BUILD A MULTIPLAYER LED RACING GAME

14278

Popular Electronics®

WORLD'S LARGEST-SELLING ELECTRONICS MAGAZINE

MARCH 1977/$1.25

16-PAGE BONUS!

SPECIAL FOCUS on CB RADIO

INCLUDING: ★40-CHANNEL CB RIGS & THE NEWEST FCC RULES & REGULATIONS ★EXTERNAL vs. BUILT-IN SPEAKERS ★SUNSPOTS & CB "SKIP" PROSPECTS ★BUILD THE "SILENCER"—Squelches Auto Radio When Receiving CB Signal

Build A Digital Bicycle Speedometer

FRICTIONLESS DEVICE USES LED s

How To Expand An "Elf" Microcomputer's Memory

Build A 10-Hz To 1-MHz EPUT Meter

TEST REPORTS

★ Empire 698 Manual Turntable

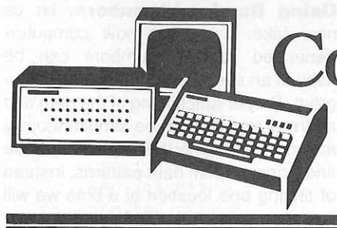
ood S-7910

M/AM Receiver

★ Ba tine 1010A Oscilloscope

2:50-3:10 p.m.	2250-2310	**Santiago, Chile	F	6.195, 9.566, 11.81, 15.15
3:00-3:30 p.m.	2300-2330	Tokyo, Japan	G	15.105
3:00-4:30 p.m.	2300-0030	London, England	G	6.175, 9.51 (via Sackville), 9.58 (via Ascension)
3:00-5:00 p.m.	2300-0100	Montreal, Canada	F	5.96
3:30-4:00 p.m.	2330-2400	**Radio Clarin, S. Domingo, Dom. Rep.	G	11.70 (irregular)
3:45-4:00 p.m.	2345-2400	**Voice of Org. of American States, Washington, USA	G	6.13, 9.64, 11.74
4:00-4:15 p.m.	0000-0015	Tokyo, Japan	G	15.105
4:00-5:00 p.m.	0000-0100	**VOA, Washington, USA	G	11.83, 11.895, 15.40
4:30-5:00 p.m.	0030-0100	Moscow, U.S.S.R.	G	6.02, 7.26, 9.635, 9.78, 12.05, 15.14, 15.18, 15.455, 17.72
4:30-7:30 p.m.	0030-0330	London, England	G	6.175 (via Sackville), 9.51 (via Greenville), 9.58 (via Ascension)
4:40-9:00 p.m.	0040-0500	HCJB, Quito, Ecuador	G	6.095, 9.56, 11.915
5:00-5:15 p.m.	0100-0115	Tokyo, Japan	G	15.105
5:00-5:30 p.m.	0100-0130	Moscow, U.S.S.R.	G	6.02, 7.175, 9.635, 9.78, 12.05, 15.14, 15.18, 15.455
5:00-7:00 p.m.	0100-0300	Melbourne, Australia	G	15.32, 17.795
5:00-8:00 p.m.	0100-0400	Madrid, Spain	F	6.065, 11.88 (Mon.-Sat.)
5:10-5:30 p.m.	0110-0130	**Santiago, Chile	F	6.195, 9.566, 11.81, 15.15
5:30-6:30 p.m.	0130-0230	Tokyo, Japan	G	15.195, 15.235, 17.725, 17.825
		Moscow, U.S.S.R.	G	6.02, 7.175, 9.635, 9.78, 11.86, 12.05, 15.14, 15.18, (to 0200), 15.455 (to 0200)
6:00-6:15 p.m.	0200-0215	Tokyo, Japan	G	15.105
6:00-8:00 p.m.	0200-0400	Taipei, Taiwan	F	9.685, 15.425, 17.89
6:10-6:30 p.m.	0210-0230	**Santiago, Chile	F	6.195, 9.566, 11.81, 15.15
6:30-7:00 p.m.	0230-0300	Stockholm, Sweden	P	9.695, 11.705
		Moscow, U.S.S.R.	G	6.02, 7.26, 9.635, 9.78, 11.86, 15.14
7:00-7:15 p.m.	0300-0315	Tokyo, Japan	G	15.105
7:00-7:30 p.m.	0300-0330	Kiev, U.S.S.R.	G	7.26, 9.58, 9.635, 9.78, 11.86
		Montreal, Canada	G	6.045, 6.065, 9.535, 9.655
7:00-7:55 p.m.	0300-0355	Peking, China	G	7.12, 9.78 (via Tirana), 11.445, 12.055, 15.06, 15.385, 17.735, 17.855
7:00-8:20 p.m.	0300-0420	**Johannesburg, S. Africa	F	3.995, 7.27, 9.58
7:10-7:30 p.m.	0310-0330	**Santiago, Chile	F	6.195, 9.566, 11.81, 15.15
7:20-8:25 p.m.	0320-0425	**TIFC, San Jose, Costa Rica	F	6.035, 9.645, (opens 0300 Sat., Sun.)
7:22-7:28 p.m.	0322-0328	Erevan, U.S.S.R.	G	6.02, 9.54, 9.735, 11.69, 15.14 (Sat./Tue./Wed./Fri.)
7:30-8:00 p.m.	0330-0400	Moscow, U.S.S.R.	G	6.02, 7.26, 9.54, 9.58, 9.635, 9.735, 9.78, 11.69, 15.14
7:30-8:15 p.m.	0330-0415	Berlin, Ger. Dem. Rep.	P	5.955, 6.08, 9.56, 9.73
7:30-8:30 p.m.	0330-0430	London, England	G	6.175 (via Sackville), 9.58 (via Ascension)
8:00-8:15 p.m.	0400-0415	Tokyo, Japan	G	15.105
8:00-8:30 p.m.	0400-0430	Sofia, Bulgaria	F	9.705 (alt. 9.70)
		Budapest, Hungary	F	6.00, 7.215, 9.585, 9.833, 11.91 (Tue., Fri.)
		Montreal, Canada	G	6.045, 9.655
8:00-9:00 p.m.	0400-0500	Moscow, U.S.S.R.	G	602, 7.175, 7.26, 9.54, 9.58, 9.61 (from 0430), 9.635, 9.735, 9.78, 11.69
8:30-9:00 p.m.	0430-0500	Berne, Switzerland	G	6.045, 9.725
8:30-11:30 p.m.	0430-0730	London, England	G	6.175 (via Antiqua)
9:00-9:15 p.m.	0500-0515	Jerusalem, Israel	F	5.90, 7.395, 7.412
		Tokyo, Japan	G	9.505
9:00-9:30 p.m.	0500-0530	Lisbon, Portugal	F	6.025, 11.935 (varies)
9:00-10:20 p.m.	0500-0620	Hilversum, Holland	G	6.165, 9.715, (via Bonaire)
9:00-11:00 p.m.	0500-0700	HCJB, Quito, Ecuador	G	6.095, 9.56
9:00-11:30 p.m.	0500-0730	Moscow, U.S.S.R.	G	6.02, 7.175, 7.26, 9.54, 9.58, 9.61, 9.635, 9.735, 9.78
9:30-9:50 p.m.	0530-0550	Cologne, Ger. Fed. Rep.	G	5.96 (via Antiqua) 6.10 (via Malta), 6.185, 9.545
10:00-10:15 p.m.	0600-0615	Tokyo, Japan	G	9.505
10:00-10:30 p.m.	0600-0630	Oslo, Norway	F	9.645, 11.87 (Sun.)
10:00-11:00 p.m.	0600-0700	Buenos Aires, Argentina	G	9.69 (Mon.-Fri.)
10:00 p.m.-12 mdt.	0600-0800	Pyongyang, Dem. Rep. Korea	G	9.82
10:30 p.m.-12:30 a.m.	0630-0830	Havana, Cuba	G	9.525
10:45 p.m.-12:45 a.m.	0645-0845	**Wellington, N.Z.	F	11.78
11:00-11:15 p.m.	0700-0715	Tokyo, Japan	G	9.505
12 mdt.-12:15 a.m.	0800-0815	Tokyo, Japan	G	9.505
12 mdt.-2:00 a.m.	0800-1000	Manila, Philippines (FEBC)	F	11.92
1:00-1:15 a.m.	0900-0915	Tokyo, Japan	G	9.505
2:00-2:30 a.m.	1000-1030	Tokyo, Japan	G	5.99

*Reception quality, East Coast (West Coast) location: G-good, F-fair, P-poor
**Not intended for North America, but receivable satisfactorily
Days refer to local date in target area.



Computer Bits

By Hal Chamberlin

MEMORY TESTING

IT'S LATE at night and you have just finished assembling an expansion memory board for your computer. You turn the computer on and operate the console. It seems to function OK. Now it's time to run a memory test program to make sure that every one of those 4096 new bytes can store and recall data reliably.

A good memory test routine should be able to detect all possible failure modes on the new board. When used at home, it can be run continuously for a day or two to "burn in" the components and detect early failures while the warranty is still in effect. The CPU, data busses, and power supply are also exercised proving their ability to handle the additional load. In a small business application it may be wise to run a memory test (and tests of other system components as well) before processing sensitive financial data.

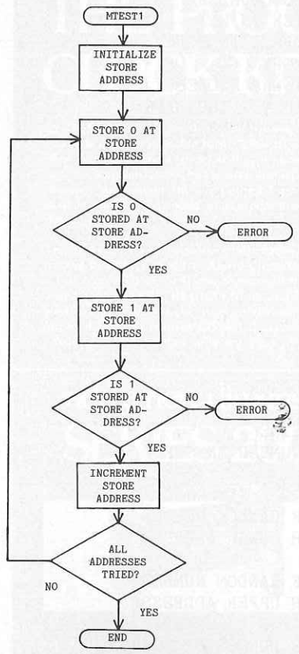


Fig. 1. Simple memory test.

A Simple Test Program. Basically a test of memory amounts to checking that each memory byte will correctly read back previously stored data. Since each byte is in turn composed of 8 bits, the data used for checking should try each bit in the "1" and "0" states. Thus a simple test procedure might be first to write all zeroes into a byte, read it back for checking, try all ones, and then go to the next address until all 4096 bytes are tested. Figure 1 shows a flowchart of such a test routine. Actually this is a very poor testing scheme because it will fail to detect a number of common memory board faults.

Shorts between two closely spaced printed circuit traces is a common problem. Assume a solder bridge short between two adjacent data lines on the board. What this means is that those two bits will always be read back identical to each other regardless of what is actually stored in the memory IC's. Usually zeroes will override; meaning that if either of the "paired" data lines has a 0, it will force the other one to a 0 also. Obviously the test scheme in Fig. 1 would not detect this problem since all bits in the byte are identical. Other complementary patterns such as 252 (10101010) and 125 (01010101) (octal) could be used but no such pattern can guarantee detection of a short between *any* pair of data lines.

Shorts or opens in the large number of parallel address lines are even more likely and would not be detected either. The effect of most address line problems is that the actual number of distinct storage locations is less than the 4096 it should be. Another way to think of this is that two or more *different* addresses will refer to the *same* memory cell. Since the routine uses the same data in each location and only one location at a time is checked, it would probably run OK even if *none* of the address lines worked! About the only circuitry this routine does test is the data buffers (if the board has them) and whatever memory cells that can be addressed correctly.

A Better Test Program. Let us try to design a better testing scheme that detects the common faults noted above. To solve the problem of detecting shorted data lines, we should try to store and recall all 256 possible 8-bit numbers. To detect bad address lines, we should look at all of the other addresses to make

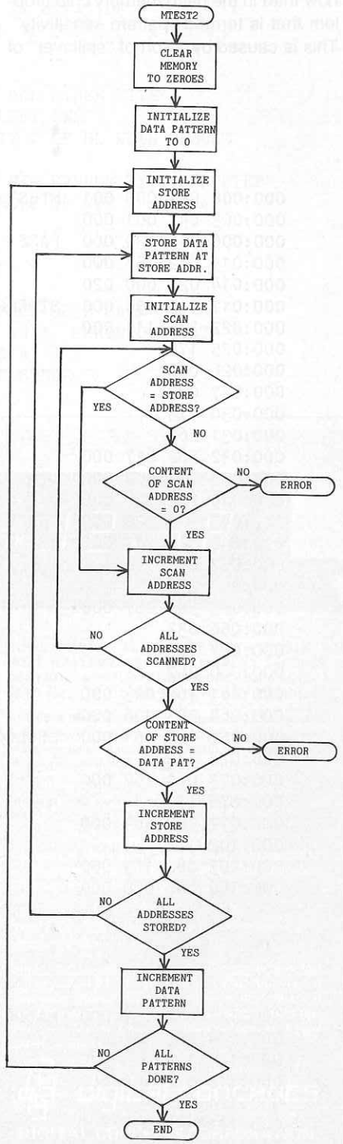


Fig. 2. Better memory test.

sure the data just stored does not pop up someplace else. Figure 2 shows a flowchart for this more effective test procedure. An estimate of the test execution time can be obtained by multiplying the execution time of the inner loop by 4096 locations times 256 data patterns. On a full-speed 8080 this is about 35 microseconds X 4096 X 4096 X 256 or nearly two days!

This routine is quite effective in locating memory board problems but cannot detect a fairly common (though less so now than in the past) memory *chip* problem that is termed "pattern sensitivity". This is caused by a sort of "spillover" of bits into their neighbors on the chip and only causes problems with certain patterns of bits. From the memory chip's point of view this routine writes a single "1" bit in a sea of zeroes and checks that the "1" remains stored and that none of the zeroes is disturbed. As the test progresses, the "1" moves around until all locations are tested. Trying all possible bit patterns is not a feasible solution since there are 2^{1024} of them or about 10^{308} on a typical memory IC. It is possible to make a thorough test of pattern sensitivity in a reasonable time but a detailed knowledge of the particular memory chip's geometry is required.

Using Random Numbers. let us now take a look at how computer-generated random numbers can be used in an even better memory test program. Proper functioning of the data and address circuitry can be simultaneously tested by changing the procedure a little and using random data patterns. Instead of testing one location at a time we will first store data in all of the locations to be tested (the store phase) and then come back and see if all of the locations held their data (the verification phase). Also instead of using the same data in all locations, different random numbers will

(Text continued on p 110)

Fig. 3

```
                        *       MEMORY TEST PROGRAM USING RANDOM NUMBERS
                        *       WRITTEN FOR A 4K BLOCK OF MEMORY ON A 4K BOUNDARY

000:000 061 000 001  MTEST   LXI  SP,400Q          INITIALIZE STACK POINTER
000:003 041 001 000          LXI  H,1               INITIALIZE RANDOM NUMBER SEED
000:006 315 130 000  PASS    CALL RAND              NEW PASS, GET A RANDOM NUMBER IN HL
000:011 042 157 000          SHLD SEED              SAVE AS SEED FOR VERIFY
000:014 021 000 020          LXI  D,4096            INITIALIZE ADDRESS COUNTER
000:017 315 130 000  STORPH  CALL RAND              GET A RANDOM NUMBER IN HL
000:022 315 111 000          CALL MADDR             FORM MEMORY ADDRESS IN BC
000:025 175                  MOV  A,L               STORE RANDOM BYTE IN MEMORY
000:026 002                  STAX B                 AT ADDRESS IN BC
000:027 033                  DCX  D                 DECREMENT ADDRESS COUNTER
000:030 173                  MOV  A,E               TEST IF IT IS ZERO
000:031 262                  ORA  D
000:032 302 017 000          JNZ  STORPH            CONTINUE STORE PHASE IF NOT
000:035 052 157 000          LHLD SEED              RESTORE RANDOM SEED FOR VERIFY PHASE
000:040 021 000 020          LXI  D,4096            INITIALIZE ADDRESS COUNTER
000:043 315 130 000  VERFPH  CALL RAND              GET A RANDOM NUMBER IN HL
000:046 315 111 000          CALL MADDR             FORM A MEMORY ADDRESS IN BC
000:051 012                  LDAX B                 GET DATA FROM MEMORY
000:052 275                  CMP  L                 COMPARE WITH WHAT WAS STORED
000:053 302 067 000          JNZ  ERRLOG            GO TO ERROR LOG IF NOT THE SAME
000:056 033                  DCX  D                 DECREMENT ADDRESS COUNTER
000:057 173                  MOV  A,E               TEST IF IT IS ZERO
000:060 262                  ORA  D
000:061 302 043 000          JNZ  VERFPH            CONTINUE TEST PHASE IF NOT
000:064 303 006 000          JMP  PASS              GO FOR ANOTHER PASS
000:067 062 161 000  ERRLOG  STA  WAS               STORE ERRONIOUS DATA IN ERROR LOG AREA
000:072 175                  MOV  A,L
000:073 062 162 000          STA  SHLDBE            STORE CORRECT DATA
000:076 170                  MOV  A,B
000:077 062 164 000          STA  ERADDR+1          STORE ADDRESS OF ERROR
000:102 171                  MOV  A,C
000:103 062 163 000          STA  ERADDR
000:106 166 000 000          HLT                    HALT OR JUMP TO ERROR PRINT

                        *       SCRAMBLED MEMORY ADDRESS FORMATION ROUTINE
                        *       USES ADDRESS COUNTER IN DE AND RANDOM NUMBER IN SEED
                        *       TO FORM A SCRAMBLED ADDRESS IN BC

000:111 072 157 000  MADDR   LDA  SEED              GET LOWER BYTE OF RANDOM NUMBER
000:114 253                  XRA  E                 EXCLUSIVE-OR WITH LOWER ADDRESS
000:115 117                  MOV  C,A
000:116 072 160 000          LDA  SEED+1            GET UPPER BYTE OF RANDOM NUMBER
000:121 252                  XRA  D                 EXCLUSIVE-OR WITH UPPER ADDRESS
000:122 346 017              ANI  17Q               SAVE ONLY 4 BITS FOR 4K MEMORY
000:124 306 xxx              ADI  (page number)     ADD IN FIRST PAGE NUMBER OF BOARD
000:126 107                  MOV  B,A               BEING TESTED
000:127 311                  RET                    RETURN
```


Fig. 3 (Cont'd.)

```
                        *       RANDOM NUMBER GENERATOR SUBROUTINE
                        *       ENTER WITH SEED IN REGISTERS H AND L
                        *       EXIT WITH NEW RANDOM NUMBER IN H AND L
                        *       USES 16 BIT FEEDBACK SHIFT REGISTER METHOD
                        *       DESTROYS REGISTERS A AND B

000:130 006 010      RAND    MVI  B,8               SET COUNTER FOR 8 RANDOM BITS
000:132 174          RAND1   MOV  A,H               EXCLUSIVE-OR BITS 3,12,14, AND 15
000:133 017                  RRC                    OF SEED
000:134 254                  XRA  H
000:135 017                  RRC
000:136 017                  RRC
000:137 254                  XRA  H
000:140 017                  RRC
000:141 255                  XRA  L                 RESULT IS IN BIT 3 OF A
000:142 017                  RRC                    SHIFT DOWN TO BIT 0 OF A
000:143 017                  RRC
000:144 017                  RRC
000:145 346 001              ANI  1                 CLEAR OUT ALL OTHER BITS
000:147 051                  DAD  H                 SHIFT HL LEFT ONE
000:150 205                  ADD  L                 REPLACE BIT 0 OF HL WITH RESULT
000:151 157                  MOV  L,A
000:152 005                  DCR  B                 TEST IF 8 NEW RANDOM BITS COMPUTED
000:153 302 132 000          JNZ  RAND1             LOOP FOR MORE IF NOT
000:156 311                  RET                    RETURN

                        *       STORAGE FOR MEMORY TEST

000:157              SEED    DST  2                 RANDOM NUMBER SEED SAVE
000:161              WAS     DST  1                 ERROR LOG AREA, ERRONIOUS DATA
000:162              SHLDBE  DST  1                 CORRECT DATA
000:163              ERADDR  DST  2                 ADDRESS OF ERROR
000:165                      END
```

be stored into each location. Finally, instead of storing and verifying in an ascending sequence of addresses, a scrambled sequence based on random numbers will be used.

One potential problem with this method is that, with true random numbers, it is not possible to tell during the verification phase what the stored data should be. One solution would be to retain a copy of the correct pattern in known good memory. A better solution is to use a "pseudo random" number generator. Such a generator works by creating a new number from an old one which is called the "seed." A sequence of random numbers is obtained by repeatedly calling the generator routine giving the last number it produced. If the same initial seed is used, then the sequence of random numbers will be the same. So we have to save only the seed to be able to regenerate the sequence for verification. A scrambled sequence of addresses can be obtained by exclusive OR'ing the lower 12 bits of the memory address with a random number that changes after each "pass" (store and verify phase) through the test routine. Also, after each pass, the data pattern seed is changed so that each pass is totally different.

Using this method the data lines will be thoroughly tested because after a short time all possible data bytes will have been tried. Addressing will be checked out also since an incorrect address during the data store phase is likely to wipe out data stored elsewhere earlier in the phase. The random address scrambling insures that a variety of pattern store sequences will be tried. After a few dozen passes through the routine, the likelihood is extremely high that every bit of memory has been tried in both one and zero states. Although pattern sensitivity of the memory chips is not specifically tested, a great variety of patterns will be tried.

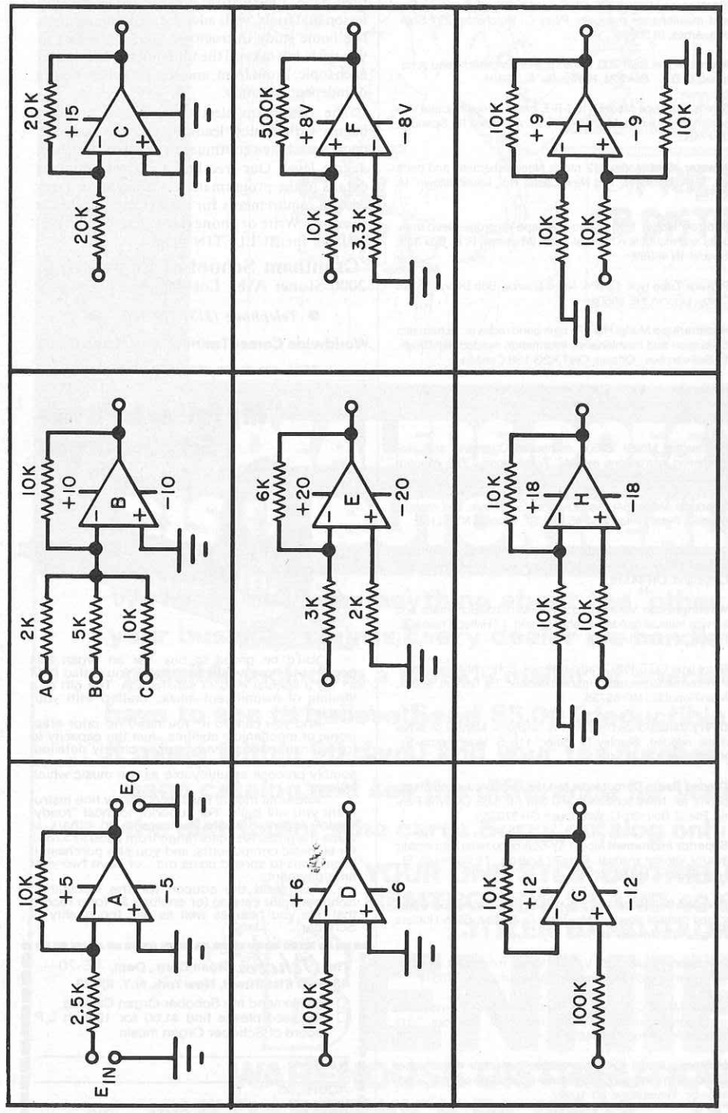
Figure 3 shows a listing of the improved memory test program in 8080 assembly language. The pseudo-random number generator subroutine simulates a 16-bit shift register with feedback for random bit generation (see the *TTL Cookbook* by Don Lancaster). Eight random bits are generated each call and are put together to make a random byte. The seed that is saved for the verify routine is also used as an address scrambler. The test program runs as an endless loop and will not repeat the sequence of addresses and patterns until 65,535 passes have been completed. Although written for testing a 4k byte memory board, it is readily modified for 8k and 16k boards also. If you are testing a so-called "dynamic" memory board, the computer should be periodically halted from the front panel for a few seconds. This will verify proper operation of the refresh circuitry since normal execution of the test routine would be sufficient to refresh the memory.

If an error is detected during the verify phase, control is passed to an error log routine. This routine stores the address of the error, the correct data byte, and the erroneous data byte in an error log area. (located at the end of the program). At this point, a print routine could take over for a permanent record of all errors. Otherwise, a simple halt could be executed allowing front-panel access to the error log area. By examining a number of error logs, it is usually possible to pinpoint the problem causing errors. For example, if only a single bit is in error and the errors are confined to a 1k block of addresses, then a bad memory chip is the probable cause. If there is a multitude of errors, and the correct data bears no resemblance to the wrong data, there is an addressing problem indicated. ◇

OPERATIONAL AMPLIFIER QUIZ

BY WILLIAM E. PARKER

Assume each of these circuits is an ideal op amp and each input is +1.0 volt dc. Determine the output voltages.



Answers A: −4; B: −8; C: 0; D: −6; E: −2; F: +8; G: +1; H: +2; I: +1;