Questdata

Volume 1 Issue #12 ©

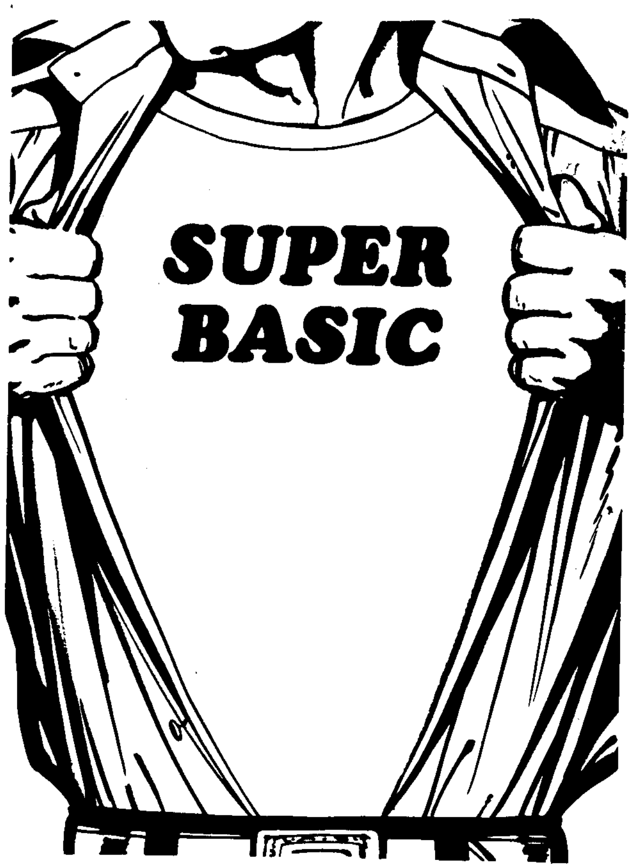
THE JOY OF FULL BASIC

AT LAST! Full size Quest Super BASIC has arrived for the 1802 microcomputer. Quest is the first COSMAC supplier to actually deliver Full BASIC software. Field testing of the Quest Full BASIC actually began last summer. This extensive testing has paid off in many letters of praise from happy users of the released production version. Now COSMACians everywhere can have strings, floating point, many built in functions and many other fine features. With both fixed and floating point routines available, you can choose either the speed and accuracy of fixed point or the wide range to be found through the exponentiation of floating point. To be exact about it, fixed point can handle decimal numbers -2,147,483,648 to +2,147,483,647 and floating point allows -.17E38 to +.17E38 . . . that is an exponent range of -38 to +38 with an accuracy of greater than 5 significant figures.

String and array space is automatically allocated as you need it and the number of variables you can use in Quest BASIC is essentially unlimited by use of the string variable capability. Caution messages tell you when you start using the last page of available memory in your system.

Quest BASIC includes the I/O drivers to work on Elf II and other 1802 systems in both serial and parallel modes. However, Full BASIC does require that you have RAM starting at location 0000. Full BASIC comes with serial I/O at 110-3000 BAUD (with automatic BAUD rate) and full or half duplex. Parallel I/O is provided for use with keyboard and memory mapped video display. By addressing an S-100 video board to location E000, you will be able to see the BASIC program as you write and run it on your monitor or TV with RF modulator. Elf II owners with their ASCII keyboard and video board can use the BASIC by following the detailed patching instructions included with the documentation. The instructions include the requirements necessary to patch in your own I/O routines.

You will need a total of 12K of memory to get started with the fun and convenience of this very complete language. Yes, it's big but it's because it is packed with features not usually found on other "Full" BASICs.



The Quest Super Full BASIC compacts your programs as it stores them in free RAM locations. What this means is that your Full BASIC programs will take up a lot less space than an interpreter which just scans what is input without modifying it for compactness. For example, the word PRINT can be represented as just a one byte code in memory. When you want to list the program on the video or TTY, the program will unpack (stretch back into original form) for human consumption. This means that your Full BASIC programs will run faster and take up less space than BASICs without this feature. Also, there is no intermediate Interpreter code used in the program like that found in Tiny BASIC, so your programs get another boost in processing speed.

When you execute CSAVE and CLOAD, the compact memory size will translate itself into faster loading and unloading of programs. CSAVE and CLOAD allow you to use a single command to either automatically save or load:

- BASIC programs
- Data
- Machine programs

Cassette SAVE and LOAD functions use the very reliable and ultra simple to use Quest cassette software. It is as simple as setting the volume to max., rewinding to start (it is not necessary to advance past the leader) and go. Full BASIC recordings are assured of being valid when you are using one of the many Quest recommended cassette recorders.

Never again will you have to turn your tape recorder on and off by crudely pressing manual buttons. Just set one recorder on PLAY and the other on RECORD and you can leave the controls to BASIC. This requires 2 recorders and the use of optional relays connected to the parallel output port. BASIC then automatically controls both recorders. If you wish to keep things as they are with one tape recorder, nobody will stop you. Your friends might call you Neanderthal and other names but it's a free world.

Don't worry if you have a homebrew or other system, Quest will supply you with the READ code from the Quest Monitor and the associated circuit schematic. Elf II cassette hardware will work as is.

Registered Super BASIC owners will receive periodic information about language updates and useful BASIC information.

As nearly as possible, Ron Cenker (the author of Full BASIC), has adhered to the conventions of the most popular BASICs around. This means almost any book you have with listings of BASIC programs will be easy to enter into your Full BASIC speaking COSMAC. We have watched patiently as others entered Radio Shack books, hobbyist magazine programs, and other listings directly into their computers. Surely we have waited long enough. Now we have the joys of Full BASIC.

Quest Full BASIC has over 75 commands including arrays and string manipulation functions not found on many "Full" BASICs. Here is a sample program which draws on a few of the many features of the language. It will draw a SINE wave for you (put dates next to it and you can call it a biorhythm if you wish). Shows you the FOR NEXT and TAB and SIN. I hope it gives you a feeling for what is in store with Quest's Super BASIC.

```
 10 FOR X=-1000 TO 1000 STEP 20
 20 PRINT TAB(15+10*SIN(X));"*"
 30 NEXT X
 40 END
RUN
```

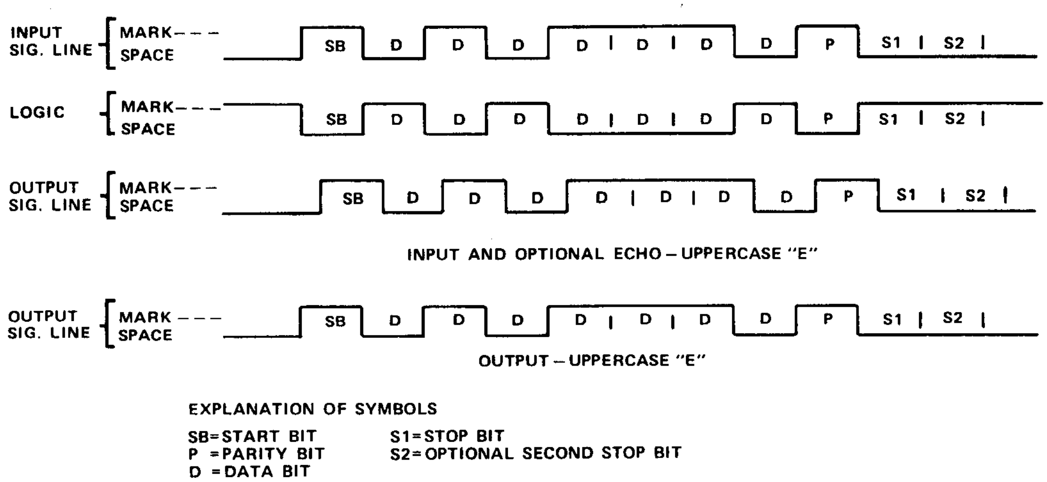


FIGURE 1 SERIAL INTERFACE WAVEFORMS

(WAVEFORM FOR SERIAL I/O ARTICLE ON NEXT PAGE)

TINY BASIC TO TELETYPE INTERFACE

This article describes a set of routines for the Super Elf which will interface Tiny BASIC to a teletype machine or other serial interface output. Tiny BASIC can be initialized for COLD or WARM start using these routines. All input and output communication and patches are also provided. The serial interface can be either RS232 or 20 ma. current loop. The "SPACE" condition on the input line to the Super Elf is a low (0 Volts) on the sense line 2. This gives an EF2=1 state. Conversely, the "MARK" condition on the input line to the Super Elf is a high (+5 Volts) on sense line 2. This is an EF2=0 state. The output line from the Super Elf is the Q-line and a "SPACE" condition is Q-line low (Q=OFF). A "MARK" condition is Q-line high (Q=ON).

The BAUD rate of the terminal connected to the serial interface is automatically computed by the INIT subroutine. For the 20 ma. current loop the rate is usually limited to a maximum of 300 BAUD by the terminal but the RS232 interface will handle from well below 110 BAUD to 2400 BAUD and is calculated workable up to rates of 3430 BAUD. These rates are computed for a clock rate of 3.58/2 megahertz. Any other clock rate adjusts the upper limit proportionately. For example, if a clock rate of 5 megahertz is used (instead of 3.58/2 as used on the Super Elf), then using the simple ratio:

$$\frac{3.58/2}{3430} = \frac{5}{X}$$

X=9580 BAUD will be the upper limit of the BAUD rate.

The INIT routine also provides for two stop bits for BAUD rates of 125 BAUD or less and one stop bit for rates over 125 BAUD: In addition, the mechanization of BAUD rate requires that you input a CR or "M". By selection either a carriage return (CR) or the character "M", you have a choice of specifying full duplex or half duplex modes, respectively.

The input and output routines handle the NULL character (hex code "00") uniquely. The input routine loops for the next character, i.e. it does not return a NULL character ever to the caller program. The output routine, on the other hand, will automatically transmit 7 NULL's upon receipt of a Line Feed (LF), (hex code "0A").

The Figure shows the waveforms on the signal line for input and output. The waveforms starting at the left represent the rise and fall of the interface signals as time progresses. The interface is characterized by a start bit and one or two stop bits. Each bit is timed to be 1/BAUD RATE in duration. So a bit time for 110 BAUD is 9.09 milliseconds. The data bits are transmitted (or received) "backwards" and upside down as shown on the Figure. To illustrate, the first bit after the START BIT is a SPACE but a logical 1, the second bit is a MARK but a logical 0, etc. If you line up the logical bits as: 1010001 and reverse the order thus: 1000101 the pattern suggests a hexadecimal "45", which is the hex code for uppercase "E".

HOW TO USE THE SERIAL I/O DRIVER PACKAGE

Load the program into your Super Elf from the hexadecimal listing (Figure 2). The range of addresses is locations 0900 through 09FF, but you can load the program in any high address (page) of memory that is convenient as long as you don't change the low-addresses. To use the driver routines with Tiny BASIC version 3.1, load exactly as given, it is already set up to run with some modifications to Tiny BASIC. Load Tiny BASIC as usual but change the following locations:

LOC.	DATA	REASON
0106-08	C00900	Connects Tiny to Input
0109-0B	C00903	Connects Tiny to Output
010C-0E	C00906	Connects Tiny to Test Break
011C-1D	0A00	To set the user program space lower bound.

To initiate the routines, use the Super Monitor option 00 or some other means (such as LBR at location 0000) to branch into location 09D1.

To use the serial routines with Tiny BASIC version 1.1 on ROM 8400-8BFF, load the routines into location 0100 to 01FF. The interfacing requires only a change to locations 01F7-F8 and 01FC-FD as follows:

LOC.	DATA	REASON
01F7-F8	8400	For COLD Start
01FC-FD	8403	For WARM Start

To initiate the routines, use a LBR at location 0000 to jump to locations 01D1, i.e. load C001D1 at location 0000 to 0002. Hit Reset and Go.

Once the routines are initiated, the system now waits for a character input from the keyboard. Only one of two inputs are valid, and any others will cause erroneous operation and you MUST start over from Reset. The characters, Carriage Return (CR) or the UPPER CASE "M", are the only valid inputs at this point. The Carriage Return (CR) means you have selected full duplex (or input character echo) and "M" means you want half-duplex operation (no echo). During the input phase, the INIT routine will compute the BAUD rate of your terminal and determine if one or two stop bits are required for echo of input or for output. INIT is used only once.

Having input either a (CR) or "M". you now enter either a "C" or "W". The "C" is for COLD start of Tiny BASIC, the "W" is for Warm start. Please refer to the Tiny BASIC manual for more information about Tiny BASIC. At this point, any input is OK but only a "C" or "W" will get you into Tiny BASIC. The program looks (loops, actually) for a "C" or "W" and when this character is received it will go into the Tiny BASIC program. When the desired start to Tiny BASIC is received, the COLON (:) prompt from Tiny BASIC will appear, indicating that Tiny BASIC is ready.

HOW TO USE THE ROUTINES

The Serial Interface routines consist of several routines, Table 1 gives their names, functions, addresses and entry points.

TABLE 1

NAME	FUNCTION	LOC.	ENTRY
INIT	Sets up BAUD rate	0997-09D0	0997
INPUT	Reads and interprets serial input line from terminal	092F-095B	092F
OUTPUT	Writes to serial output line to terminal	095C-098D	095C
TEST BREAK	Performs Test Break function for Tiny BASIC	098E-0996	098E
TINY INTERFACE	Provides complete start Tiny interface	09D1-09FF	09D1
DELAY	Provides timing for input and output routines	0909-092E	010B

These routines are supplied as a working set to provide the function as described above in the "How to Use the Serial Package." With your own "custom designed" interfacing, you can use these routines in a like manner or for output routines alone. The routines in Table 1 are designed to work with the Standard Call and Return (SCRT) technique explained in the RCA user manual for the 1802 (MPM-201, except the DELAY routine which is slaved to the input and output routines).

INIT. To use the INIT routine, load the routine and load the DELAY routine. The Table 1 addresses are 0997-09D0 for INIT and 0909-092E for DELAY. With a Program Counter (PC) of Register 3, and a Stack pointer of Register 2 (SEX 2) pointing to one free byte of RAM, branch to the entry address given in the table. The INIT routine loads the address of the DELAY subroutine into register C. INIT then waits for the input character (CR) or "M" as previously described. Once the input is received, the BAUD rate is calculated and stored in Register E. This register and register C must be reserved for the DELAY routine (NOT AVAILABLE without program modification—DON'T use Registers C and E). Two flags are set in the upper two bits of Register E for echo/no-echo and for 1 or 2 stop bits. The uppermost bit is set to 1 for echo, the second uppermost bit is set to a 1 for 1 stop bits. The criteria for setting 1 versus 2 stop bits is the comparison of the derived BAUD rate against a two byte constant stored at location 09BE (low byte) and 09C1 (high byte). This constant (00D6) is set for 125 BAUD at a clock rate of 3.58/2 MHz. If your clock rate is different than 3.58/2 MHz., then the constant must be recomputed for your use. Use this formula:

$$\text{CONSTANT} = \frac{\text{CLOCK FREQ. (Hz)}}{64 \times 125} - 9.$$

Plug in your CLOCK FREQ. and compute the CONSTANT. Convert to hexadecimal and store the low byte in location 09BE and the high byte into location 09C1.

The INIT routine then exits via a SEP 5 (standard SCRT convention).

INPUT. To use the INPUT routine, load the routine and load the DELAY routine. Either use the INIT routine for BAUD rate determination, or you must provide a similar routine on your own. To do it yourself you will need to:

(1) Load the entry address of the DELAY routine into Register C.

(2) With the BAUD rate you desire and the clock frequency in Hz. of your computer, solve this equation:

$$\text{TLC} = \frac{\text{CLOCK FREQ. (Hz)}}{64 \times \text{BAUD RATE}} - 9$$

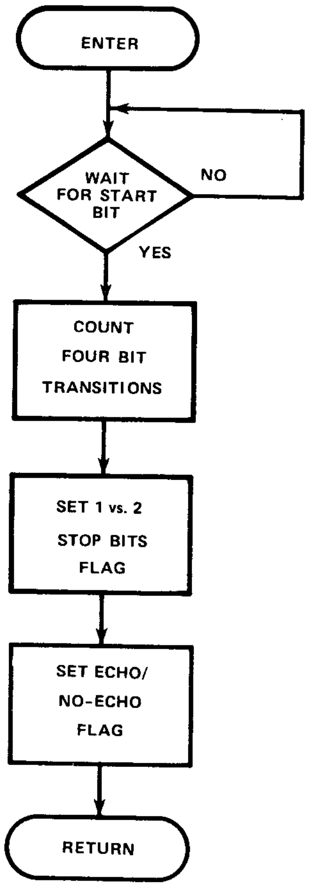
Convert the answer to hexadecimal, for loading into Register E. Set the upper bits of Register E for 1 or 2 Stop Bits and ECHO/NO–ECHO, see INIT routine description above. (For example, for BAUD=110, FREQ=1790000Hz then TLC=245.26, in hex =F5. Thus, Register E=00F5. For 2 Stop Bits and Full Echo adjust the final Register E value to C0F5).

Branch to the INPUT routine at the entry address given in Table 1. Have R3 as PC and X=2. The routine will then wait for the input character, assemble it and return via a SEP to R5. The data character is in the Data Register and high byte of Register F.

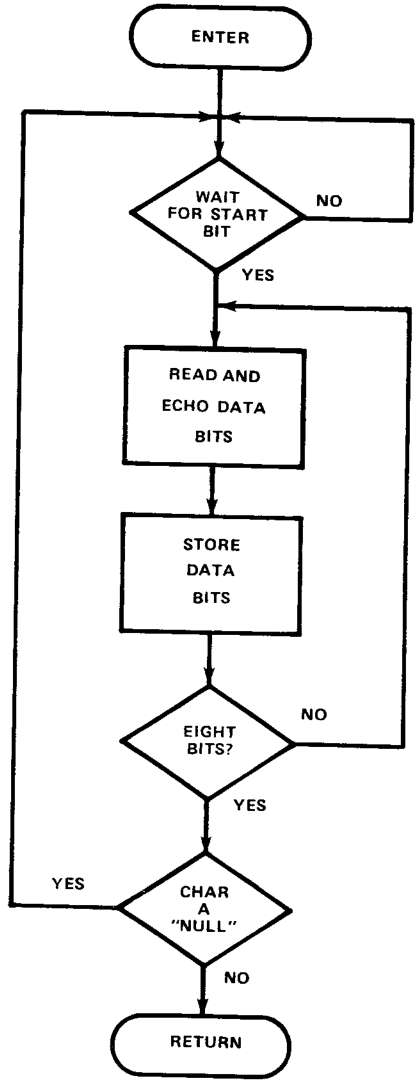
OUTPUT. To use the output routine, load the routine and load the DELAY routine. Either use the INIT routine for BAUD rate determination, or you'll have to provide the same service it does yourself. The same techniques (1) and (2) of the INPUT routine are used to construct your own OUTPUT routine.

SERIAL I/O FLOWCHART

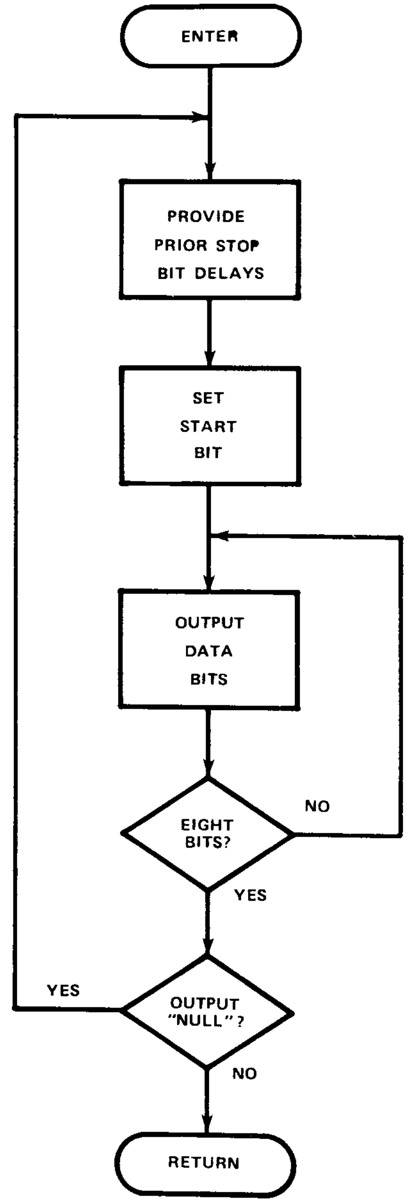
INITIALIZATION ROUTINE



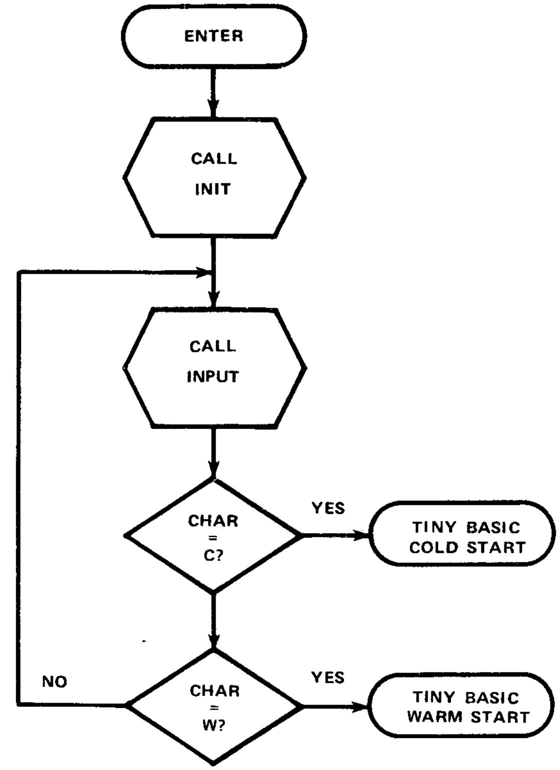
INPUT ROUTINE



OUTPUT ROUTINE



TINY BASIC INTERFACE



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

REGISTER ASSIGNMENTS

REGISTER	SYMBOL	USE
0	–	–
1	–	–
2	ST	STACK
3	PC	PROGRAM COUNTER
4	–	–
5	R5	RETURN REGISTER
6	–	–
7	–	–
8	–	–
9	–	–
A	–	–
B	–	–
C	RC	DELAY ROUTINE PROGRAM COUNTER
D	LC	DELAY CONSTANT & FLAGS
F.1	CHASSY	CHARACTER ASSEMBLY (INPUT)
	ACHOUT	OUTPUT CHAR. HOLDER (OUTPUT)
F.0	CNTR	NULL OUTPUT COUNTER
F	TLC	INTERVAL COUNTER (INIT)

LOC.	CODE	MNEM.	ACTION
0900	302F00	BR X'302F'	VECTOR TO INPUT
0903	305C00	BR X'305C'	VECTOR TO OUTPUT
0906	308E00	BR X'308E'	VECTOR TO TEST BRAKE

DELAY ROUTINE

LOC.	CODE	MNEM.	ACTION
0909	9E	DRET:GHI LC	Return with High Delay Constant
0A	D3	SEP PC	in D-Register
0B	43	DENT:LDA PC	Get argument byte, Note: Lower 3 bits;
0C			000=½ BT, 001=1 BT, 010=1½ BT, etc.
0C	FC 08	OHLP:ADI X'08'	Perform overhead adjustment
0E	3B 0C	BNF OHLP	
10	52	HBTLP:STR ST	Store result in STACK. (Requires free
11			Byte here) Results = N of ½ BT
11			
11	8E	GLO LC	Count Down Lower Signif. No. of
12	FF 01	LTLP:SMI 1	Instructions
14	33 12	BDF LTLP	
16			
16	9E	GHI LC	Countdown upper significant No. of
17	FA 3F	ANI X'3F'	Instructions
19	32 28	UCLP:BZ NBTCK	If LC (w/o Flags) = 00, GO ON
1B	22	DEC ST	Else; Countdown LC.1 at the rate
1C	52	STR ST	of 512 inst. times LC.1
1D	F8 FC	LDI X'FC'	
1F	FF 01	USLP:SMI 1	
21	33 1F	BDF USLP	
23	42	LDA ST	
24	FF 01	SMI 1	
26	30 19	BR UCLP	
28	02	NBTCK:LDN ST	Check for N of ½ BT's
29	FF 01	SMI 1	and repeat if needed
2B	3B 09	BNF DRET	Else RETURN
092D	30 10	BR HBTLP	

INPUT ROUTINE

LOC.	CODE	MNEM.	ACTION
092F	F8 80	INPENT:LDI X'80'	ENTRY. Load a one bit in the
31	BF		High Bit of CHASSY. When it
32			is shifted out, the character will
32			be assembled.
32			
32	35 32	B2 *	Wait for START BIT
34			
34	DC F8	SEP RC X'F8'	Delay ½ BT
36			
36	FE	SHL	Set DF with ECHO Flag and Echo
37	33 3A	BDF SETONE	Back start bit if set
39	38	SKP	
3A	7B	SETONE:SEQ	
3B			
3B	DC F9	SEP RC X'F9'	Delay 1 BT, get first Bit
3D	C8	LSKP	
3E			
3E	DC F9	GNXT:SEP RC X'F9'	Delay 1 BT, get next Bit
40			
40	3D 47	BN2 GOTZER	Test Sense Line
42			
42	FF 00	SMI 0	Bit is a "1", Set DF=1
44	7A	REQ	
45	30 4D	BR STRBT	
47			
0947	FA 80	GOTZER:ANI X'80'	Bit is a "0", DF=0 already
47			Check Echo
49	3A 4C	BNZ ECONE	Echo Flag set, so echo the "0" Bit.
4B	38	SKP	
4C	7B	ECONE:SEQ	
4D			
4D	9F	STRBT:GHI CHASSY	Store Bit into character assembly
4E	76	RSHR	BYTE
4F	BF	PHI CHASSY	
50			
50	3B 3E	BNF GNXT	Did the "1" Bit leak out?
52			
52	DC F9	SEP RC X'F9'	Yes, character is assembled
54	7A	REQ	Wait one Bit time & send
55			STOP BIT
55			
55	9F	GHI CHASSY	Check for NULL Input
56	FA 7F	ANI X'7F'	If NULL, GO Back and Get
58	32 2F	BZ INPENT	Next Character
5A	9F	GHI CHASSY	Else; RETURN. (Results left
095B	D5	SEP 5	in D-Reg.)

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

OUTPUT ROUTINE

```
LOC.  CODE          MNEM.          ACTION
095C  BF   OUTENT:PHI ACHOUT       Entry
  5D                               Store D-Reg. Data in ACHOUT.
  5D
  5D  FE           SHL             Check for line feed
  5E  FB 14        XRI X'14'       If so; set up loop
  60  3A 65        BNZ NONUL       Count for outputting
  62  F8 07        LDI 7           7 NULL Characters
  64  C8           LSKP            After the "LF" Output
  65  F8 01 NONUL:LDI 1            Else; Load for 1 loop only
  67  AF           PLO CNTR
  68  DC F9        SEP RC X'F9'    Send 1 BT delay and check
  6A  FA 40 TWOSB:ANI X'40'        For 2nd BT delay
  6C  CE           LSZ
  6D  DC E1        SEP RC X'E1'
  6F
  6F  7B           SEQ             Send Start Bit and
  70  DC F1        SEP RC X'F1'    Delay 1 BT
  72
  72  FF 00        SMI 0           Set up check Bit
  74  9F           GHI ACHOUT      And set first Bit
  75  76           RSHR            for OUTPUT
0976  BF           PHI ACHOUT
  77
  77  33 7B OUBITS:BDF OUTONE      Output the Bits
  79  7B           SEQ             Output a zero
  7A  C8           LSKP
  7B
  7B  7A  OUTONE:REQ               Output a one
  7C  7A           REQ             NOP
  7D
  7D  DC F9        SEP RC X'F9'    Delay one BT
  7F  9F           GHI ACHOUT      Set up next Bit
  80  F6           SHR             for Output
  81  BF           PHI ACHOUT
  82  3A 77        BNZ OUBITS      Check for completion
  84                               (Check bit has just
  84                                been shifted out)
  84
  84  2F           DEC CNTR        Yes, completed
  85  8F           GLO CNTR        Check for LF-NULL
  86  7A           REQ             Outputs. Send stop
  87  32 8D        BNZ OUTRET      Bit. If no-NULL's
  89                               needed GO back.
  89
  89  DC F9        SEP RC X'F9'    Else, Repeat Output loop
  8B  30 6A        BR TWOSB
  8D
098D  D5  OUTRET:SEP R5            EXIT
```

TINY BASIC TEST BREAK INTERFACE

```
LOC.  CODE        MNEM.          ACTION
098E  FC 00 TBENT:ADI 0          Set DF=0
  90  35 96       B2 TBRET
  92  FF 00       SMI 0          Reset DF
  94
  94  3D 94       BN2 *          Loop until EF2 goes
  96                             high
0996  D5    TBRET:SEP R5         RETURN
```

INIT ROUTINE

```
LOC.  CODE        MNEM.          ACTION
0997  93    INENT:GHI PC         Entry for BAUD rate
  98                             echo and stop bit
  98                             determination
  98
  98  BC          PHI RC         Set the delay
  99  F8 0B       LDI L(DENT)    routine prog.
  9B  AC          PLO RC         counter
  9C  F8 FF       LDI X'FF'      Set up LC and
  9E  BF          PHI TLC        TLC registers
  9F  BE          PHI LC         for counting 20
  A0  AE          PLO LC         instruction loops.
  A1  1E          INC LC
  A2  F8 F7       LDI X'F7'      Set TLC to -9 for
  A4  AF          PLO TLC        overhead compensation
  A5
  A5  35 A5       B2 *           Wait for Start Bit
  A7                             of carriage return
  A7                             (CR) or "M"
  A7                             (CR) = ECHO
  A7                             "M" = NO ECHO
  A7
  A7  1F     LC2T:INC TLC        First timing interval
  A8  DC F0       SEP RC X'F0'
  AA  3D A7       BN2 LC2T
  AC
  AC  1F     LC3T:INC TLC        Second timing
  AD  DC F0       SEP RC X'F0'   interval
09AF  35 AC       B2 LC3T
```

```
09B1   1F      LC4T:INC TLC        3rd timing interval
  B2   DC F0        SEP RC X'F0'
  B4   3D B1        BN2 LC4T
  B6
  B6   1F      LC5T:INC TLC        4th timing interval
  B7   DC F0        SEP RC X'F0'
  B9   35 B6        B2 LC5T
  BB
  BB   8F           GLO TLC        Set up loop constant
  BC   AE           PLO LC         value. Determine if
  BD   FD D6        SDI X'D6'      2 stop bits are
  BF   9F           GHI TLC        needed
  C0   7D 00        SDBI X'00'
  C2   9F           GHI TLC
  C3   CF           LSDF
  C4   F9 40        ORI X'40'
  C6   BE           PHI LC
  C7
  C7   DC BC        SEP RC X'BC'   Now wait 2½
  C9   35 CD        B2 * +4        Bit times and sense
  CB   F9 80        ORI X'80'      for EF2=1. If so
  CD   BE                          then NO ECHO, else;
  CE                               ECHO.
  CE   DC FB        SEP RC X'FB'   Wait 2 BT's and
09D0   D5           SEP R5         EXIT
```

TINY BASIC INTERFACE SET-UP

```
LOC.   CODE         MNEM.          ACTION
09D1   F8 00        LDI X'00'      Set up Stack Address
  D3   B2           PHI ST
  D4   F8 FF        LDI X'FF'
  D6   A2           PLO ST
  D7   E2           SEX ST
  D8
  D8   F8 09        LDI X'09'
  DA   B3           PHI PC
  DB   B5           PHI R5
  DC   F8 E0        LDI L(TBSENT)
  DE   A3           PLO PC
09DF   D3           SEP PC
```

TINY BASIC INITIALIZATION

```
LOC.   CODE         MNEM.                    ACTION
09E0   F8 E5   TBSENT:LDI L(AFTBUD)          Address
  E2   A5           PLO R5
  E3
  E3   30 97        BR INENT                 Initialize terminal
  E5
  E5   F8 E9   AFTBUD:LDI L(R3CONV)          Return SEP back
  E7   A3           PLO PC                   to R3 and Provide
  E8   D3           SEP PC                   R5 address
  E9   F8 EE   R3CONV:LDI L(GOIN)
  EB   A5           PLO R5
  EC
  EC   30 2F        BR INPENT                Get COLD or WARM
  EE   F8 F2   GOIN:LDI L(CHRTST)            Start Input character
  F0   A3           PLO PC
  F1   D3           SEP PC
  F2
  F2   9F      CHRTST:GHI CHASSY             Is it "C" for
  F3   FE           SHL                      COLD START?
  F4   FB 86        XRI X'86'
  F6   C2 01 00     LBZ X'0100'
  F9
  F9   FB 28        XRI X'28'                Is it "W" for
  FB   C2 01 03     LBZ X'0103'              WARM START
  FE
09FE   30 E9        BR R3CONV                No, try again
```

SERIAL I/O FOR TINY BASIC MACHINE DUMP

```
LOC.  CODE
0900  302F 0030 5C00 308E 009E D343 FC08 3B0C
0910  528E FF01 3312 9EFA 3F32 2822 52F8 FCFF
0920  0133 1F42 FF01 3019 02FF 013B 0930 10F8
0930  80BF 3532 DCF8 FE33 3A38 7BDC F9C8 DCF9
0940  3D47 FF00 7A30 4DFA 803A 4C38 7B9F 76BF
0950  3B3E DCF9 7A9F FA7F 322F 9FD5 BFFE FB14
0960  3A65 F807 C8F8 01AF DCF9 FA40 CEDC E17B
0970  DCF1 FF00 9F76 BF33 7B7B C87A 7ADC F99F
0980  F6BF 3A77 2F8F 7A32 8DDC F930 6AD5 FC00
0990  3596 FF00 3D95 D593 BCF8 0BAC F8FF BFBE
09A0  AE1E F8F7 AF35 A51F DCF0 3DA7 1FDC F035
09B0  AC1F DCF0 3DB1 1FDC F035 B68F AEFD D69F
09C0  7D00 9FCF F940 BEDC BC35 CDF9 80BE DCFB
09D0  D5F8 00B2 F8FF A2E2 F809 B3B5 F8E0 A3D3
09E0  F8E5 A530 97F8 E9A3 D3F8 EEA5 302F F8F2
09F0  A3D3 9FFE FB86 C201 00FB 28C2 0103 30E9
```

FIGURE 2

TEST YOUR E.S.P.

By Gerald M. Van Horn

This original game will test your mental powers. It can be run on a basic or expanded Elf system. To run the program, one person enters a secret number (say 30). The number does not show but the Q light comes on. The second person enters his number, obtained by ESP. If the numbers don't match, the second number is displayed and the Elf stands ready for another try. If they do match, there is a display of E599 and some musical tones, the Elf is then ready for the next round of ESP. After 8 tries (or other number as set by the byte at location 0010 the Elf displays the number of matches (say 03).

Agree with the other person to select numbers from 1 to 7. I like to enter them in the left bytes because the number of matches will appear in the right byte. They say 3 matches out of 7 shows signs of ESP.

LOC.	CODE	COMMENTS
0000	F8 00 B7	Initialization
03	B8 B9 BA	HI order
06	BC BD BE	Registers
09	F8 2F AA	Sub. Return
0C	F8 3C AC	Sub. Address
0F	F8 08 A9	Number of Turns
12	F8 00 A7	Store no. of matches
15	F8 AA A8	Work area
18	58 E8	point
1A	3F 1A	Wait for
1C	37 1C	INPUT
1E	6C AE	Store 1st no.
20	7B	turn on lite
21	3F 21	Wait for
23	37 23	2nd no.
25	6C 64 28	and display
28	7A 8E F3	turn lite out & compare nos.
2B	3A 2F	Jump if unequal
2D	17 DC	count 1 match & GO SUB
2F	29 89	Then go for another try
31	3A 18	or
33	87 58 E8	Display
36	64 28	number
38	30 38	of matches
3A	7A DA	Reset Sub
3C	F8 04 AE	number of
3F	2E 8E	repeats
41	32 3A	Return to MAIN when done
43	F8 E5 A4,	This SUB Program indicates a match
46	54 E4	
48	64 24	
4A	7A	
4B	F8 20 A1	
4E	21 81	
50	3A 4E	
52	31 4A	
54	7B	
55	F8 01 BF	
58	2F 9F	
5A	3A 4B	
5C	F8 99 AD	

LOC.	CODE
5F	5D ED
61	64 2D
63	7A
64	F8 10 A1
67	21 81
69	3A 67
6B	31 63
6D	7B
6E	F8 01 BF
71	2F 9F
73	3A 64
75	30 3F

ESP GAME MACHINE DUMP

```
LOC.  CODE
0000 F800 B7B8 B9BA BCBD BEF8 2FAA F83C ACF8
0010 08A9 F800 A7F8 AAA8 58E8 3F1A 371C 6CAE
0020 7B3F 2137 236C 6428 7A8E F33A 2F17 DC29
0030 893A 1887 58E8 6428 3038 7ADA F804 AE2E
0040 8E32 3AF8 E5A4 54E4 6424 7AF8 20A1 2181
0050 3A4E 314A 7BF8 01BF 2F9F 3A4B F899 AD5D
0060 ED64 2D7A F810 A121 813A 6731 637B F801
0070 BF2F 9F3A 6430 3F
```


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

PICK A BUTTON, ANY BUTTON . . .

By Gus Smeadstad

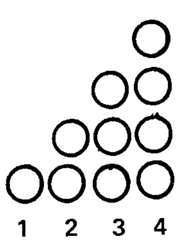
The game of Nim can be played with pebbles, buttons, coins, sharks teeth or Tyrannosaurus rex teeth. Beans are another good choice, as long as they are not cooked or mashed. Thus cooked pinto beans are fine for a taco but they are not useful for nimble nim players. Also, if this has been teleported via an H.G. Wells time machine and you are a so called "ape person" make sure that the Tyrannosaurus rex is suitably dead before playing nim with his teeth. Bearing these few precautions in mind it is possible for a person to have some fun playing nim.

Line your objects up as shown in the diagram. Under the columns place the numbers 1, 2, 3, 4. In the first column you place a single object, in the second you place two, three in the third, and four in the fourth. Say, did you know there is a documented case of a pre-historic tribe where the people only had numbers for one, two, three, and many? Thus, if more that three objects were being discussed you would say, "there are many buffalo in a dozen." Mathematics was easier in those days.

The object is to take the last thing in the game. To play fairly you can only take as many as you like in one column only. Thus, you could choose to take one two, three, or many from one column.

To play against the computer, make your move and then record the number of things you have taken by pressing the row number and the number of buttons you took. When you enter your move (by pressing and releasing INPUT) the computer will reply in the same format. Just remember column and number taken. Now you take the number of buttons for the computer since the computer does not have the ability to do this for itself. Actually the computer really could do this for itself but computers know their place in society and like to make the humans feel needed. Victory is enough for a computer, they aren't out for blood. It's your move.

DIAGRAM



LOC.	CODE	MNEM.	ACTION
0000	90	GHI 0	Set R1.1,
01	B1	PHI 1	R2.1, and
02	B2	PHI 2	R3.1 to 0
03	B3	PHI 3	
04	F8	LDI	Set R2.0,
05	FF		R3.0 to FF
06	A2		
07	A3		
08	F8	LDI	Set R1.0 to
09	1B		subroutine
0A	A1	PLO 1	
0B	3F	BN4	Wait for INPUT
0C	0B		press and release
0D	37	B4	
0E	0D		
0F	E2	SEX 2	Set X=2
10	6C	INP 4	INPUT
11	D1	SEP 1	GOTO subroutine
12	E3	SEX 3	Set X=3
13	64	OUT 4	and output
14	23	DEC 3	move
15	E2	SEX 2	X=2
16	03	LDN	Enter move
17	D1	SEP 1	Go sub
18	30	BR	branch
19	0B		
1A	D0	SEP 0	Go MAIN
1B	A4	PLO 4	store move
1C	FA	ANI	isolate lower
1D	07		
1E	A5	PLO 5	store
1F	FE	SHL	shift
20	A6	PLO 6	store
21	84	PLO 4	give move
22	FA	ANI	cut off lower
23	F0		4 bits
24	FF	SMI	Branch if
25	10		10
26	32	BZ	
27	32		
28	FF	SMI	If 20, branch
29	10		
2A	32	BZ	
2B	35		
2C	FF	SMI	branch if 30
2D	10		
2E	32	BZ	
2F	3E		
30	30	BR	branch
31	45		
32	23	DEC 3	R3-1
33	30	BR	branch
34	1A		
35	86	GLO 6	give no.
36	FE	SHL	shift twice
37	FE		
38	52	STR 2	store at work
39	83	GLO 3	give move address
3A	F7	SM	address-work to D
3B	A3	PLO 3	set address to D
3C	30	BR	branch
3D	1A		
3E	86	GLO 6	give no.#
3F	52	STR 2	store at work
40	83	GLO 3	give move address
41	F7	SM	address-work to D
42	A3	PLO 3	& set address to D
43	30	BR	branch
44	1A		
45	90	GHI 0	Set R7.0 to 00
46	A7	PLO 7	
47	17	INC 7	R7+3
48	17	INC 7	
49	17	INC 7	
4A	25	R5-1	
4B	85	Give unshifted no.	

LOC.	CODE	MNEM.	ACTION
4C	3A		branch if not 00
4D	47		
4E	87	GLO 7	give product
4F	FE	SHL	shift 3 times
50	FE	SHL	
51	FE	SHL	
52	52	STR 2	store at work
53	83	GLO 3	give address
54	F7	SM	address-work to
55	A3	PLO 3	R3.0
56	30	BR	branch
57	1A		

Moves Reply List

LOC.	CODE
0088	00
89	11
8A	31
8B	11
8C	32
8D	31
8E	33
8F	32
90	21
91	F0
92	F0
93	31
94	31
95	32
96	32
97	33
98	22
99	21
9A	21
9B	22
9C	F0
9D	11
9E	31
9F	32
A0	41
A1	F0
A2	F0
A3	41
A4	31
A5	32
A6	32
A7	33
A8	F0
A9	41
AA	41
AB	F0
AC	32
AD	31
AE	33

LOC.	CODE
AF	32
B0	21
B1	22
B2	22
B3	21
B4	41
B5	F0
B6	F0
B7	11
B8	42
B9	41
BA	41
BB	42
BC	F0
BD	11
BE	31
BF	F0
C0	41
C1	42
C2	42
C3	41
C4	21
C5	F0
C6	F0
C7	21
C8	F0
C9	11
CA	31
CB	F0
CC	42
CD	41
CE	41
CF	42
D0	43
D1	42
D2	42
D3	43

LOC.	CODE
D4	41
D5	F0
D6	F0
D7	11
D8	42
D9	43
DA	43
DB	42
DC	F0
DD	11
DE	41
DF	F0
E0	41
E1	F0
E2	F0
E3	31
E4	43
E5	42
E6	22
E7	43
E8	44
E9	43
EA	43
EB	44
EC	42
ED	41
EE	41
EF	42
F0	43
F1	44
F2	44
F3	43
F4	41
F5	42
F6	42
F7	41
F8	42
F9	41
FA	41
FB	42
FC	44
FD	43
FE	43

NIM GAME MACHINE DUMP

```
LOC.  CODE
0000 90B1 B2B3 F8FF A2A3 F81B A13F 0B37 0DE2
0010 6CD1 E364 23E2 03D1 300B D0A4 FA07 A5FE
0020 A684 FAF0 FF10 3232 FF10 3235 FF10 323E
0030 3045 2330 1A86 FEFE 5283 F7A3 301A 8652
0040 83F7 A330 1A90 A717 1717 2585 3A47 87FE
0050 FEFE 5283 F7A3 301A
0060
0070
0080                     0011 3111 3231 3332
0090 21F0 F031 3132 3233 2221 2122 F011 3132
00A0 41F0 F041 3132 3233 F041 41F0 3231 3332
00B0 2122 2221 41F0 F011 4241 4142 F011 31F0
00C0 4142 4241 21F0 F021 F011 31F0 4241 4142
00D0 4342 4243 41F0 F011 4243 4342 F011 41F0
00E0 41F0 F031 4342 2243 4443 4344 4241 4142
00F0 4344 4443 4142 4241 4241 4142 4443 43
```

COMING ATTRACTIONS

- Blockade Arcade Amusement Game
- Music Program—Harmoneous Sequencer
- 15 Puzzle
- And Much, Much, More . . .

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

Publisher . . . . Quest Electronics
Editor . . . . .Bill Haslacher
Proofreading. . . . . Elaine Hale
Circulation. . . . . Janet Roberts

SQUARE ROOT EXTRACTION FOR TINY BASIC

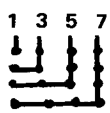
By Phillip B. Liescheski III

This 1802 machine code subroutine is intended to give Tiny BASIC more mathematical muscle. It is called by using the USR function or by coding into the interpreter a square root function call. This routine accepts a positive integer and returns an approximate square root integer value. It returns with a negative one error flag when a negative value is submitted.

The algorithm for this subroutine is based on an interesting property which is related to the series of odd numbers. In number theory, there exists a theorem which states that the square of N is equal to the sum of the series of the first N positive odd numbers. Mathematically, this theorem may be expressed as:

$$\sum_{i=1}^{N} (2i-1) = N^2$$

A nested square tends to give an intuitive proof of this theorem:



The algorithm utilizes this fact by generating an ordered sequence of consecutive odd numbers starting with one. The numerical value whose square root is needed is subtracted by each odd number from the sequence in an orderly fashion, until the numerical value becomes negative. The square root is equal to the number of odd numbers needed to make the numerical value negative. This number is obtained by dividing the last odd number by two. The integer division will result in the square root of the original numerical value since the integer division by two of the i^{th} odd number results in the value i, the number of odd numbers.

This subroutine is designed to utilize the algorithm with 1802 machine code and to be called by Tiny BASIC. The numerical value whose square root is desired is passed to the subprogram through R8. First, the subroutine tests the value contained in R8 to insure that it is positive. This test is performed by masking the sign bit and checking its value. A zero value indicates that the number is positive. A sign bit that is set indicates a negative value. A negative value will cause a negative one (-1) to be pushed into RA.1 and the accumulator, and control returned back to the interpreter, so a negative one indicates error. Next RA is prepared by setting it to one (1). It is used as the odd number generator. The iteration process is entered by pushing the odd number contained in RA onto the stack which has R2 as stack pointer. The value in R8 is moved into the accumulator, subtracted by the top number on stack, and loaded back into R8. The sign bit in R8 is tested to determine the sign of its value. If its value is positive, then RA is incremented twice to generate the next odd number, and the process is repeated until the value of R8 becomes negative. When the value of R8 is negative, the number in RA is divided by two by using a single right shift operation. With this the square root is obtained and is passed back to the interpreter through RA.1 and the accumulator. It should be noted that all of the mathematical operations performed by this subroutine are double-precision.

This new function is intended to broaden the applications of Tiny BASIC. It allows for the use of the distance formula and any other formula needing the square root function. Also it can be used to perform as an absolute value function by first multiplying the numerical value with itself and then taking the square root of the results. One of the shortcomings of this subprogram is that the resulting square root is always an integer with fractional precision loss. This precision loss can be reduced slightly by scaling the numerical value by one hundred before extracting its square root. Since the square root of one hundred is ten, the resulting root will be scaled by ten, thus bringing the digit just right of the decimal point into the units position. For small numerical values, the value may be scaled by ten-thousand to increase precision even more; however, overflow will occur for large values and should be checked. An example which illustrates this train of thought for finding the square root of some small number (two) with three significant figures and remaining in the realm of integers is given below:

```
10 LET A=2*10000
20 LET B=USR(3920,A)
30 LET C=B-100
40 LET D=B/100
50 PR"THE SQ. ROOT OF 2 IS: ";
60 PR D;".";C
70 END
:RUN
THE SQ. ROOT OF 2 IS: 1.41
```

The general format for calling the machine routine with the USR function is:

USR (3920, expr)

The subprogram is designed to be a part of a general BASIC utility package. This is the reason for its odd location in memory; however, it can be easily modified to operate at other locations. It operates on numbers very quickly, and most important, it is based on a very interesting number theory theorem.

REFERENCES

Schmid, Hermann. *Decimal Computation.* New York: John Wiley & Sons, Inc., 1974.

Weller, Walter J. *Assembly Level Programming for Small Computers.* Lexington, Massachusetts: D.C. Heath and Company, 1975.

EXAMPLES OF USR FUNCTION IN TINY BASIC

EXAMPLE NUMBER 1:

```
PRINT USR (3920,144)
12
```

EXAMPLE NUMBER 2:

```
10 LET X=USR (3920,81)
20 PRINT X
30 END
RUN

9
```

EXAMPLE NUMBER 3:

```
10 LET X=USR(3920,81)
11 X=X*2
20 PRINT X
30 END
RUN

18
```

(SQRT EXAMPLES CONTINUED ON PG. 14)

LOC.	CODE	COMMENTS
0F50	98	Entry at 3920_{10}
51	FA 80	Mask sign bit of value
53	32 (59)	Test sign of value
55	F8 FF	Negative value
57	BA	Prepare error flag
58	D5	Error return
59	E2	Positive value; OK
5A	F8 00	Clear odd number generator
5C	AA BA	
5E	1A	Bump odd number
5F	9A	
60	73	Push odd number onto stack
61	8A	
62	52	
63	88	
64	F7	Subtract lower byte of value
65	12	
66	A8	
67	98	
68	77	Subtract upper byte
69	B8	
6A	FA 80	Mask sign bit of value
6C	3A (71)	Test sign of value
6E	1A	Still positive; Generate next
6F	30 (5E)	. . odd number and do it again
71	9A	Finished
72	F6	Divide odd number by two and
73	BA	. . pass results
74	8A	
75	76	
76	D5	RETURN

The Parentheses ( )'s indicate RELOCATION POINTS in the program.

SQUARE ROOT MACHINE DUMP

```
LOC.  CODE
0F50  98FA 8032 59F8 FFBA D5E2 F800 AABA 1A9A
0F60  738A 5288 F712 A898 77B8 FA80 3A71 1A30
0F70  5E9A F6BA 8A76 D5
```

SQRT EXAMPLES

EXAMPLE NUMBER 4:

```
10 LET X=USR(3920,81)
11 Z=X*2+3
20 PRINT Z
30 END
RUN

21
```

EXAMPLE NUMBER 5:

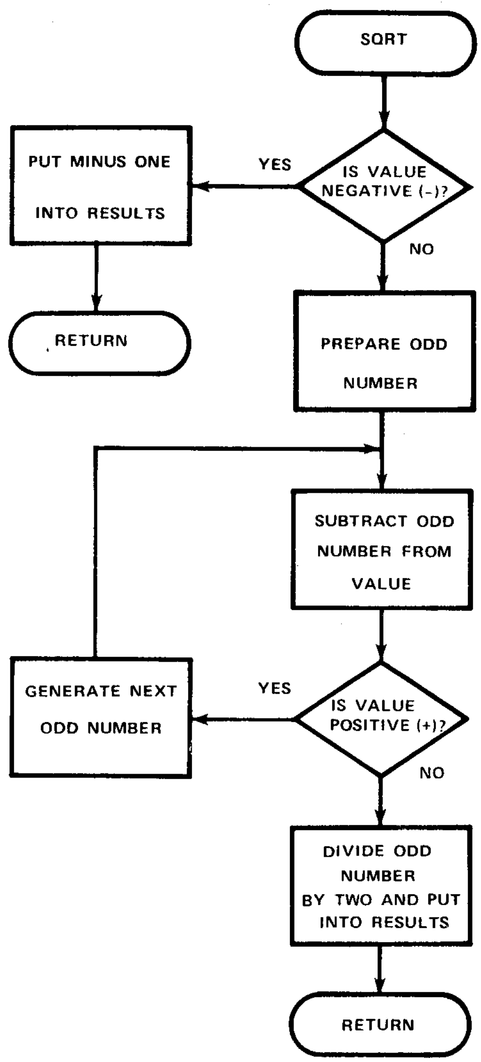
```
10 LET A=10
20 A=A-1
30 LET X=USR (3920,A)
40 PR "SQRT OF b̸";A;"b̸ISb̸";
50 PRINT X
60 IF A=Ø GOTO 80
70 GOTO 20
80 END
RUN

SQRT OF 9 IS 3
SQRT OF 8, ETC.
. . .
SQRT OF 0 IS 0
```

NOTE: b̸ MEANS BLANK SPACE

EXAMPLE NUMBER 6:

```
:PR USR(3920,40000)
-1
:
:
:
```



SQUARE ROOT FLOWCHART

COSMAC CLUB COSMAC CLUB COSMAC CLUB COSMAC CLUB COSMAC CLUB COSMAC

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

ADDRESS CORRECTION REQUESTED

BULK RATE
U.S. Postage Paid
QUEST
Electronics

Permit No. 549
Santa Clara, CA