 2B      (L9)    DEC 2          LOOP
0048 9B              GHI B
0049 3A47            BNZ    (L9)    Until RB.1=0
                                    (Half Note)
004B 3038            BR     (L7)    GOTO (L7)
004D 85      (L10)   GLO 5          D=Notes Played
004E FC30            ADI    (L6)    D=(L6)+Notes Played
0050 AC              PLO C          RC points to
                                    status
0051 85              GLO 5          D=Notes Played
0052 FC28            ADI    (L5)    D=D+Notes Played
0054 AD              PLO D          RD Points to
                                    divisor index
0055 0D              LDN 0          D=divisor index
0056 B5              PHI 5          R5.1= divisor
                                    index
0057 85              GLO 5          D=Notes Played
0058 E9              SEX 9
0059 F5              SD             D=Number of notes
                                    left
005A EC              SEX C
005B F3              XOR
005C 3264            BZ     (L11)   If(num notes left=
                                    status) GOTO(L11)
005E 0C              LDN C          otherwise,
005F FC01            ADI    01      Increment status
0061 5C              STR C          Store Incremented
                                    status
0062 306D            BR     (L13)
0064 F8FF    (L11)   LDI    X'FF'   Status has reached
                                    max evolution
                                    parameter
0066 5C              STR C          Status=parameter
0067 95              GHI 5          D=Divisor index
0068 FC01            ADI    01      Increment divisor
                                    index
006A FA0F            ANI    X'0F'   Restrict range to
                                    00 to 0F
006C 5D              STR D          Store new divisor
                                    index
006D 85      (L13)   GLO 5          D=Notes Played
006E FC01            ADI    01      Increment Notes
                                    Played
0070 A5              PLO 5          Save Incremented
                                    value
0071 95              GHI 5          D=divisor index
0072 FC16            ADI    (L2)
0074 A8              PLO 8          R8 points to
                                    divisor
0075 08              LDN 8          D=divisor
0076 54              STR 4          Scratch=divisor
0077 E4              SEX 4
0078 64              OUT 4          Display divisor
0079 24              DEC 4
007A 7B              SEQ            NUM.(instr. till
                                    req.)=Divisor
```

```
ADDR CODE     LABEL   OPCODE OPERAND COMMENT
007B 08               LDN 8          D=Divisor
007C FF05             SMI    05      D=Divisor-5
007E F6               SHR            D=Loops to do
007F 3B82             BNF    (L15)   Skip instr. if
                                     DF = 0
0081 E4               SEX 4
0082 FF01     (L15)   SMI    01      LOOP
0084 3A82             BNZ    (L15)   Until D=0
0086 7A               REQ            NUM. (instr. till
                                     end)=Divisor
0087 08               LDN            D=DIVISOR
0088 F6               SHR
0089 F6               SHR            D=Divisor/4
008A E4               SEX
008B 54               STR            Scratch=Divisor
008C 8B               GLO B          D=Low tempo count
008D F7               SM             D=D-Scratch
008E AB               PLO B          Low tempo count
                                     decremented
008F 9B               GHI B          D=high tempo count
0090 7F00             SMBI   00      D=D-Borrow
0092 BB               PHI            high tempo count
                                     decremented
0093 323B             BZ     (L8)    If note done,
                                     GOTO (L8)
0095 08               LDN            D=Divisor
0096 FF12             SMI    X'12'   D=Divisor-18
0098 F6               SHR            D=Loops to do
0099 3B9C             BNF    (L16)   Skip instr. if
                                     DF=0
009B E4               SEX 4
009C FF01     (L16)   SMI    01      LOOP
009E 3A9C             BNZ    (L16)   Until D=0
00A0 307A             BR     (L14)   Continue note, GOTO
                                     (L14)
```

```
0000 F800 B393 B4B8 B9BA BCBD F827 A4F8 26A9
0010 F815 AA30 3838 F0B4 2D90 785A B450 483C
0020 302D 5A28 241E 0400 0102 0304 0506 0708
0030 0000 0000 0000 0000 F800 A50A BBF8 FFAB
0040 85E9 F33A 4D9B BB2B 9B3A 4730 3885 FC30
0050 AC85 FC28 AD0D B585 E9F5 ECF3 3264 0CFC
0060 015C 306D F8FF 5C95 FC01 FA0F 5D85 FC01
0070 A595 FC16 A808 54E4 6424 7B08 FF05 F63B
0080 82E4 FF01 3A82 7A08 F6F6 E454 8BF7 AB9B
0090 7F00 BB32 3B08 FF12 F63B 9CE4 FF01 3A9C
00A0 307A
```

BCD TO BINARY CONVERTER

by Al Williams

This program will convert a BCD (binary coded decimal) number to hex on a basic elf or other 1802 machine, using the same I/O. The program masks the least significant digit (LSD) in the D register and stores it in the stack then shifts the most significant digit (MSD) to the right two times. The result is stored in the stack then shifted twice to the right again. That is added to the last entry in the stack, shifted left once and added to the first stack entry. Now the D register contains the binary equivalent of the decimal input. This is more easily done than said.

To use the program, enter a decimal number (00-99) and run the program. The hex number will be displayed on the LED's. To convert another number, enter the number using the Input key.

```
ADDR CODE                    COMMENT

0000 90 B2                   ; R2.1=00
0002 F8 FF A2                ; R2.0=00
0005 E2                      ; Set stack
                               pointer to R2
0006 6C BF                   ; Input to D and
                               RF.1
0008 FA 0F                   ; Select LSD
000A 73                      ; Push D to stack
000B 9F                      ; Restore original
                               input
000C FA F0                   ; Select MSD
000E 76 76                   ; Shift right
                               twice
0010 73                      ; Push D to stack
0011 76 76                   ; Shift right
                               twice
0013 60 F4                   ; D + stack
0015 7E                      ; Shift left
0015 60 F4                   ; D + stack
0018 52                      ; Stack = D
0019 64 22                   ; Output and
                               correct R2
001B 3F 1B                   ; Wait
001D 37 1D                   ; "
001F 30 06                   ; Go back to top
```

Program Listing

Decimal	Hex
10	0A
50	32
80	50
99	63

```
0000 90B2 F8FF A2E2 6CBF FA0F 739F FAF0 7676
0010 7376 7660 F47E 60F4 5264 223F 1B37 1D30
0020 06
```

15 PUZZLE FOR THE ELF II

by
Robert V. Dipippo

The listing for the 15 Puzzle by Ray Tully Vol. 2 Issue #1 must be modified for use with the ELF II.

The changes that I have made do not require any additional memory. It is necessary to set up the registers before entering the program. To do this I used the unused locations 0083 to 0089 to set the registers, and locations 01F0 to 01F5 for the routine to turn on the TV.

I hope this little change will make more 1802 owners enjoy this truly exciting game.

```
ADDR CODE    LABEL   OPCODE OPERAND COMMENT
0083 F8              LDI
0084 01
0085 B3              PHI R3.1
0086 F8              LDI
0087 03
0088 A3              PLO R3.0
0089 D3              SEP R3  Go to beg. of pgm.

01F0 E2              SEX R2
01F1 69                      Turn on TV
01F2 0B
01F3 5F
01F4 30
01F5 4D                      Jump back display
                             square
```

Change the following locations in the program to jump to the TV on routine.

```
014B 30
014C F0
```

ELF-II PATCHES FOR "TB TTY IF"

By Chuck Reid

I must thank Questdata for Issue #12. The article "Tiny Basic to Teletype Interface" has come to my rescue. I've been trying to address my serial ASCII keyboard (Netronics' Terminal) on my ELF-II with minimal success. But now thanks to Questdata I have succeeded and here are the "mods" as they fit on an ELF-II.

Serial I/O on a Netronics Terminal operates on Sense Line EF4 and uses reverse logic to that listed in Questdata Issue #12 pages 3 to 8. Change the following single bytes:

ADDR	DATA	COMMENT
0932	3F	Note: Data is "35" for the Super Elf
0990	3F	
09A5	3F	
09AF	3F	
09B9	3F	
09C9	3F	
0940	37	Note: Data is "3D" for the Super Elf
0994	37	
09AA	37	
09B4	37	

[EDITOR'S NOTE: We are printing these patches without verification in hopes that other ELF-II owners can find them useful. We understand that a delay is required for the Clear Screen command to the video board. These routines do not accommodate the delay so it will have to be supplied by the using routine.

Notes From The Publisher

Resistance to change has been the nemesis of individuals and companies alike, particularly in the young, fast-growing electronic industry.. Most companies that know what they are doing in fact know that change is inevitable, and it is only a question of who, where, when or how to make the best of it. In this context, I am reluctant to announce that our Editor, Bill Haslacher, has decided that it is time for him to move on to new and more challenging avenues of pursuit. In his new position with Atari, he will be involved in technical editing and software with some creative writing. Although I can't imagine anything more challenging or demanding than editing a software newsletter single-handedly, we wish Bill the best of luck and success and know that he will contribute greatly to whatever he chooses to do, as he has done for Questdata. We do hear through the grapevine, however, that in spite of his new job there is no way that Bill will give up his hobby and love for the 1802 in all its vagaries. He is a machine language fanatic to the end. Happy Cosmacing!

In the same breath, we are pleased to announce that Paul Messinger has been named the new managing editor of Questdata. Paul has been an avid hobbyist for years who has extensive background in both hardware design and software. He brings intimate knowledge of the 1802 in all its forms to the job. Because Paul has additional responsibilities, we will have Associate Editors who will help Paul check out programs in addition to providing some creative writing, to enable us to get the issues out more on schedule. He has committed quality on a timely basis. We will fully support him in meeting these goals.

Speaking of support, we continue to appreciate your growing acceptance of Questdata both in subscriptions and in new material. There is a tremendous reservoir of talent in the ranks of our readership and we are continually amazed at the quality of programming being accomplished on a hobby basis. Keep up the good work and thank you for your efforts.

P.S. We need more short programs that can be run on 256 bytes or less on any subject.

ADDRESS CORRECTION REQUESTED

BULK RATE
U.S. Postage Paid
QUEST
Electronics

Permit No. 649
Santa Clara, CA