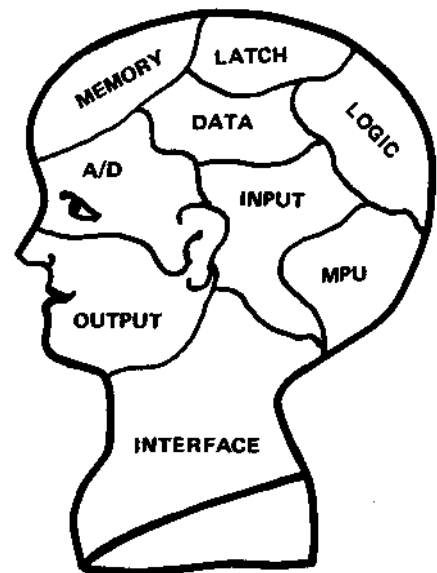


DOWN MEMORY LANE

YOUR computer will say, "thanks for the good memories" after running this program. The program, as written, will quickly test all 4K of a Super Elf and report any failed locations. The comprehensive memory test fits snugly into the 256 bytes of RAM of the original Elf (which moves up to locations 9800-98FF when the original Elf is plugged into the Super Expansion System).

After loading the program at locations 9800-9872, you can enter a long jump at location 0000 (C0 98 00) or you can use a monitor jump to location 9800. If your computer fails the test, what do you do? By depressing the INPUT key twice (once for hi addr. and once for low addr.) you find the location or block of memory which has bad memory. The Quest Super Elf Expansion Board uses 2102 static RAM's. The memory of a single 2102 contains 1024 locations (computer people call it 1K although it is obviously not an even thousand). The 2102 memory chip contains one bit of DATA in each of the 1K locations.

You now know which 1K block in which the memory failure occurred. If the location after INPUT is pressed twice is revealed to be 013F, you have a memory problem in the first 0000-03FF block (1K). This block is located in the back or farthest away position as you look at the board with the parallel ports nearest you and the board number and Quest logo so you can read it. If you wanted you could replace the 8-2102's with new memory chips, and the problem would be solved. With memory of the 2102 type being about \$1.00 and everyone wanting to save money, we proceed to finding the exact culprit. To discover which 2102 has devilishly foiled our otherwise perfect memory we decode the failed bits. Depressing the INPUT key reveals that the information inside (DATA- D0 thru D7) should have been 02 but the actual information stored was 00. In other words, 0000 0000 should have been 0000 0010. "Ha, ha-got you!" you exclaim. You then proceed to replace the memory chip U17 (the second from the left in 1K memory bank). You restart the memory program and all the memory checks out OK. Its actually kind of fun detective work to track down the murdered chip and discover who-done-it.



The memory array on the Super Expansion is organized as follows:

TOP ROW is 0000-03FF
2nd ROW is 0400-07FF
3rd ROW is 0800-0BFF
4th ROW is 0C00-0FFF

Bit 0 is on the LEFT with Bit 7 on the RIGHT.

You deserve some explanation of the Assembler notations used in the program. One of the products on the drawing board at Quest is an Assembler, so that this program will be a good "how to" reference for you in the future. Since Assemblers yield insight into machine language programming, the time to start thinking about them is at hand.

ASSEMBLER EXPLAINED

So far we have been "assembling programs by hand," this process, given the regularity of the COSMAC code and the small size of our programs, is not too bad. Probably, however, you have found

yourself wishing that you had something the equivalent of relative addressing. The Assembler gives you the ability to jump to **LABELS** instead of absolute value hex locations. After you construct a program using the mnemonics, symbols and labels, you make an Assembler run and out comes hex code and memory locations. Historically, Assemblers were one of the first higher level languages. They make it easier for programmers to write, document and maintain software.

The first thing you notice about the Assembler listing is that there are both line numbers and locations. The line numbers are used to reference specific lines of code, so that a programmer knows exactly what is happening. An error in line 18 would tell the programmer that perhaps there is something wrong with the symbol he is using. It is the same reason that books have page numbers -- "please turn to page 53, for today's grammar lesson," says the English teacher.

"Rule number one is that all comments and remarks have two **PERIODS** . . in front of them," says the mystic rites of Assemblers teacher. The two periods in front of the remarks is the RCA convention and it makes it easy to draw neat little boxes, so it is an OK rule. The rest of the line after the two periods is ignored by the Assembler but the diligent programmer should not ignore putting them in. The rule of thumb is that it is far better to have too many comments and remarks (they can always be ignored) than too few. Programmers who ignore this are savegly sacrificed upon the alter of **GIGO** (Garbage In, Garbage Out). Judging from the level of documentation submitted with your programs to **QUESTDATA**, you need not fear being sacrificed to the god **GIGO**.

Let's look at line 18. **START=#0000** is called a directive by RCA (probably to avoid calling it a pseudo-op -- a word IBM has copyrighted). Anyhow, the **EQUATE (= sign)** directs the Assembler to identify the symbol **START** with 0000. These two become twins or two peas in a pod -- when the Assembler sees the word **START** it thinks to itself, "Ah, me put'em in 0000, now." The number 0000 prefixed by the "#" sign is a constant and is one of several constants used in the program. Here it is a hex constant as indicated by the "#" symbol. Register constants have two hex digits in this program. The stack register is hex 02 (referencing one of the 16 scratchpad registers).

All the assignments have been made and the program is ready to begin at line 162. **ORG** is the RCA directive to start assigning locations from that location on with the information given on the **ORG** statement. **ORG MEMENTST** has been assigned the location value 9800 on line 20. So putting it all together the Assembler shorthand is saying, "start assigning locations at location 9800." Without an **ORG** statement the Assembler would assume that you were starting the program at 0000 (which is OK if you are starting at 0000). **ORG #9800** would have been another choice for starting

the program at location 9800 hex. **ORG's** can be used anytime. One of the really nice things about an Assembler is that it gives you this freedom of choice. You can represent things in a manner which is convenient, meaningful and easy to use (once you have learned its tricks).

Line number 190 contains the label **SUBRET**. The **COLON :** is used to the right of the symbol **SUBRET** to identify that this is a "statement label" The computer notes the location of all labels in its "symbol table" and notes both the label and the loc. to which it refers. Now when the programmer wants to jump to the **SUBRET** location (location 9814), all that is needed is **BR SUBRET**. This is exactly what happens in line 201, where the programmer uses this jump to the **SUBRET** label. Since the computer knows the exact hex location of **SUBRET** it is possible to refer to location 9816 as **SUBRET+2**. The equate at line 21 (**ADDBUF=#981E**) is used by the Assembler's computation ability later on in the program. The computer knows (computers aren't so dumb) that **ADDBUF=#981E** from line 21. So when it comes to line 228 and sees **ADDBUF+1**, it goes to its table and finds **ADDBUF** and adds one, thus arriving at 1F for the lower byte of this address (**1E + 1 = 1F**).

The **ASTERISK *** in the RCA Assembler is a handy device to refer to the location the computer is now at. For example, when we have wanted to wait for an **INPUT** button to be pressed we have used **3F 18** (18 being the address of the 3F), now we do not even have to glance at where we are on the location part of the coding sheet. **BN4 *** will branch back to the start of the two byte **BN4** instruction. An **ORG*+6** will skip six locations from the point it is found in the program.

Anything inside a **PARENTHESIS ()** is considered to be an address by the Assembler. This represents another kind of constant, an "address constant." An example of this is line 170 in our **MEMTST** program. **LDI A.1 (ADDBUF)** will cause the Assembler to frantically scramble for its symbol table (**ADDBUF=#981E**), and put the high part of its findings (the A.1 or 98) into the second half of the F8 load into the D part of the load immediate. In the case of, **A(START)** it takes the whole address. Line number 205 loads 0000 into the two bytes starting with 981E. The comma in front of the **A(START)** is another Assembler construction. It tells the Assembler to obtain the address, or address part, and insert it directly into the program at that point.

The **MEMTST** program, like all programs, uses the **END** directive to halt the Assembly process.

Assemblers offer many different ways to declare data, specify addresses and make the programmer's life easier. While Assemblers are not difficult, they do take some study to understand. Assemblers are really helpful in putting together **MACHINE LANGUAGE** with half the effort.

```

0000 1 .. MEMORY TEST PROGRAM
0000 2 ..
0000 3 ..
0000 4 .. VERSION 1.0
0000 5 ..
0000 6 .. TO LOAD AND RUN PROGRAM:
0000 7 ..
0000 8 .. LOAD PROGRAM AT LOCATION DESIGNATED BY THE
0000 9 .. SYMBOL "MEMTST". USE SUPER MONITOR OR OTHER
0000 10 .. MEANS TO BRANCH INTO THE PROGRAM AT LOCA-
0000 11 .. TION 00. ***** CAUTION ***** DO NOT USE REGS
0000 12 .. 2,4,5, OR 6 AS PROGRAM COUNTER WHEN ENTER-
0000 13 .. THIS PROGRAM.
0000 14 ..
0000 15 .. OPERATION OF PROGRAM:
0000 16 ..
0000 17 .. 1. PROGRAM PARAMETERS:
0000 18 .. START=#0000
0000 19 .. STOP=#0FFF
0000 20 .. MEMTST=#9000
0000 21 .. ADDBUF=#901E
0000 22 .. STACK=#90FF
0000 23 .. EXADR=#8000
0000 24 ..
0000 25 .. THE START ADDRESS FOR THE PORTION OF MEMORY
0000 26 .. TO BE TESTED IS DEFINED BY THE PARA. "START"
0000 27 .. IT IS LOCATED AT THE ADDRESS DEFINED BY THE
0000 28 .. PARAMETER "ADDBUF". YOU CAN PATCH A NEW ONE
0000 29 .. AT ADDBUF+0 AND ADDBUF+1. SIMILARLY, THE
0000 30 .. STOP ADDRESS FOR THE MEMORY UNDER TEST IS
0000 31 .. DEFINED BY THE PARA. "STOP". YOU CAN PATCH
0000 32 .. A NEW STOP ADDRESS AT ADDBUF+2 AND ADDBUF+3
0000 33 ..
0000 34 .. THE LOCATION OF THE ADDRESS BUFFER CAN BE
0000 35 .. PATCHED TO A DIFFERENT LOCATION IN MEMORY
0000 36 .. BY CHANGING MEMTST+1 TO THE NEW HI ADDRESS
0000 37 .. AND MEMTST+39(DEC) TO THE NEW LO ADDRESS.
0000 38 ..
0000 39 .. THE STACK ADDRESS IS DEFINED BY "STACK". A
0000 40 .. NEW STACK AREA CAN BE PATCHED IN BY CHANG-
0000 41 .. ING MEMTST+4 TO THE NEW HI ADDRESS, AND BY
0000 42 .. CHANGING MEMTST+7 TO THE NEW LO ADDRESS.
0000 43 ..
0000 44 .. THE EXIT ADDRESS CAN BE PATCHED TO A DIFF-
0000 45 .. ERENT QUARTER-K OF MEMORY THAN CURRENTLY
0000 46 .. DEFINED BY "EXADR" (BUT NOT TO ANY ADDRESS
0000 47 .. ON THE QUARTER-K THAN 00) BY CHANGING THE
0000 48 .. BYTE AT MEMTST+11(DEC).
0000 49 ..
0000 50 .. THE PROGRAM IS RELOCATABLE PROVIDED THE
0000 51 .. START/STOP ADDRESS BUFFER POINTER (R4) IS
0000 52 .. PROPERLY HANDLED AS SUGGESTED ABOVE. THE
0000 53 .. STACK IS CARED FOR, AND A FURTHER PATCH
0000 54 .. MADE AT MEMTST+11(DEC); REPLACE THE DATA

```

```

55 .. THERE WITH THE DESIRED HI ADDRESS OF THE
56 .. NEW QUARTER-K.
57 ..
58 .. 2. PROGRAM OPERATION:
59 ..
60 .. ONCE THE PROGRAM IS ENTERED, IT STARTS
61 .. LOADING TEST PATTERNS IN BLOCKS AND COMP-
62 .. ARES MEMORY IMMEDIATELY AFTER EACH BLOCK IS
63 .. LOADED. ALL VALUES OF 00 THROUGH FF ARE PUT
64 .. INTO EACH CELL OF MEMORY IN THE REGION DE-
65 .. SIGNATED. THE TECHNIQUE USED IS TO START AT
66 .. THE TAIL-END OF MEMORY FIRST WITH 00 AND
67 .. USE THE NEXT HIGHER COUNT OF THE PATTERN IN
68 .. THE PRIOR ADDRESS ETC. UNTIL THE DESIGNATED
69 .. BLOCK OF MEMORY HAS BEEN FILLED. THE RE-
70 .. SULTING SET OF PATTERNS IS THEN CHECKED TO
71 .. DETERMINE IF ANY ERRORS HAVE OCCURED IN THE
72 .. STORAGE OPERATION. ERRORS FOUND ARE REPORT-
73 .. ED TO THE USER AS EXPLAINED BELOW. THEN THE
74 .. PROGRAM RETURNS TO THE LAST LOCATION OF THE
75 .. BLOCK OF MEMORY, STORES 01 AND REPEATS THE
76 .. ABOVE STORE AND COMPARE PROCEDURE. THIS RE-
77 .. PEATS OVER AND OVER UNTIL FF HAS BEEN STOR-
78 .. ED AND COMPARED IN THE LAST LOCATION. THEN
79 .. THE PROGRAM LOADS EXADR IN R0, ASSIGNS R0
80 .. TO X AND SEPS TO R0.
81 ..
82 .. IF AN ERROR IS DETECTED, THE FOLLOWING
83 .. PROCEDURE IS FOLLOWED:
84 .. A. DEPRESS I KEY, THE HI ADDRESS OF THE
85 .. FAILED MEMORY LOCATION IS DISPLAYED ON
86 .. HEX DISPLAY.
87 .. B. DEPRESS I KEY, THEN THE LO ADDRESS OF THE
88 .. FAILED LOCATION IS DISPLAYED.
89 .. C. DEPRESS I KEY, THEN THE CURRENT TEST DATA
90 .. ("SHOULD-BE" VALUE) IS DISPLAYED.
91 .. D. DEPRESS I KEY, THEN THE FAILED VALUE IS
92 .. DISPLAYED.
93 .. A SUBSEQUENT I KEY DEPRESSION WILL RESUME
94 .. THE TESTING OF MEMORY.
95 ..
96 .. WHILE THE PROGRAM IS RUNNING AND NO ERRORS
97 .. HAVE BEEN DETECTED, THE CURRENT LAST LOCA-
98 .. TION TEST VALUE IS DISPLAYED. THE INDICAT-
99 .. ION OF AN ERROR IS THE STEADY DISPLAY OF EE
100 .. AND THE GLOWING OF THE 0 LED.
101 ..
102 .. 3. TERMINATION OF RUN
103 ..
104 .. TERMINATION OF THE PROGRAM IS INDICATED BY
105 .. A STEADY 00 ON THE HEX DISPLAY. IF YOU THEN
106 .. DEPRESS THE I KEY, THE PROGRAM WILL RETURN
107 .. TO THE SUPER MONITOR.
108 ..

```

```

0000 0000 109 ..REGISTER ASSIGNMENTS:
0000 0000 110 ..REG.
0000 0000 111 .. 0 USED FOR EXIT FROM PROGRAM, LOADED WITH
0000 0000 112 .. "EXADR" AND SEP'D, AS WELL AS SEX'D.
0000 0000 113 .. EXTR=00
0000 0000 114 .. 1
0000 0000 115 .. 2 USED AS STACK POINTER
0000 0000 116 .. ST=02
0000 0000 117 .. 3
0000 0000 118 .. 4 USED FOR POINTER TO START AND STOP ADD-
0000 0000 119 .. RESS BUFFER, THE BUFFER ADDRESS IS DES-
0000 0000 120 .. IGNATED BY "ADDBUF"
0000 0000 121 .. ADPTR=04
0000 0000 122 .. 5 USED FOR THE SUBROUTINE
0000 0000 123 .. SUB=05
0000 0000 124 .. 6 PROGRAM COUNTER
0000 0000 125 .. PC=06
0000 0000 126 .. 7.0 DUAL USE:
0000 0000 127 .. 1. HOLDS MAJOR LOOP TEST PATTERN
0000 0000 128 .. 2. PROVIDES MAJOR LOOP CONTROL
0000 0000 129 .. TSPAT=07
0000 0000 130 .. 7.1 USED TO STORE A FAILED VALUE DETECTED
0000 0000 131 .. (IF ONE IS FOUND), THE FORM OF THE DATA
0000 0000 132 .. WILL THEN BE THE XOR OF THE CURRENT
0000 0000 133 .. PATTERN.
0000 0000 134 .. FALVAL=07
0000 0000 135 .. 8 HOLDS THE CURRENT TEST PATTERN IN EITH-
0000 0000 136 .. ER THE STORE OR COMPARE LOOPS.
0000 0000 137 .. CURPAT=08
0000 0000 138 .. 9 HOLDS THE CURRENT ADDRESS OF THE MEMORY
0000 0000 139 .. BEING TESTED, INITIALIZED AT THE STOP
0000 0000 140 .. ADDRESS AND INDEXED BACKWARDS FOR THE
0000 0000 141 .. PATTERN STORE, THEN INDEXED FORWARDS FOR
0000 0000 142 .. THE COMPARE
0000 0000 143 .. MEMPTR=09
0000 0000 144 .. A NUMBER OF BYTES COUNTER FOR USE IN THE
0000 0000 145 .. MEMORY STORE LOOP
0000 0000 146 .. STBYCT=0A
0000 0000 147 .. B NUMBER OF BYTES COUNTER FOR USE IN THE
0000 0000 148 .. TEST COMPARE LOOP
0000 0000 149 .. CMBYCT=0B
0000 0000 150 .. NOTE: REGS A AND B INITIALLY HOLD IN THE
0000 0000 151 .. MAJOR LOOP THE VALUE OF THE NUMBER OF
0000 0000 152 .. BYTES OF THE BLOCK OF MEMORY UNDER TEST
0000 0000 153 .. PLUS AN ADDITIVE FACTOR OF 00FF(HEX), TO
0000 0000 154 .. ALLOW DECREMENTING AND TESTING MORE EFF-
0000 0000 155 .. ICIENTLY.
0000 0000 156 .. C
0000 0000 157 .. D
0000 0000 158 .. E
0000 0000 159 .. F
0000 0000 160 ..
0000 0000 161 ..
0000 0000 162 .. ORG MEMTST
9800 9800 163 ..
9800 9800 164 ..
9800 9800 165 ..
9800 9800 166 .. INITIALIZE THE PROGRAM
9800 9800 167 ..
9800 9800 168 ..
9800 9800 169 ..
9800 F890 170 LDI A.1(ADDBUF)
9800 B4 171 PHI ADPTR
9800 9803 172
9800 F890 173 ..
9800 B2 174 LDI A.1(STACK)
9800 B2 175 PHI ST
9800 F8FF 176 LDI STACK
9800 A2 177 PLO ST
9800 E2 178 SEX ST
9800 179 ..
9800 F890 180 LDI A.1(*)
9800 B6 181 PHI PC
9800 B5 182 PHI SUB
9800 183
9800 184 ..
9800 F815 185 LDI SUBENT
9800 A5 186 PLO SUB
9800 187 ..
9800 F822 188 LDI MAIN
9800 A6 189 PLO PC
9800 D6 190 SUBRET:SEP PC
9800 191
9800 192
9800 193
9800 194 ..
9800 52 195 SUBENT:STR ST
9800 64 196 OUT 4
9800 22 197 DEC ST
9800 198 ..
9800 3F18 199 BN4 *
9800 371A 200 B4 *
9800 3014 201 BR SUBRET
9800 202 ..
9800 203 ..ADDRESS BUFFER
9800 204 ..
9800 0000 205 A(START)
9800 00FF 206 A(STOP)
9800 207 ..
9800 208 ..
9800 209 ..
9800 210 ..START THE PROGRAM
9800 211 ..
9800 212 ..
9800 213 ..
9800 F800 214 MAIN: LDI 0
9800 A7 215 PLO TSPAT
9800 216 ..

```

```

9825 ..
9825 ..
9825 ..
9825 ..
9825 AB
9825 52
9826 64
9827 22
9829 F01F
9829 A4
982C 44
982C 52
982E 14
982F 04
9830 A9
9831 F7
9832 AA
9833 AB
9834 24
9835 24
9836 24
9837 44
9838 52
9839 14
983A 04
983B 09
983C 77
983D FC01
983F BA
9840 BB
9841 E9
9842
9842
9842
9842
9842 18
9843
9843
9843
9843
9843
9843 28
9844 BB
9845 73
9846 2A
9847 9A
9848 3A43

217 ..
218 ..
219 .. MAJOR LOOP
220 ..
221 ..
222 ..
223 PATST: PLO CURPAT
224 STR ST
225 OUT 4
226 DEC ST
227 ..
228 LDI ADDBUF+1
229 PLO ADBPTR
230 LDR ADBPTR
231 STR ST
232 INC ADBPTR
233 LDR ADBPTR
234 PLO MEMPTR
235 SM
236 PLO STBYCT
237 PLO CMBYCT
238 DEC ADBPTR
239 DEC ADBPTR
240 DEC ADBPTR
241 LDA ADBPTR
242 STR ST
243 INC ADBPTR
244 LDR ADBPTR
245 PHI MEMPTR
246 SIB
247 ADI 1
248 PHI STBYCT
249 PHI CMBYCT
250 ..
251 SEX MEMPTR
252
253
254
255
256 ..
257 INC CURPAT
258 ..
259 ..
260 ..
261 .. TEST PATTERN STORE LOOP
262 ..
263 ..
264 ..
265 STOR: DEC CURPAT
266 GLO CURPAT
267 STXO
268 DEC STBYCT
269 GHI STBYCT
270 BNZ STOR

984A 19
984B
984B
984B
984B
984B
984B
984B
984B
984B 88
984C F3
984D 3260
984F
984F
984F
984F
984F
984F
984F
984F
984F 7B
9850 87
9851 E2
9852 F0EE
9854 D5
9855 99
9856 D5
9857 89
9858 D5
9859 88
985A D5
985B 97
985C F3
985D D5
985E E9
985F 7A
9860
9860
9860
9860
9860 18
9861 19
9862 28
9863 98
9864 3A48
9866
9866
9866

INC MEMPTR
TEST COMPARE LOOP
COMPARE MEMORY
AGAINST THE SET OF
PATTERNS JUST STOR-
ED
GLO CURPAT
XOR
BZ GSTEST
FAILURE HANDLING
PHI FALVAL
SEX ST
LDI 0EE
SEP SUB
GHI MEMPTR
SEP SUB
GLO MEMPTR
SEP SUB
GLO CURPAT
SEP SUB
GHI FALVAL
XOR
SEP SUB
SEX MEMPTR
REQ
TURN ON THE 0 LED.
DISPLAY THE EE ERROR
INDICATION ON THE
HEX DISPLAY. THEN
UPON USER DEPRESSION
OF THE I KEY PROVIDE
AT EACH DEPRESSION.
THE FOLLOWING DATA:
THE HI ADDRESS OF
THE FAILED MEMORY.
THE LO ADDRESS, THE
EXPECTED PATTERN AND
THE ACTUAL PATTERN
FOUND. AT A SUBSE-
QUENT DEPRESSION OF
THE I KEY, CLEAR THE
ERROR INDICATION AND
GO ON
(RESUME COMPARE LOOP)
GSTEST: INC CURPAT
INC MEMPTR
DEC CMBYCT
GHI CMBYCT
BNZ COMP
SET THE NEXT PATTERN
CHECK FOR END OF
THIS LOOP. IF NOT
DONE, DO THE COMPARE

```



QuestData Documentation and Software by Roger Pittis is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License.

```

9866          325 .. END OF THE MAJOR LOOP-THROUGH
9866          326 ..
9866          327 .....
9866          328 ..
9866 E2       329          SEX ST          ..SET THE NEXT PATTERN
9867 17       330          INC TSTPAT       ..AND CHECK FOR COM-
9868 87       331          GLO TSTPAT       ..LETION. IF NOT DONE,
9869 3A25     332          BNZ PATST        ..START STORING THE
9868          333          ..NEXT PATTERN
9868          334 ..
9868          335 .....
9868          336 ..
9868          337 .. DONE WITH JOB
9868          338 ..
9868          339 .....
9868          340 ..
9868 D5       341          SEP SUB          ..DISPLAY THE END IN-
986C A0       342          PLO EXITR        ..DICATION AND JUMP TO
986D F800     343          LDI A.1(EXADR)     ..THE EXIT LOCATION
986F B0       344          PHI EXITR
9870 E0       345          SEX EXITR
9871 D0       346          SEP EXITR
9872          347          END
    
```

LISTING

```

9800 F898 B4F8 9882 FBFF A2E2 F099 B6B5 F815
9810 A5F8 22A6 D652 6422 3F18 371A 3014 0000
9820 0FFF F800 A7A8 5264 22F8 1FA4 4452 1404
9830 A9F7 AAAB 2424 2444 5214 04B9 77FC 01BA
9840 BBE9 1828 0073 2A9A 3A43 1988 F332 607B
9850 B7E2 F8EE D599 D589 D588 D597 F3D5 E97A
9860 1819 2B9B 3A4B E217 873A 25D5 A0FB 80B0
9870 E0D0
    
```

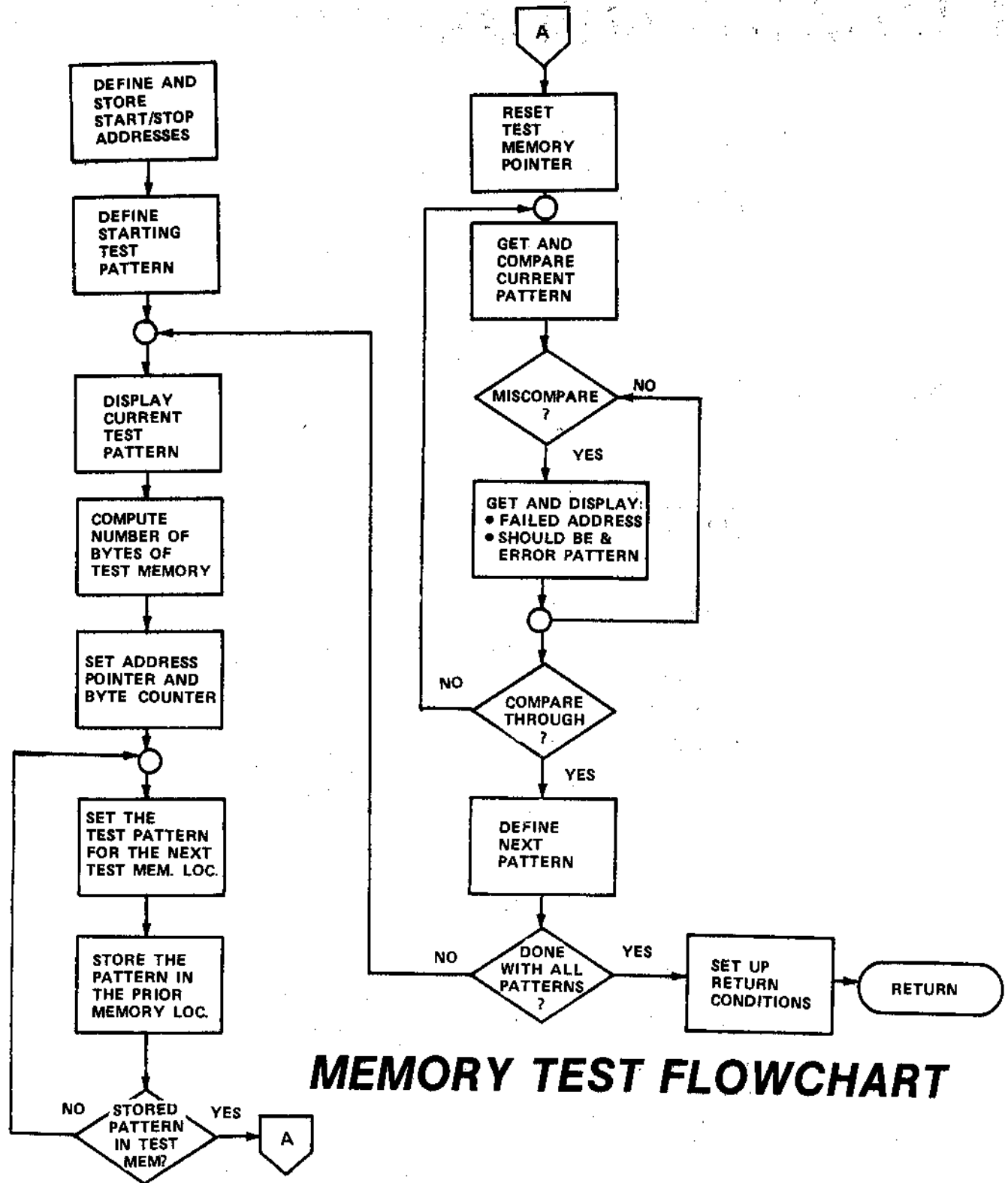
SHADES OF GRAY

By Eugene Jackson

Exciting visual effects will be yours when you load and run **SHADES OF GRAY**. The program displays 15 pages of RAM memory in slow to rapid succession (you choose the rate). Thus, if you load a box outline on page F of memory and fill in the box on pages E and D, an interesting effect will result. By experimenting with material entered in the COSMAC pages via the Super Monitor or other means, you can create blinking lights and 3-D effects. By putting the outline of trains, stoplights, cars, etc. on sequential pages in slightly different positions and places—animation can be produced using this program.

The speed of the sequence of memory pages displayed by the 1861 is determined by keypad entries while the program is running. Thus if you enter 01 you will get 60 pages per second (each page displayed four times per second!). On the other hand, if you enter FF you will get each page displayed steadily for five seconds at a whack. You will find it fun to experiment with different speeds and objects on the display pages.

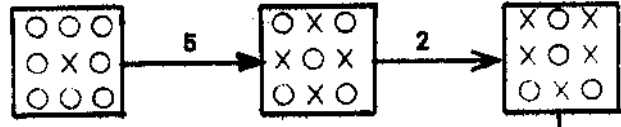
LOC.	CODE	COMMENTS
0000	90 B1 B2 B3 B5 B6	
06	F8 1D A6	Pointer for current page
09	F8 32 A3	PC
0C	F8 4F A2	STACK
0F	F8 15 A1	Video interrupt routine pointer
12	D3	
13	72 70	EXIT (VID. INT. Routines)
15	22 78 22 52	ENTRY
19	C4 C4 C4	
1C	F8 0F B0	Load current pages address
1F	F8 00 A0	
22	80 E2	
24	E2 20 A0 E2 20 A0 E2 20 A0	
2D	3C 22 25 30 13	(Note the DEC of R5)
32	E2 61 22	Turn on VIDEO
35	90 B7	
37	F8 10 A7	Load page counter (counts down)
3A	6C 64 22 A5	Load timer value
3E	85 3A 3E	Timer loop
41	27 87 32 35	
45	56 30 3A	Store current page in Loc. 001D



MEMORY TEST FLOWCHART

Quest Electronic Documentation and Software by Roger Pflin is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License.

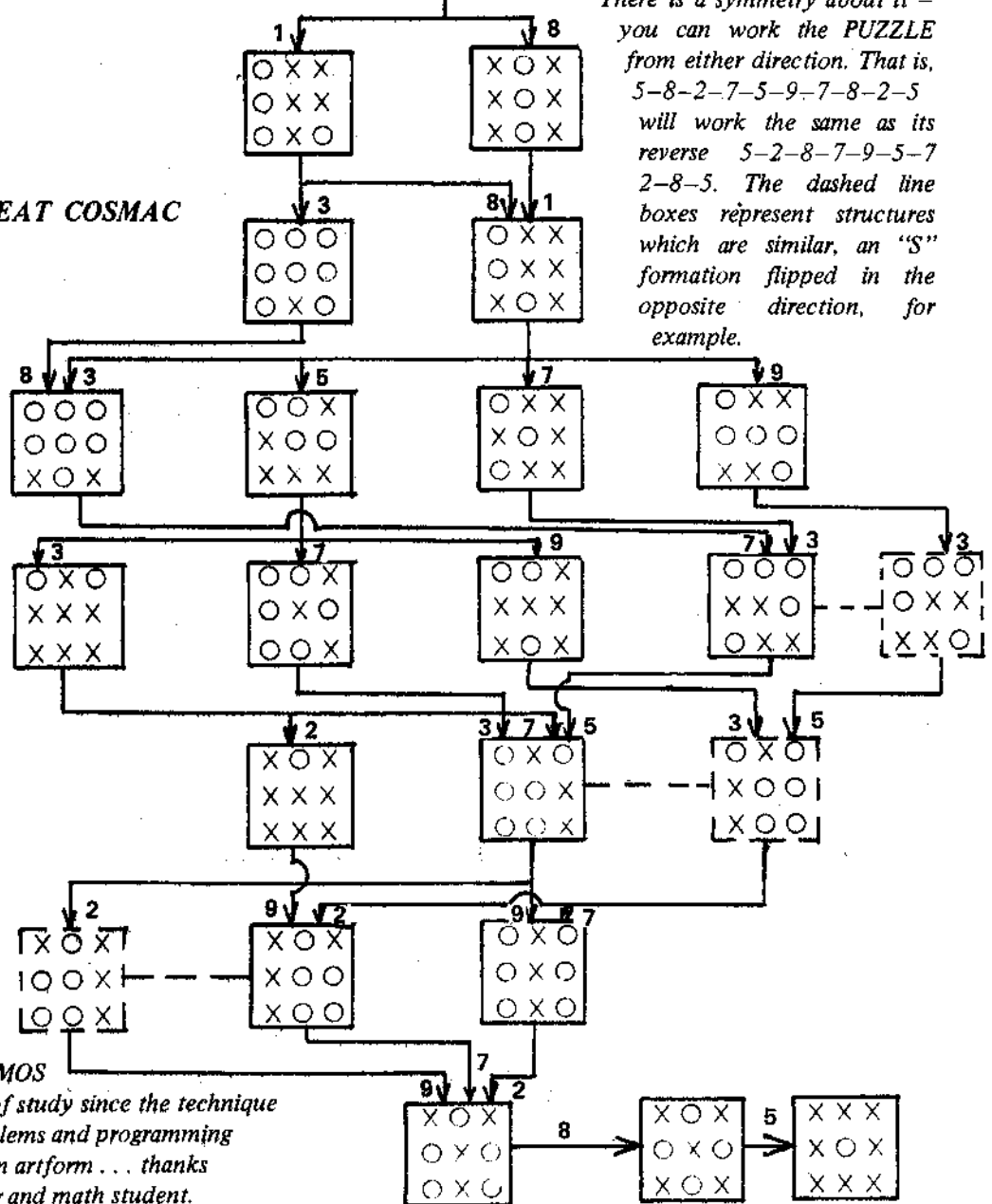
PUZZLE UNSCRAMBLED



Computer people are problem solvers. How did you fare on THE GREAT COSMAC COSMOS PUZZLE in QUESTDATA #10? The solution is as fascinating as the PUZZLE.

There is a symmetry about it - you can work the PUZZLE from either direction. That is, 5-8-2-7-5-9-7-8-2-5 will work the same as its reverse 5-2-8-7-9-5-7-2-8-5. The dashed line boxes represent structures which are similar, an "S" formation flipped in the opposite direction, for example.

Solution to THE GREAT COSMAC COSMOS PUZZLE
By Reed Davis



The GREAT COSMAC COSMOS PUZZLE diagram is deserving of study since the technique can prove useful for other problems and programming situations. Clear diagrams are an artform... thanks go to Reed Davis, programming and math student.

Copyright © 1984 by Reed Davis. All rights reserved. This document is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License.

COSMAC BATTLE OF NUMBERS

[Note: Floyd Oats felt the need to take a break from chess programming (who can blame him), and the result of his diversion is the following video puzzle which will run on 256 byte machines. He has used a modified version of the multipurpose Dzombak TVT. The lack of documentation describing how the program works is deliberate. If you knew how the brainteaser worked you would know the trick. Of course you could always trace through the code—making this a doubly challenging puzzle.]

By Floyd L. Oats

This game will run in practically any COSMAC computer, including the 256-byte basic ELF. It requires exactly one page of memory to run, with the last eighty bytes reserved for variables and I/O. The command code for the game is listed in Table 1, and execution should begin at address hex 00 with the P-Register equal to zero.

The computer will display forty spots at the lower portion of the screen, arranged as five rows of eight dots per row. The player takes turns with the computer in claiming these spots. Each, in his turn, will claim at least one but not more than four spots. Spots claimed by the computer will be replaced by short horizontal bars and spots claimed by the player will

be replaced by long horizontal bars. He who claims the fortieth spot wins the game.

After starting the game, the first step is to decide who goes first. If the player wishes to go first, he enters his claim, hex 01 thru hex 04 (depending on how many he wishes to claim), on the keyboard or toggles and presses the INPUT button. The number of spots claimed will now be replaced by the long bars, and the computer will immediately display short bars to show how many it claims. To cause the computer to go first, enter hex FF and press the INPUT button. The computer will display the short bars and await your response. The machine will reject numbers outside the acceptable range.

The winner of the game is acknowledged on the hex display. If the computer wins, a hex CC will be displayed and the Q-LED will light. If the opponent wins, a hex 55 is displayed and the Q-LED is out.

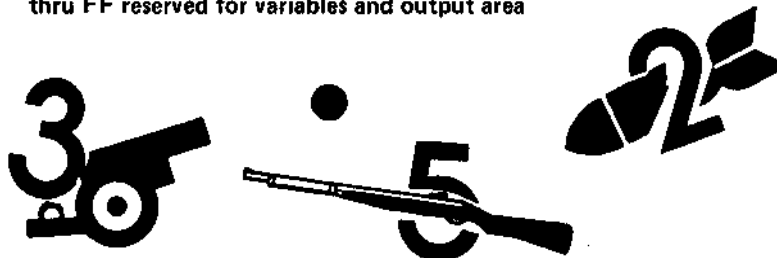
Even after you master the game, you can still have fun watching your friends try to beat it. People sometimes do funny things in attempts to defeat the computer.

I will leave you with two general rules which might help you to defeat the computer: (1) A SINGLE human error will, in ALL cases, result in a computer victory. (2) If the human player claims first, the computer has a WON GAME! Good luck!

COSMAC BATTLE OF NUMBERS

BY F.L. Oats

LOCATION	DATA															
0000	90	B1	B2	B3	F8	2E	A3	F8	B4	A2	F8	10	A1	D3	72	70
0010	22	78	22	52	C4	C4	C4	F8	00	A0	F8	00	B0	80	E2	E2
0020	20	A0	E2	20	A0	E2	20	A0	3C	1D	30	0E	00	00	E2	69
0030	F8	A7	AC	90	BF	BC	BA	F8	B0	AF	FA	08	32	42	F8	08
0040	30	44	F8	00	5F	1F	8F	3A	3A	A8	2F	F8	B8	AF	7A	EC
0050	1D	3F	50	37	53	6C	FB	FF	32	66	FC	01	33	50	FC	04
0060	3B	50	7A	0C	30	7B	31	4F	F8	A8	AA	EA	88	F5	33	73
0070	1A	30	6C	3A	7A	8D	FA	03	FC	01	7B	A9	31	82	F8	7E
0080	30	84	F8	1C	5F	1F	8F	FA	08	32	85	18	88	FB	28	32
0090	99	29	89	3A	7C	31	4F	30	68	31	A3	F8	55	EC	5C	64
00A0	2C	30	A1	F8	CC	30	9D	00	00	05	0A	0F	14	19	1E	23
00B0	thru FF reserved for variables and output area															



A FREE EDUCATION ON SLOT MACHINES

By Patrick E. Taylor

The idea and basic flowpath for SLOT MACHINE came from Radio Shack's *Computer Programming in BASIC for Everyone* (62-2015). Needless to say, the program underwent a significant rewrite.

SLOT MACHINE is a Tiny BASIC program in which the player bets \$1 and then the Elf randomly selects a combination of three oranges, lemons or cherries. Three-of-a-kind wins \$6, all other combinations lose. The Elf keeps track of the money and ends the game when you lose your last dollar. A sample run would look like this:

RUN

```
$1 SLOT MACHINE
PAYOFF IS $6 FOR
3 OF A KIND. ALL
OTHERS LOSE.
NUMBER OF DOLS.
TO START? 10
DO YOU WISH TO
PLAY (1 YES 0 NO)
?1
*OR**LM**CH*      TOO BAD YOU LOST
```

```
YOU NOW HAVE $9
DO YOU WISH TO
PLAY (1 YES 0 NO)
?1
*LM**LM**LM*      YOU WON $6
```

```
YOU NOW HAVE $15
DO YOU WISH TO
PLAY (1 YES 0 NO)
?0
SORRY ABOUT THAT
```

Listing of Program:

```
10 PRINT "$1 SLOT MACHINE"
20 PRINT "PAYOFF IS $6 FOR"
30 PRINT "3 OF A KIND. ALL"
40 PRINT "OTHERS LOSE."
50 PRINT "NUMBER OF DOLS."
60 PRINT "TO START";
70 INPUT M
80 LET X=RND(345)
90 PRINT "DO YOU WISH TO"
92 PRINT "PLAY (1 YES 0 NO)"
100 INPUT A
110 IF A=0 THEN GOTO 410
120 LET C=0
130 LET L=0
140 LET O=0
150 LET I=0
155 IF I=3 THEN GOTO 270
160 LET N=RND(3)+1
170 IF N=1 THEN GOTO 180
172 IF N=2 THEN GOTO 210
174 IF N=3 THEN GOTO 240
180 PRINT "**CH*";
190 LET C=C+1
200 GOTO 260
210 PRINT "**LM*";
220 LET L=L+1
230 GOTO 260
240 PRINT "**OR*";
250 LET O=O+1
260 LET I=I+1
265 GOTO 155
270 IF C=3 THEN GOTO 350
280 IF L=3 THEN GOTO 350
290 IF O=3 THEN GOTO 350
300 PRINT "      TOO BAD YOU LOST"
      ↙ leave 8 spaces
310 LET M=M-1
320 PRINT
330 IF M=0 THEN GOTO 400
340 GOTO 380
350 PRINT "      YOU WON $6"
      ↙ leave 8 spaces
360 LET M=M+6
370 PRINT
380 PRINT "YOU NOW HAVE $";M
390 GOTO 90
400 PRINT "NO MORE MONEY"
410 PRINT "SORRY ABOUT THAT"
420 END
```

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

Publisher Quest Electronics
Editor Bill Haslacher

Circulation Laurie Buzzell

The contents of this publication are copyright © and shall not be reproduced without permission of QUESTDATA. Permission is granted to quote short sections of articles when used in reviews of this publication. QUESTDATA welcomes contributions from its readers. Manuscripts will be returned only when accompanied by a self-addressed stamped envelope. Articles or programs submitted will appear with the authors name unless the contributor wishes otherwise. Payment is at the rate of \$15 per published page. QUESTDATA exists for the purpose of exchanging information about the RCA 1802 microcomputer. Subscriptions are \$12 for this monthly publication.

QUESTDATA COSMAC CLUB

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

ELECTRONIC DICE

By Ron Binning

This program uses a fast sequential count to simulate the roll of a pair of dice. Maximum length of time between addition of one to the value of the dice is about 3 ten-thousandths of a second, so it leaves to chance any roll.

Load program at location M(0000) or use ROM monitor. When using 32 byte ROM monitor, add starting location to M(17), M(21) and M(23).

To start your roll, push run (G) button and to stop the roll, push the wait (W) button.

LOC.	CODE	MNEM.	ACTION
0000	90	GHI 0	Initialization
01	B1	PHI 1	
02	F8	LDI	Pointer to wk. area
03	FC		
04	A1	PLO 1	R(1) points to 00FC
05	F8	LDI	Starting value of roll is 11
06	11		
07	51	STR 1	Mem. loc. FC=11
08	E1	SEX 1	X=1
09	64	OUT 4	Show loc. FC
0A	21	DEC 1	Restore address R(1)
0B	F0	LDX	D Reg. = Loc. ff
0C	FC	ADI	ADD 1 to D Reg.
0D	01		
0E	51	STR 1	Put result in loc. FC
0F	F0	LDX	D Reg. = Loc. FF
10	FE	SHL	Shift left
11	FE	SHL	to get rid of
12	FE	SHL	high order digit
13	FE	SHL	... (thus 4 SHL)
14	FB	XRI	Test
15	70		for 7
16	3A	BNZ	If D ≠ 0
17	08		then GOTO 08
18	F0	LDX	D Reg. = Loc. FF
19	FC	ADI	ADD 11
1A	11		to adjust
1B	FF	SMI	Subtract 7
1C	07		to adjust units
1D	51	STR 1	Store in loc. FF
1E	FB	XRI	Test
1F	71		for 71
20	32	BZ	if D ≠ 71
21	05		Then loc. 03
22	30	BR	If not 71
23	08		then loc. 08

WHAT'S THE MYSTERY?

Why has Tiny BASIC source code been secret for so long? What hints to programmers lurk in the Tiny BASIC code? Find out by ordering your very own copy today. The complete source with Assembly listing is available from QUEST Electronics for \$19.

By ordering your copy of the source code you will be able to modify Tiny BASIC code to suit your individual needs. You will also see how Tom Pittman uses a modified version of RCA's SCRT to write the code. QUEST Electronics has had many inquiries about the availability of the source code for Tiny BASIC by people who wish to modify or just read how the compact coding of Tiny was achieved. If you have been wondering "just what exactly is the Interpretive Language embedded in the Tiny BASIC code?"—you now have a chance to find out. There are many interesting tricks and routines you can adapt to other purposes. Now is your chance to find out exactly what the "mystery" source code is all about.

For a peek and poke at the inner workings of Tiny BASIC and the inventive mind of Tom Pittman, don't pass up this opportunity to buy the Tiny BASIC SOURCE CODE.

QUESTDATA

P.O. Box 4430
Santa Clara, CA 95054

A one year subscription to QUESTDATA, the monthly publication devoted entirely to the COSMAC 1802 is \$12.

(Add \$6.00 for airmail postage to all foreign countries except Canada and Mexico.)

Your comments are always welcome and appreciated. We want to be your 1802's best friend.

Payment:

- Check or Money Order Enclosed
Made payable to Quest Electronics
- Master Charge No. _____
- Bank Americard No. _____
- Visa Card No. _____
- Expiration Date: _____

Signature _____

- Renewal
- New Subscription

NAME _____

ADDRESS _____

CITY _____

STATE _____

ZIP _____

NUMBER GUESS

By Patrick E. Taylor

NUMBER GUESS is a Tiny BASIC program in which the Elf picks a random number and then players A and B try to guess which number the Elf has picked. Ten points are assigned to the player who's guess is the closest. The Elf keeps track of the score. A sample run would look like this:

RUN

```

NUMBER GUESS
PLAYER A VERSUS
PLAYER B. TEN
POINTS TO THE
WINNER. PICK A #
FROM 1 TO 100
PLAYER A? 25
PLAYER B? 78
ELF PICKED 83
PLAYER B WON!
A=0 B=10
LET'S TRY AGAIN
PLAYER A?

```

(and so on . . .)

Note: To end the game enter 0 for player A's guess.

Listing of Program:

```

1 PRINT "NUMBER GUESS"
2 PRINT "PLAYER A VERSUS"
3 PRINT "PLAYER B. TEN"
4 PRINT "POINTS TO THE"
5 PRINT "WINNER. PICK A #"
6 PRINT "FROM 1 TO 100"
15 LET F=0
16 LET G=0
20 PRINT "PLAYER A";
30 INPUT A
35 IF A=0 THEN GOTO 1000
40 PRINT "PLAYER B";
50 INPUT B
60 C=RND (100)
65 PRINT "ELF PICKED ";
66 PRINT C
70 LET D=C-A
80 LET E=C-B
90 IF D*D>E*E THEN GOTO 500
100 PRINT "PLAYER A WON!"
101 LET F=F+10
110 GOTO 950
500 PRINT "PLAYER B WON!"
501 LET G=G+10
950 PRINT "A=";F
951 PRINT "B=";G
990 PRINT "LET'S TRY AGAIN"
991 GOTO 20
1000 END

```

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

11 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