IDENTIFICATION

PRODUCT CODE: MAINE-11-238-9-0-0
PRODUCT NAME: 0-124K MEMORY EXERCISER
DATE CREATED: FEBRUARY, 1974
MAINTAINER: DIAGNOSTIC GROUP

AUTHOR: JOHN ADAMS

THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO CHANGE WITHOUT NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT CORPORATION. DIGITAL EQUIPMENT CORPORATION ASSUMES NO RESPONSIBILITY FOR ANY ERRORS THAT MAY APPEAR IN THIS MANUAL.

THE SOFTWARE DESCRIBED IN THIS DOCUMENT IS FURNISHED TO THE PURCHASER UNDER A LICENSE FOR USE ON A SINGLE COMPUTER SYSTEM AND CAN BE COPIED (WITH INCLUSION OF DIGITAL'S COPYRIGHT NOTICE) ONLY FOR USE IN SUCH SYSTEM, EXCEPT AS MAY OTHERWISE BE PROVIDED IN WRITING BY DIGITAL.

DIGITAL EQUIPMENT CORPORATION ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS SOFTWARE ON EQUIPMENT THAT IS NOT SUPPLIED BY DIGITAL.

COPYRIGHT © 1974, BY DIGITAL EQUIPMENT CORPORATION
TABLE OF CONTENTS

ABSTRACT

CHAPTER 1 REQUIREMENTS
1.1 EQUIPMENT
1.2 STORAGE
1.3 PRELIMINARY PROGRAMS

CHAPTER 2 LOADING AND STARTING PROCEDURE
2.1 ACT11 OPERATION

CHAPTER 3 SWITCH SETTINGS

CHAPTER 4 SUBROUTINE ABSTRACTS
4.1 SCOPE

CHAPTER 5 ERRORS
5.1 PARITY ERROR

CHAPTER 6 RESTRICTIONS
6.1 STARTING RESTRICTION
6.2 OPERATIONAL RESTRICTION

CHAPTER 7 MISCELLANEOUS
7.1 STACK POINTER
7.2 PASS COUNT
7.3 ERROR COUNT
7.4 DISPLAY REGISTER
7.5 PROGRAM RELOCATION
7.6 POWER FAIL
7.7 EXECUTION TIME
CHAPTER 6 PROGRAM DESCRIPTION

8.1 PROGRAM 2 USER SELECTIONS
8.1.1 PROGRAM 2 USER PARAMETERS
8.1.2 PROGRAM 2 USE

8.2 PROGRAM 3

8.3 PROGRAM 4

8.4 PROGRAM 5

8.5 PROGRAM 6

CHAPTER 7 BRANCH Gobble MOS TEST

9.1 ABSTRACT

9.2 OPERATING PROCEDURE

9.3 ERRORS

9.4 PROGRAM DESCRIPTION
ABSTRACT

PROGRAM DIOMB TESTS CONSECUTIVE MEMORY ADDRESS FROM 000000 TO 7FFFFFF. IT VERIFIES THAT EACH ADDRESS IS UNIQUE (AN ADDRESS TEST) AND THAT EACH MEMORY LOCATION CAN BE READ/WRITTEN RELIABLY (WORST CASE NOISE TESTS). IF MEMORY MANAGEMENT IS AVAILABLE, ALL TESTING IS PERFORMED WITH MEMORY MANAGEMENT ENABLED, (UNLESS DISABLED).

THIS PROGRAM MAY BE USED TO ADJUST MARING MEMORY.

ALSO INCLUDED IS A TOGGLE IN ADDRESS TEST.

ALSO INCLUDED IS THE BRANCH Gobble MCB TEST. NOTE THAT ONLY SECTIONS 9.1 THROUGH 9.4 APPLY TO BRANCH Gobble.
CHAPTER 1
REQUIREMENTS

1.1 EQUIPMENT
THE PDP-11 FAMILY PROCESSOR WITH 9K MEMORY.
OPTIONAL...
KT11-C OR KT11-D MEMORY MANAGEMENT OPTION OR MF11 PARITY OPTION.

1.2 STORAGE
PROGRAM STORAGE - THE PROGRAM USES MEMORY 0-17777.

1.3 PRELIMINARY PROGRAMS
IF PDP-11 FAMILY INSTRUCTION EXERCISER 'DGKE' OR 'DZKOC' KT11-C KT11-D LOGIC TESTS.
CHAPTER 2
LOADING AND STARTING PROCEDURE

LOAD PROGRAM INTO MEMORY USING ABS LOADER.
\[\text{LOAD ADDRESS 200} \]
\[\text{SET 5412 IN DESIRED POSITION (SEE CHAPTER 3).} \]
\[\text{PRESS START.} \]

ASTERISK "*" WILL BE PRINTED AFTER EACH PASS.
"DZMB DONE!" WILL BE PRINTED AFTER 8 PASSES.

PASS COUNT MAY BE MONITORED IN THE DISPLAY REGISTER .11.45 OF LOCATION 756.

NOTE

THE PROGRAM WILL RELOCATE BACK TO 0-4K AND HALT AT 176. PRESS CONTINUE "C" RESUME TESTING.

2.1 ACTIL OPERATION

IF THE PROGRAM IS RUN IN QUICK "ERIF" MODE UNDER ACTIL, THE PROGRAM IS DONE AFTER THE FIRST PASS. ALSO THE PROGRAM DOES NOT RELOCATE "C"-ES THE LOWER 4K OF MEMORY.
CHAPTER 3
SWITCH SETTINGS

SW15=1 OR UP          HALT ON ERROR

NOTE
IF SW15=1 WHEN AN ERROR OCCURS THE PROGRAM WILL HALT, AND THE CORRECT DATA WILL NOT BE LOADED INTO THE FAILING ADDRESS. IF SW15 IS RAISED AFTER THE ERROR TIMEOUT BEGINS, THE PROGRAM WILL HALT WHEN THE TIMEOUT COMPLETES AND THE CORRECT DATA WILL BE LOADED INTO THE FAILING ADDRESS.

SW14=1 OR UP          LOOP SUBTEST
SW13=1 OR UP          INHIBIT ERROR TIMEOUT
SW12=1 OR UP          INHIBIT USE OF MEMORY MANAGEMENT

NOTE
INHIBITING THE USE OF MEMORY MANAGEMENT CAN BE DONE ONLY WHEN THE PROGRAM IS STARTED. IF THE USE OF MEMORY MANAGEMENT IS INHIBITED THE LAST ADDRESS AS TYPED BY THE PROGRAM WILL ONLY REFLECT THE AMOUNT OF MEMORY UP TO 29A (LAST ADDRESS = 160000).

SW11=1 OR UP          INHIBIT SUBTEST ITERATION
SW9=1 OR UP  DISPLAY ERROR COUNT IN DISPLAY REGISTER
SW9=0 OR DOWN DISPLAY PASS COUNT IN DISPLAY REGISTER
SW8=1 OR UP  HALT PROGRAM-UNRELOCATED AND RESTORE LIPERS.
SW5=1 OR UP  INHIBIT PARITY ERROR DETECTION INITIAL STARTUP ONLY.

NOTE

WITH PARITY ERROR DETECTION ENABLED A MEMORY FAILURE WILL CAUSE A PARITY ERROR. THE ERROR PRINTOUT ON A PARITY ERROR DOES NOT TYPE THE GOOD DATA. THUS, A BIT DROP OR PICKUP WILL NOT BE TYPED AS SUCH. IT IS BEST TO RUN THE PROGRAM FOR 1 PASS (UNTIL AN * IS TYPED) WITH PARITY DISABLED, THEN RESTART THE PROGRAM WITH PARITY ENABLED.
CHAPTER 4
SUBROUTINE ABSTRACTS

4.1 SCOPE

THE PROGRAM STORES IN R1 THE PC OF THE LAST "TEST SUCCESSFULLY EXECUTED"
AND MAY BE USED AS AN AID IN DEBUGGING IF THE PROGRAM "BOBB" SUFFERED
OF A HARDWARE FAILURE.
CHAPTER 5

ERRORS

These tests print out the PC where the error was detected, the failing address, the good data, and the bad data I.E.

PC=xxxxxx  ADDRESS aaaaaa  GOOD DATA gggggg  BAD DATA bbbbbbb

The address of the failing location is the true 18 bit physical address.

Note

When testing memory locations 0-17776
The PC typed will be a multiple of 20000 greater than reflected in the program listing.

The address of the bad data is in (r2) -2
The good data in r0
The bad data in r3

The address of good data is in r4 (random data test only). When an error is detected when exercising the memory using the worst case noise patterns, the user should restart the program selecting program (see chapter 6 for details) selecting the appropriate parameters. The user can use the PC and address of the failure to select the proper core bank(s) affected and also the specific pattern. This allows maximum scope capabilities.

6.1 Parity Error
IF THE MEMORY PARITY OPTIONS ARE INSTALLED THE PROGRAM RUNS WITH THE
ACTION ENABLE BIT SET (BIT 0). IF A PARITY ERROR IS DETECTED THE PROGRAM WILL TYPE:

PARITY ERROR

AND SCAN MEMORY FOR THE ADDRESS(ES) CAUSING THE PARITY ERROR(ES). WHEN THE PARITY ERROR IS DETECTED AN ERROR WILL BE Typed AS SHOWN BELOW:

PC=XXXXXX ADDRESS AAAAAA BAD DATA BBBBBB

PRESS CONTINUE OR RESTART TO RESUME TESTING. IF A PARITY ERROR IS NOT DETECTED ON SCAN THE PROGRAM WILL TYPE:

PARITY ERROR NOT FOUND ON SCAN

PC=XXXXXX ADDRESS=AAAAA

WHERE:

AAAAAA=PC AT TIME PARITY ERROR WAS DETECTED.

NOTE

PARITY IS DISABLED WHEN THE PROGRAM IS RELOCATED.
CHAPTER 6

RESTRICTIONS

7.1 STARTING RESTRICTION

PROGRAM MUST NOT BE RELOCATED WHEN RESTARTING.

7.2 OPERATIONAL RESTRICTION

PROGRAM CHECKS CONTIGUOUS MEMORY. IF A PARITY ERROR TRAP OCCURS, THE PROGRAM IS RELOCATED. PROGRAM ACTION IS INDETERMINATE IF PARITY MEMORY IS AVAILABLE OR SELECTED. THE 3 X 3 TEST PATTERN IS INITIATED. MEMORY ONLY. DO NOT POWER FAIL THE PROGRAM WHEN THE PROGRAM IS RUNNING IN HIS MEMORY OR RELOCATED.
CHAPTER 7
MISCELLANEOUS

IF THE PROGRAM HALTS IN THE TRAP INTERRUPT VECTOR AREA, EXAMINE REGISTER 8 (THE STACK PTR); AS CONTAINS THE ADDRESS WHERE THE TRAP OCCURRED. SEE A.35 (S. SPECIFIES THE LAST TEST COMPLETED).

NOTE
THE PCP11:45 WILL DISPLAY THE TRAP VECTOR ADDRESS+4 IN THE ADDRESS LIGHTS. THIS IS A "BUS ERROR" ACTUAL DISPLAY IS 10 IN THE ADDRESS LIGHTS.

7.1 STACK POINTER

THE STACK POINTER IS INITIALLY SET TO 000 AND IS RESET TO THIS VALUE AT THE START OF EACH TEST.

7.2 PASS COUNT

SEVEN PASSES ARE REQUIRED FOR COMPLETION OF THIS PROGRAM. AT EACHassa LIME AN "X" WILL BE PRINTED. THE PASS COUNT WILL BE OBSERVED WHEN THE SWITCH TO THE DISPLAY POSITION. THE PASS COUNT IS STORED IN LOCATION 1000. THE PASS COUNT SHOULD BE "SET" TO 0 FOR THE EVENT THAT THE PROGRAM ENTERS AN UNDEFINED LOOP. NO "SET" TO 0" SHOULD INDICATE THAT THE PROGRAM IS IN ITS RELOCATED LOCATION.
7.4 DISPLAY REGISTER

Either the pass count or the error count is displayed in the display register. The count to be displayed is controlled by the setting of switch 9. Bit 15 of the display register, however, is used as a relocation indicator and is not part of either the pass count or the error count. When bit 15 is on, the program is performing a relocating cycle. When the program is relocated, the special restart procedures of chapter 2 must be followed.

7.5 PROGRAM RELOCATION

When the program is relocated, verification is made that the program has been relocated correctly. If the program cannot be relocated upward, the relocated test phase is bypassed. If an error occurs while relocating the program back to the lower 4K, an error message is typed and the program halts. Continuing the program requires the downward relocation. Downward relocation will be attempted until it is successful or the program is relocated.

7.6 POWER FAIL

The program may be power failed when running. When the power returns the program will continue in sequence.

CAUTION

Program action is undefined if the program is relocated or in DOS memory.

Do not turn power off/on until the message "Power Failed" has been typed. This is because the stack may overflow.

7.7 EXECUTION TIME
EXECUTION TIME IS DEPENDENT ON TYPE OF PROCESSOR, TYPE OF MEMORY, ETC.
AMOUNT OF MEMORY. SOME REPRESENTATIVE TIMES (PER PASS ARE):

- 11 DB WITH 28K MEMORY = 1 MIN.
- 11 DB WITH 96K MEMORY = 3 MIN.
CHAPTER 9

PROGRAM DESCRIPTION

The program verifies each address by writing the value of each address into itself starting at location 20000 and ending at the last address in memory. The value of the last location +2 is typed on the Ity. Next, the values written are verified. To complete the address test, the complement value of each memory address is written starting at the last memory address and ending at address 20000. The written complement values are then verified. The next phase of testing includes reading, writing, and checking memory using several worst case noise test patterns (1 XOR 8, 3 XOR 9, and 8 XOR 13). A subtest is dedicated to checking each pattern. The test proceeds by exercising each bank of memory using the test patterns noted above. Note that with the memory management option installed, that all addresses are written, read, and checked with the memory management enabled. After all memory from 20000 to the last address has been tested, the program relocates to the next 4K memory bank and tests locations 0-17775 using (1 XOR 8). The program then relocates to 40000 (100000 if available) and checks memory using 3 XOR 9, and 8 XOR 13 test pattern. The program then checks memory using random data. This routine moves the program code throughout memory starting at location 20000 and relocates the data by a 32 (decimal) word offset in each subsequent relocation. I.e., first relocation is to 20000, next is to 20000, then 20000, etc. After relocation of the code move is checked against the original code (0-17775). When the random test is complete, the program then successively rotates a '1' bit (ROTI) through all of memory. When all testing is complete, the program relocates to its original position, increments the pass count (location 10000) and restarts beginning with the worst case noise tests. An asterisk (*) will be typed on completion of each pass. And when 9 passes have been completed, the program will type '22MB done' and restart the program beginning with the every address tests.
9.1 PROGRAM & USER SELECTIONS
PCP-11 0-124K MEMORY EXERCISER

PROGRAM DESCRIPTION

THIS PROGRAM IS PROVIDED TO ALLOW THE USER TO SPECIFY CERTAIN "TEST"
PARAMETERS AS SHOWN BELOW:

1. ENABLE/DISABLE PARITY ERROR INTERRUPT'S
2. STARTING BANK NUMBER FOR TEST
3. NUMBER OF 4K BANKS TO TEST
4. PATTERN TO BE USED

NOTE
ALL INPUTS ARE IN OCTAL.

8.1.1 PROGRAM 2 USER PARAMETERS

1. ENABLE PARITY? 1= Y, N= N. TYPE 1 TO ENABLE INTERRUPT ON
   PARITY ERROR. TYPE 0 TO DISABLE INTERRUPT.

2. STARTING BANK (8)? TYPE THE 4K BANK WHERE YOU WISH TO BEGIN
   TESTING.

   TYPE TO START AT  TYPE TO START AT
   000000 000000 000000
   000001 000001 000001
   000010 000010 000010
   000011 000011 000011
   000100 000100 000100
   000101 000101 000101
   000110 000110 000110
   000111 000111 000111
   001000 001000 001000
   001001 001001 001001
   001010 001010 001010
   001011 001011 001011
   001100 001100 001100
   001101 001101 001101
   001110 001110 001110
   001111 001111 001111
   010000 010000 010000
   010001 010001 010001
   010010 010010 010010
   010011 010011 010011
   010100 010100 010100
   010101 010101 010101
   010110 010110 010110
   010111 010111 010111
   011000 011000 011000
   011001 011001 011001
   011010 011010 011010
   011011 011011 011011
   011100 011100 011100
   011101 011101 011101
   011110 011110 011110
   011111 011111 011111
   100000 100000 100000
   100001 100001 100001
   100010 100010 100010
   100011 100011 100011
   100100 100100 100100
   100101 100101 100101
   100110 100110 100110
   100111 100111 100111
   101000 101000 101000
   101001 101001 101001
   101010 101010 101010
   101011 101011 101011
   101100 101100 101100
   101101 101101 101101
   101110 101110 101110
   101111 101111 101111
   110000 110000 110000
   110001 110001 110001
   110010 110010 110010
   110011 110011 110011
   110100 110100 110100
   110101 110101 110101
   110110 110110 110110
   110111 110111 110111
   111000 111000 111000
   111001 111001 111001
   111010 111010 111010
   111011 111011 111011
   111100 111100 111100
   111101 111101 111101
   111110 111110 111110
   111111 111111 111111
   
   NOTE
   TYPE ONLY NUMBERS SHOWN!!!

3. NUMBER OF 4K BANKS TO TEST (8)? TYPE IN OCTAL THE NUMBER OF
   4K BANKS TO TEST.

4. PATTERN #
PROGRAM DESCRIPTION

0 1 XOR 8 TEST PATTERN
3 XOR 9 TEST PATTERN
8 XOR 13 TEST PATTERN
USER CONSTANT
ROTATING 0
ROTATING 1
3 XOR 9 PARITY PATTERN
0, 1, 2, 4, 5 ABOVE

NOTE

PROGRAM WILL NOT ALLOW AN ODD NUMBER OF
4K BANKS TO BE TESTED IF PATTERN 2 OR 7
IS SELECTED. IF PATTERN #3 IS SELECTED
THE PROGRAM WILL REQUEST A CONSTANT.
TYPE A 6 DIGIT OCTAL NUMBER, TO ENTER A
NEW CONSTANT TYPE AN 'A' AND WAIT FOR
THE PROGRAM TO RESPOND. THE STARTING
ADDRESS IS 214.

8.1.2 PROGRAM 2 USE

PROGRAM 2 CAN BE EFFECTIVELY USED TO MAKE PROPER ADJUSTMENTS TO A
SPECIFIC MEMORY BANK AND ALSO TO 'MARGIN' MEMORY. THIS IS SO BECAUSE
THE PROGRAM IS NOT RUNNING IN THE MEMORY BANK(S) BEING
ADJUSTED/MARGINED. THUS ALL MEMORY FROM 0-124K MAY BE
ADJUSTED/MARGINED. PARITY SHOULD BE DESELECTED WHEN MAKING ANY
ADJUSTMENTS PARTICULARLY WHEN TESTING THE FIRST 4K BANK(S).

8.2 PROGRAM 3

THIS PROGRAM IS THE SAME AS PROGRAM 2 WITH THE FOLLOWING EXCEPTIONS:

1. INSTEAD OF NUMBER OF 4K BANKS TO TEST, TYPE NUMBER OF
   256(DECIMAL), 400(OCTAL) WORD BLOCKS TO TEST.

2. DO NOT SELECT PATTERN 2 OR 7.

THE STARTING ADDRESS IS 220.

8.3 PROGRAM 4

PROGRAM 4 CAN BE USED TO WRITE/READ USER DEFINED DATA INTO ANY SINGLE
ADDRESS. THE PROGRAM WRITES THE DATA AND CHECKS IT.
THE PROGRAM WILL REQUEST AN 18 BIT ADDRESS AND IF SWITCH C = 0, A 16 BIT CONSTANT (DATA). IF SWITCH J = 1 THE PROGRAM WILL R: FE THE
5.4 PROGRAM 5

Program 5 is a toggle in memory address test. This test is useful when an address selection failure is suspected involving the first 4K of memory. This program writes the value of each address into itself starting with the lower limit and continuing to the upper limit. After all addresses have been written each address is checked for the correct contents starting with the upper limit and continuing to the lower limit.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>CONTENTS</th>
<th>MNEMONIC</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>012700</td>
<td>MOV #50, R0</td>
<td>GET FIRST ADDRESS</td>
</tr>
<tr>
<td>14</td>
<td>000050</td>
<td></td>
<td>TO TEST</td>
</tr>
<tr>
<td>16</td>
<td>010001</td>
<td>MOV R0, R1</td>
<td>SAVE IN R1</td>
</tr>
<tr>
<td>18</td>
<td>020037</td>
<td>1%</td>
<td>CMP R0, #SWR : CHECK UPPER LIMIT</td>
</tr>
<tr>
<td>20</td>
<td>177507</td>
<td></td>
<td>(SWITCH REGISTER)</td>
</tr>
<tr>
<td>22</td>
<td>000043</td>
<td>BEQ 2%</td>
<td>BRANCH IF AT UPPER LIMIT</td>
</tr>
<tr>
<td>24</td>
<td>010010</td>
<td>MOV R0, (RO)</td>
<td>LOAD VALUE INTO ADDRESS</td>
</tr>
<tr>
<td>26</td>
<td>005720</td>
<td>TST (RO)+</td>
<td>STEP TO NEXT ADDRESS</td>
</tr>
<tr>
<td>30</td>
<td>000072</td>
<td>BR 1%</td>
<td>LOOP UNTIL DONE</td>
</tr>
<tr>
<td>32</td>
<td>010034</td>
<td>2%</td>
<td>MOV R0, R4 : SAVE UPPER LIMIT</td>
</tr>
<tr>
<td>34</td>
<td>020001</td>
<td>3%</td>
<td>CMP R0, R1 : CHECK IF AT LOWER LIMIT</td>
</tr>
<tr>
<td>36</td>
<td>001767</td>
<td>BEQ 1%</td>
<td>BRANCH IF DONE</td>
</tr>
<tr>
<td>40</td>
<td>024000</td>
<td>CMP -(RO), RO</td>
<td>CHECK DATA WRITTEN</td>
</tr>
<tr>
<td>42</td>
<td>001774</td>
<td>BEQ 3%</td>
<td>BRANCH IF OK</td>
</tr>
<tr>
<td>44</td>
<td>000000</td>
<td>HALT</td>
<td>ERROR</td>
</tr>
<tr>
<td>46</td>
<td>000072</td>
<td>BR 3%</td>
<td>LOOP BACK</td>
</tr>
</tbody>
</table>

After toggling the program LA=10 **SET UPPER LIMIT**, START.

NOTE:

The upper limit address obtained from the switch register may be changed during program operation. However, occasionally the program may halt because of "switch bounce." The best procedure when changing limits is to stop the program, make the change and...
### PROGRAM 5

PROGRAM 5 IS ALSO A TOGGLE IN PROGRAM TO BE USED WITH PROGRAM 6. FOR MORE COMPLETE ADDRESS TESTING. THIS PROGRAM WRITES THE COMPLEMENT VALUE OF EACH ADDRESS INTO ITSELF STARTING WITH THE UPPER LIMIT CONTINUING TO THE LOWER LIMIT. AFTER ALL ADDRESSES HAVE BEEN WRITTEN, EACH ADDRESS IS CHECKED FOR THE CORRECT CONTENTS STARTING WITH THE LOWER LIMIT ADDRESS AND CONTINUING TO THE UPPER LIMIT TOGGLE IN THE FOLLOWING PATCHES TO PROGRAM 5 ABOVE.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>CONTENTS</th>
<th>MNEMONIC</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td>100</td>
<td>BEQ #5</td>
<td>CHANGE LOWER LIMIT</td>
</tr>
<tr>
<td>000022</td>
<td>MOV $R2</td>
<td>BRANCH TO PROGRAM 6</td>
<td></td>
</tr>
<tr>
<td>000044</td>
<td>ADD $R1</td>
<td>SET UPPER LIMIT</td>
<td></td>
</tr>
<tr>
<td>000066</td>
<td>CMP (R2)</td>
<td>COMPLEMENT ADDRESS</td>
<td></td>
</tr>
<tr>
<td>000088</td>
<td>CMB R2,R1</td>
<td>CHECK IF AT LOWER LIMIT</td>
<td></td>
</tr>
<tr>
<td>0000A0</td>
<td>BEQ #4</td>
<td>LOOP UNTIL DONE</td>
<td></td>
</tr>
<tr>
<td>0000C2</td>
<td>SUB $R1</td>
<td>CHECK IF AT UPPER LIMIT</td>
<td></td>
</tr>
<tr>
<td>0000E4</td>
<td>MOV $R2</td>
<td>GO TO PROGRAM 5 IF DONE</td>
<td></td>
</tr>
<tr>
<td>000106</td>
<td>ADD $R3</td>
<td>SET VALUE OF ADDRESS</td>
<td></td>
</tr>
<tr>
<td>000128</td>
<td>CMB R3,R4</td>
<td>COMPLEMENT VALUE</td>
<td></td>
</tr>
<tr>
<td>00014A</td>
<td>BEQ #6</td>
<td>CHECK ADDRESS</td>
<td></td>
</tr>
<tr>
<td>00016C</td>
<td>MOV $R3</td>
<td>B impover</td>
<td></td>
</tr>
<tr>
<td>00018E</td>
<td>ADD $R2</td>
<td>ERROR</td>
<td></td>
</tr>
<tr>
<td>0001AE</td>
<td>CMB R2,R1</td>
<td>DIREC</td>
<td></td>
</tr>
<tr>
<td>0001CE</td>
<td>BNE #5</td>
<td>DIRECT ACCESS</td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 9
BRANCH GIBBLE MOS TEST

9.1 ABSTRACT

THE BRANCH GIBBLE PROGRAM IS USED TO TEST MOS MEMORY. CONTIGUOUS LOCATIONS ARE TESTED BETWEEN TWO LIMITS IN A MINIMUM 8K, MAXIMUM 16K MEMORY MACHINE. IF PARITY IS AVAILABLE IT IS ENABLED.

9.2 OPERATING PROCEDURE

1. LOADING: LOAD THE DZMBG PROGRAM INTO MEMORY USING THE ABSOLUTE LOADERS.

2. STARTING: LOAD ADDRESS 070 AND PRESS THE START BUTTON.

3. THE PROGRAM WILL FIRST IDENTIFY ITSELF ON TTY:
   
   BRANCH GIBBLE

4. THEN THE ABSOLUTE LOADER WILL BE SAVED.

5. A CHECK WILL BE MADE FOR PARITY REGISTERS. IF NONE ARE FOUND THE MESSAGE:
   
   NO PARITY

   WILL BE TYPED TO THE USER. IF PARITY IS FOUND IT IS TURNED ON, AND THE MESSAGE:
   
   PARITY ENABLED

   WILL BE TYPED TO THE USER. THIS WILL BE FOLLOWED BY A LIST OF THE UNIBUS ADDRESSES OF THE PARITY REGISTERS FOUND AND ENABLED.
6. The user will then be asked if he wants the Fencing Test to
BE RUN USING MEMORY MANAGEMENT.

USE KTII (Y OR N)

IF THE USER TYPES Y THEN MEMORY MANAGEMENT WILL BE USED DURING THE PENDING TEST. IF HE TYPES N THEN MEMORY MANAGEMENT WILL NOT BE USED. TYPING ANYTHING ELSE OTHER THAN Y OR N WILL CAUSE THE QUESTION TO BE REPEATED.

7. THE USER WILL THEN BE ASKED TO GIVE THE LIMITS OF THE TEST SPAN:

HIGH LIMIT

AND:

LOW LIMIT

RESTRICTIONS ON THE USER'S RESPONSE ARE:

A. THE NUMBERS MUST BE VALID 16-BIT, 8-DECIT OCTAL ADDRESSES, REAL NOT VIRTUAL.

B. THE NUMBERS SHOULD BE MULTIPLES OF 100. (OCTAL)

C. THE HIGH LIMIT MUST BE GREATER THAN THE LOW LIMIT.

D. IF MEMORY MANAGEMENT IS NOT USED. HIGH LIMIT MUST BE LESS THAN OR EQUAL TO 163840.

E. HIGH LIMIT CAN BE 1 + THE HIGHEST REAL CORE ADDRESS. FOR EXAMPLE, IN AN 8K MACHINE, HIGH LIMIT CAN EQUAL 40960.

VIOLATIONS TO THESE RESTRICTIONS WILL BE DEALT WITH IN THIS WAY:

A. A QUESTION MARK AND THE PROMPT WILL BE ISSUED:

? 

THE USER IS THEN EXPECTED TO INPUT THAT LIMIT AGAIN: THIS TIME CORRECTLY.

B. WHAT EVER THE LAST TWO OCTAL DIGITS OF THE NUMBER WHICH THE USER TYPED THEY WILL BE ASSEMBLED AS ZEROS.

C. THE USER WILL BE ASKED FOR OTHER LIMITS BY REPEATING THIS STEP 7. BEFORE STEP 7 IS REPEATED
BRANCH JUMBLE MOS TEST

Will be typed.

3. Same as 3.

8. The test starts. The test will stop indefinitely between the two limits unless the user halts the program or an error is encountered. An asterisk is typed at the beginning of each pass mode.

9. To stop the test running and start another hit the halt switch and return step 2. Any test but the first will not include step 4.

10. To stop the test and restore the loader, hit the halt switch, load address 162 and start. When the loader is restored the program will halt at location 200.

11. To stop the test and start the 0-124k memory test hit the halt switch, load address 200, and start.

12. Data lights. The data lights will display the current location being tested during a branch jumble test. When a MOS memory failure occurs these lights will contain virtual (16-bit) address "near" the failure.

9.3 Errors

1. Errors in operating the program are described in operating procedure 9.2.

2. If a parity error is detected the user is told the PC at the time of the error:

   PARITY ERROR
   PC=xxxxxx

   Then the scan is made through all of memory to try to force the error to arise again. If it is not found the message

   SCAN COMPLETE

   IS TYPED AND THE TEST IS RESTARTED.

   If the error is detected on the scan then the user is given the address of the location causing the parity error and the contents of that location:

   XXXXXX had bad data XXXXXX

   If memory management was off during the scan for the error
KT11 OFF

IS TYPED.

IF MEMORY MANAGEMENT WAS ON DURING THE SCAN:

KT11 ON PAR=XXXXXX

IS TYPED, WHERE THE PAR (PAGE ADDRESS REGISTER) GIVEN IS THAT
PAR WHICH SHOULD BE USED IN RELOCATING THE VIRTUAL ADDRESS
GIVEN FOR THE ERROR ONTO A REAL CORE ADDRESS. THE METHOD FOR
THIS RELOCATION IS GIVEN IN THE NOTE BELOW.

AFTER ANY PARITY IS ENCOUNTERED AND THE USER NOTIFIED, THE
TEST WILL BE RESTARTED.

3. WHENEVER THE BRANCH Gobble TEST BRINGS OUT AN ERROR IN MOS
MEMORY IT MAY SURFACE AS A PARITY AND BE HANDLED AS DESCRIBED
ABOVE. OTHERWISE THE ADDRESS (VIRTUAL IF MEMORY MANAGEMENT
IS ON) IN THE DATA LIGHTS WILL DESIGNATE THE VICINITY OF THE
ERROR. IF MEMORY MANAGEMENT IS ON RELOCATE THE ADDRESS IN
THE DATA LIGHTS IN THE MANNER DESCRIBED IN THE NOTE BELOW.

NOTE

TO COMPUTE THE REAL ADDRESS OF AN
ADDRESS RELOCATED BY MEMORY MANAGEMENT,
ADD THE LOW ORDER 13-BITS OF THE VIRTUAL
ADDRESS TO THE CORRESPONDING PAR
SHIFTED, 6 BITS TO THE LEFT:

VIRTUAL ADDRESS = 0 00X XXX XXX XXX XXX
PAR = YYY YYY YYY YYY 000 000
REAL ADDRESS = ZZZ ZZZ ZZZ ZZZ ZZZ ZZZ

TO DETERMINE WHICH PAR TO USE REMEMBER
THAT ON KERNEL SPACE IS USED IN ANY TEST
HERE, TAKE THE HIGH ORDER 3-BITS OF THE
VIRTUAL ADDRESS AND USE THEM TO
DESIGNATE THE KIPAR TO USE. FOR
INSTANCE, IF THE VIRTUAL ADDRESS IS
031676 USE KIPAR1 BECAUSE THE UPPER 3
BITS OF THE VIRTUAL ADDRESS ARE 0011.

4. IF AN ERROR CONDITION ARISES, EITHER AS A PARITY ERROR OR ONE

NOT APPARENT SUCH AS THE PROGRAM HALTS OR IT IS CLEAR FROM
THE ADDRESS AND DISPLAY LIGHTS THAT THE PROGRAM IS NOT
RUNNING ITS NORMAL COVERAGE, THE USER CAN ENTER CONSOLE MODE AND
EXAMINE THE CONTENTS OF THE MEMORY LOCATIONS STARTING AT THE
ADDRESS IN THE DISPLAY REGISTER. THE CONTENTS OF THESE LOCATIONS SHOULD BE COMPARED TO THE CONTENTS IN
LISTINGS: OF LOCATIONS 15226 THROUGH 15334 WITH THE EXCEPTIONS: 15328 IS UNDETERMINABLE AND 15304 SHOULD CONTAIN EITHER 15370 OR 35370. IN THIS WAY THE USER SHOULD BE ABLE TO DETERMINE WHICH BITS WERE LOST IN WHAT WORDS.

9.4 PROGRAM DESCRIPTION

THIS VERSION OF THE BRANCH GOBBLE TEST IS TAKEN ALMOST DIRECTLY FROM THE DZORK-4 INSTRUCTION EXERCISER WHICH CONTAINED THE ORIGINAL BRANCH GOBBLE TEST. WHAT HAS BEEN DONE HERE IS TO GIVE THAT TEST AN INTERFACE TO THE USER AND MEMORY MANAGEMENT FACILITIES. THESE ADDITIONS HAVE BEEN DONE IN A WAY WHICH ALLOWS THE TEST TO RUN AS IT DID IN ITS ORIGINAL FORM. DATA IS COLLECTED FROM THE USER AND IF MEMORY MANAGEMENT IS NEEDED IT IS SET UP AND THAT TEST IS ALLOWED TO RUN BETWEEN THE DESIGNATED LIMITS.

:\MLIST MD,M3
:\LIST ME
:\ABS $TYPEx
:\CALL $TYPEx
\TITLE TEST DZMB-G 0-124K MEMORY EXERCISER
\SBITL STARTING INST & DEFINITIONS
\COPYRIGHT 1973 DIGITAL EQUIPMENT CORP., MAYNARD, MASS.

:THIS TEST CHECKS THAT ALL MEMORY ADDRESSES ARE UNIQUE USING ADDRESS TESTS AND CHECKS DATA RELIABILITY OF MEMORY USING WORST CASE NOISE TEST PATTERN: A RANDOM # PATTERN (PROGRAM CODE RELOCATED), A ROTATING 0 AND ROTATING 1 PATTERN. ALSO INCLUDED ARE USER TESTS WHICH CAN BE USED TO TEST SPECIFIED SEGMENTS OF MEMORY USING THE PATTERNS MENTIONED ABOVE. ADDITIONALLY A 28 WORD TOGGLE IN PROGRAM IS DOCUMENTED (SEC 9.5 OF THE DOCUMENT) WHICH CAN BE USED IF AN ADDRESSING MALFUNCTION IS SUSPECTED INVOLVING THE FIRST 4K OF MEMORY.

A MOS MEMORY TEST HAS BEEN ADDED, THE BRANCH GOBBLE ROLINE.

THE PROGRAM MAY BE POWER FAILED WHEN RUNNING. THE PROGRAM WILL PRINT A MESSAGE (POWER FAILED) AND CONTINUE IN SEQUENCE WHEN THE POWER COMES BACK UP. **CAUTION** DO NOT POWER FAIL THE PROGRAM IF THE PROGRAM IS IN A MOS MEMORY OR IF THE PROGRAM IS RELOCATED.

LOADING AND STARTING INSTRUCTIONS

LOAD ADDRESS 20D AND START

NOTE: PROGRAM WILL RUN WORST CASE TEST PATTERNS IN LOWEST 4K

:THE PROGRAM CANNOT BE RESTARTED AT 20D IF RELOCATED. TO PREVENT RELOCATION FROM OCCURRING DEPOSIT 20D INTO LOCATION 42 (NOT NECESSARY IF LOADED VIA ACTII). THIS ACTION WILL PREVENT RELOCATION AND ALSO INHIBIT TESTING MEMORY IN LOWEST 4K.

:THIS PROGRAM ALSO RELOCATES THE ABS AND BOOT LOADERS TO ALLOW TESTING OF MEMORY TO RESTORE THE LOADERS RESTART AT 162.

:STACK POINTER IS SET AT 50D
### M03

**AN ASTRISK '*' WILL BE PRINTED ON COMPLETION OF EACH PASS. A '*'**

THE PROGRAM NAME WILL BE PRINTED WHEN TEST IS COMPLETE.

#### GENERAL REGISTER ASSIGNMENTS

<table>
<thead>
<tr>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>00000</td>
</tr>
<tr>
<td>R1</td>
<td>00001</td>
</tr>
<tr>
<td>R2</td>
<td>00002</td>
</tr>
<tr>
<td>R3</td>
<td>00003</td>
</tr>
<tr>
<td>R4</td>
<td>00004</td>
</tr>
<tr>
<td>R5</td>
<td>00005</td>
</tr>
<tr>
<td>SP</td>
<td>00006</td>
</tr>
<tr>
<td>PC</td>
<td>00007</td>
</tr>
<tr>
<td>P0</td>
<td>00008</td>
</tr>
<tr>
<td>R10</td>
<td>00009</td>
</tr>
<tr>
<td>R11</td>
<td>00010</td>
</tr>
<tr>
<td>R12</td>
<td>00011</td>
</tr>
<tr>
<td>R13</td>
<td>00012</td>
</tr>
<tr>
<td>R14</td>
<td>00013</td>
</tr>
<tr>
<td>R15</td>
<td>00014</td>
</tr>
</tbody>
</table>

#### STATUS REGISTER (PSW) BIT ASSIGNMENTS

<table>
<thead>
<tr>
<th>Bit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>0</td>
</tr>
<tr>
<td>V</td>
<td>1</td>
</tr>
<tr>
<td>Z</td>
<td>2</td>
</tr>
<tr>
<td>N</td>
<td>3</td>
</tr>
<tr>
<td>T</td>
<td>4</td>
</tr>
<tr>
<td>PRTY7</td>
<td>5</td>
</tr>
<tr>
<td>PRTY4</td>
<td>6</td>
</tr>
<tr>
<td>KM</td>
<td>7</td>
</tr>
<tr>
<td>SM</td>
<td>8</td>
</tr>
<tr>
<td>LM</td>
<td>9</td>
</tr>
<tr>
<td>PK</td>
<td>10</td>
</tr>
<tr>
<td>PS</td>
<td>11</td>
</tr>
<tr>
<td>PU</td>
<td>12</td>
</tr>
<tr>
<td>RE</td>
<td>13</td>
</tr>
</tbody>
</table>

#### VECTOR ADDRESSES

<table>
<thead>
<tr>
<th>Vector</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERRVEC</td>
<td>00004</td>
</tr>
<tr>
<td>REVEC</td>
<td>00006</td>
</tr>
<tr>
<td>SVEC</td>
<td>00008</td>
</tr>
<tr>
<td>TVEC</td>
<td>00010</td>
</tr>
<tr>
<td>BPTVEC</td>
<td>00014</td>
</tr>
<tr>
<td>IOTVEC</td>
<td>00018</td>
</tr>
<tr>
<td>Pivec</td>
<td>00020</td>
</tr>
<tr>
<td>PVEC</td>
<td>00022</td>
</tr>
<tr>
<td>TVEC</td>
<td>00024</td>
</tr>
<tr>
<td>TRAPVEC</td>
<td>00026</td>
</tr>
<tr>
<td>TVC</td>
<td>00028</td>
</tr>
<tr>
<td>ITKVEC</td>
<td>00030</td>
</tr>
<tr>
<td>ITPVEC</td>
<td>00032</td>
</tr>
<tr>
<td>PIRVEC</td>
<td>00034</td>
</tr>
<tr>
<td>FPEVEC</td>
<td>00036</td>
</tr>
<tr>
<td>MMVEC</td>
<td>00038</td>
</tr>
<tr>
<td>PS</td>
<td>00040</td>
</tr>
<tr>
<td>SLR</td>
<td>00042</td>
</tr>
<tr>
<td>PIQ</td>
<td>00044</td>
</tr>
<tr>
<td>UBREAK</td>
<td>00046</td>
</tr>
</tbody>
</table>

#### REGISTER ADDRESSES

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>PS</td>
<td>177776</td>
</tr>
<tr>
<td>SLR</td>
<td>177774</td>
</tr>
<tr>
<td>PIQ</td>
<td>177772</td>
</tr>
<tr>
<td>UBREAK</td>
<td>177770</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>177776</td>
<td>PS</td>
</tr>
<tr>
<td>177774</td>
<td>SLR</td>
</tr>
<tr>
<td>177772</td>
<td>PIQ</td>
</tr>
<tr>
<td>177770</td>
<td>UBREAK</td>
</tr>
<tr>
<td>Location</td>
<td>Value</td>
</tr>
<tr>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>1298</td>
<td>177560</td>
</tr>
<tr>
<td>1299</td>
<td>177562</td>
</tr>
<tr>
<td>1300</td>
<td>177564</td>
</tr>
<tr>
<td>1301</td>
<td>177566</td>
</tr>
<tr>
<td>1302</td>
<td>177567</td>
</tr>
<tr>
<td>1303</td>
<td>177570</td>
</tr>
<tr>
<td>1304</td>
<td>177575</td>
</tr>
<tr>
<td>1305</td>
<td>177577</td>
</tr>
<tr>
<td>1306</td>
<td>177578</td>
</tr>
<tr>
<td>1307</td>
<td>177579</td>
</tr>
<tr>
<td>1308</td>
<td>177580</td>
</tr>
<tr>
<td>1309</td>
<td>177581</td>
</tr>
<tr>
<td>1310</td>
<td>177582</td>
</tr>
<tr>
<td>1311</td>
<td>177583</td>
</tr>
<tr>
<td>1312</td>
<td>177584</td>
</tr>
<tr>
<td>1313</td>
<td>177585</td>
</tr>
<tr>
<td>1314</td>
<td>177586</td>
</tr>
<tr>
<td>1315</td>
<td>177587</td>
</tr>
<tr>
<td>1316</td>
<td>177588</td>
</tr>
</tbody>
</table>

### INITIAL STACK POINTER SETTING

STKPTR=500

### MISCELLANEOUS BIT ASSIGNMENTS

BIT5=  100  
BIT4=  010000  
BIT3=  0020000  
BIT2=  0100000  
BIT1=  0001000  
BIT0=  0000000  
BIT6=  0000100  
BIT7=  0001000  
BIT8=  0000000  
BIT9=  0000000  
BIT10= 0000000  
BIT11= 0000000  
BIT12= 0000000  
BIT13= 0000000  
BIT14= 0000000  
BIT15= 0000000

### MEMORY MANAGEMENT REGISTER ADDRESS ASSIGNMENTS

SR0=177572 : ADDRESS OF MEM MGMT REGISTER SR0
SR1=177574 : ADDRESS OF MEM MGMT REGISTER SR1
SR2=177576 : ADDRESS OF MEM MGMT REGISTER SR2
SR3=177578 : ADDRESS OF MEM MGMT REGISTER SR3
SR4=177580 : ADDRESS OF MEM MGMT REGISTER SR4
SR5=177582 : ADDRESS OF MEM MGMT REGISTER SR5
SR6=177584 : ADDRESS OF MEM MGMT REGISTER SR6
SR7=177586 : ADDRESS OF MEM MGMT REGISTER SR7
SR8=177588 : ADDRESS OF MEM MGMT REGISTER SR8
SR9=177590 : ADDRESS OF MEM MGMT REGISTER SR9
SR10=177592 : ADDRESS OF MEM MGMT REGISTER SR10
SR11=177594 : ADDRESS OF MEM MGMT REGISTER SR11
SR12=177596 : ADDRESS OF MEM MGMT REGISTER SR12
SR13=177598 : ADDRESS OF MEM MGMT REGISTER SR13
SR14=177600 : ADDRESS OF MEM MGMT REGISTER SR14
SR15=177602 : ADDRESS OF MEM MGMT REGISTER SR15
SR16=177604 : ADDRESS OF MEM MGMT REGISTER SR16
SR17=177606 : ADDRESS OF MEM MGMT REGISTER SR17
SR18=177608 : ADDRESS OF MEM MGMT REGISTER SR18
SR19=177610 : ADDRESS OF MEM MGMT REGISTER SR19
SR20=177612 : ADDRESS OF MEM MGMT REGISTER SR20
SR21=177614 : ADDRESS OF MEM MGMT REGISTER SR21
SR22=177616 : ADDRESS OF MEM MGMT REGISTER SR22
SR23=177618 : ADDRESS OF MEM MGMT REGISTER SR23
SR24=177620 : ADDRESS OF MEM MGMT REGISTER SR24
SR25=177622 : ADDRESS OF MEM MGMT REGISTER SR25
SR26=177624 : ADDRESS OF MEM MGMT REGISTER SR26
SR27=177626 : ADDRESS OF MEM MGMT REGISTER SR27
SR28=177628 : ADDRESS OF MEM MGMT REGISTER SR28
SR29=177630 : ADDRESS OF MEM MGMT REGISTER SR29
SR30=177632 : ADDRESS OF MEM MGMT REGISTER SR30
SR31=177634 : ADDRESS OF MEM MGMT REGISTER SR31

### INSTRUCTION EQUATIONS

104400  HLTT=TRAP
104000  SCOE=ENT

### MISC. EQUATIONS

RW=8
UP=0
The subroutine where is called before any test is run to see if branch juggle relocated the entire first four K of core into the second for K and didn't relocate everything back. If this is the case where will put the program back into the first four K and return to the beginning of the test the user designated by loading his starting address. Note that this routine will not relocate the program if it has been moved by any other subprogram except the branch juggle program. The relocation of the program to the first four K is inacted by the user in the same way it was in the DQMBF and DQMBE versions of the 0-124. Yes, if it has been relocated by the part of this test which was taken from DQMBF, then this is all the program except branch juggle.

HERE: 1ST 31120

;3:
MOV SP, SAV/PCB+2000
JSR NS.RELOC+2003
WORD+20000
WORD+20000

;8:
MOV SAV, SP
SLP 31130
RTS

;18:

FONE: MOV $503, SP ;STARTING ADDRESS TO RELOCATE LOADERS.
JSR PC WHERE

PTAD: MOV $500, SP ;STARTING ADDRESS OF 0-124K MEMORY EXERCISER.
JSR PC WHERE
CO4

STARTING INST S DEFINITIONS

:ROUTINE TO SAVE REGISTERS ON THE STACK
:CALLED BY SAVE MACRO OR JSR PS:PSAV

S$AVR: MOV .SP+,18 ;SAVE RETURN PC
MOV $6646,(SP)
MOV $449,(SP)
MOV $837,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
MOV $A07,(SP)
D04

MOV R2, R3 ; SAVE KIPRD-KIPDR7

SDB R2 18 ; GET ADDRESS OF KIPARC

MOV RO:+,-.SP. ; SAVE KIPRD-KIPPAR7

SDB R3, 28 ; SAVE STACK PTR IN FOLLOWING LOCATIONS

MOV SP, (PC)+ ; CONTAINS STACK PTR AFTER POWER FAIL.

CNOO ; SET POWER FAIL VECTOR TO PUP ROLTINE

HALT

JP ROUTINE

NOP

MOV 2APFSTK, SP ; SET STACK PTR

MMAVA ; CHECK IF MEM MGMT IS AVAILABLE

BEJ 48

MOV KIPAR7+2, R0 ; GET ADDRESS OF KIPAR7+2

MOV R2, R3

SDB R3, 18 ; RESTORE KIPAR7-KIPARC

MOV (SP)+,-(RD) ; RESTORE KIPRD7-KIPRD7

MOV R2, R3

SDB R3, 28 ; RESTORE SR2

MOV (SP)+, 25SP2 ; RESTORE SRO

MOV (SP)+, 25SRO ; RESTORE SRO

TST MAVA ; CHECK IF PARITY REGISTERS ARE ENABLED

BEG 55 ; BRANCH IF NOT

PC, MAMA ; BRANCH IF PARITY REGISTERS

JSR ; RESTORE REGISTERS FROM STACK

PC, GRESTR

JSR ; SET POWER FAIL TRAP TO PDWN ROUTINE

PC, PDWNT

NOP ; GO TO PRINT ROUTINE

PURFAIL

R5, SPRINT

RETURN

PURFAIL: ASCIZ 'POWER FAILED'

SUBTIL ; TAGS 3 PRINT ROUTINE

EVEN

ICOUNT: WORD 0 ; CONTAINS ITERATION PATTERN

ERCHT: 0 ; CONTAINS ERROR COUNT

LCTISP: 0 ; CONTAINS DISPLAY REGISTER IMAGE

MMAVA: 0 ; MEM MGMT AVAILABLE INDICATOR

GNOT AVAIL.-1=AVAIL

BELOP: WORD 0 ; CONTAINS RELOCATION FACTOR

COUNT: WORD 0 ; TEMPORARY WORKING SOCKET
; ROUTINE TO PASS MESSAGE ADDRESS TO TE ROUTINE BELOW
; CALL: "NS", AS PRINT MESSAGE ADDRESS
; "PRINT": MOV .RS + 1, R5
; ;SET MESSAGE ADDRESS
; "ADD": ADD .RS
; ;ADD RELOCATION FACTOR
; "PUSH ASC ON THE STACK"
; "RET": JSR .RTS
; ;CALL TYPE ROUTINE
; "TE": RTS RS
; RETURN

; ROUTINE TO TYPE ASCII MESSAGE MESSAGE MUST TERMINATE WITH A 0 BYTE.
; THE ROUTINE WILL INSERT A NUMBER OF NULL CHARACTERS AFTER A LINE FEED.
; CALL: "TYPE"
; "MESADR": MESA IS FIRST ADDRESS OF ASCII STRING

; TAGS USED BY THE TYPE ROUTINE BELOW
; "CONTAINS NULL CHARACTER"
; "FILLER": BYTE R5
; "CONTAINS # OF FILLER CHARACTERS"
; "TEPL": BYTE R5
; "CONTAINS TELEPRINTER AVAILABLE FLAG"
; "C 377 = AVAIL/NOT AVAIL"
; "CONTAINS KEYBOARD AVAILABLE FLAG"
; "TEPLS": BYTE R5
; "ADDRESS OF TELEPRINTER STATUS REGISTER"
; "RS": WORD R5
; "SPO SP": ;ADDRESS OF TELEPRINTER DATA BUFFER
; "SAVE RO"
; "SET MESSAGE ADDRESS"
; "ADJUST RETURN PC"

; BS: MOV .RS + 1, R5
; ;PUSH CHARACTER TO BE TYPE INTO STACK
; "BRANCH IF NOT THE TERMINATOR"
; "PDP TERMINATOR CHAR OFF THE STACK"
; "RESTORE RO"
; "RETURN TO CALLER"

; BS: MOV .RS + 1, R5
; "TYPE CHARACTER"
; "CHECK IF CHARACTER WAS A LINE FEED"
; "BRANCH IF NOT LINE FEED"
; "SET # OF FILLERS REQUIRED AND FILLER CHARACTER"

; LE: DEC B
; "DECREMENT FILLERS REQ. COUNT"
; "BRANCH IF NO MORE FILLERS ARE REQUIRED"

; BS: MOV .RS + 1, R5
; "TYPE FILLER CHARACTER"

; TS: TSB .SP, BPSB
; "WAIT FOR OUTPUT DEVICE"
; "OUTPUT CHARACTER"

; ERROR TRAP SERVICE ROUTINE
; "CHECK IF HALT ON ERROR"
; "BRANCH IF NO HALT ON ERROR"
; "HALT"
<table>
<thead>
<tr>
<th>Address</th>
<th>Assembly Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000H</td>
<td>MOV 16, SP1, RO</td>
</tr>
<tr>
<td></td>
<td>MOV #5, R3</td>
</tr>
<tr>
<td></td>
<td>CLR R2</td>
</tr>
<tr>
<td></td>
<td>ROL RO</td>
</tr>
<tr>
<td></td>
<td>ROL R2</td>
</tr>
<tr>
<td></td>
<td>ADD #260, R2</td>
</tr>
<tr>
<td></td>
<td>MOV R8, R2, #8</td>
</tr>
<tr>
<td></td>
<td>JSR R8, SPRINT</td>
</tr>
<tr>
<td></td>
<td>MOV SP1, SP+1</td>
</tr>
<tr>
<td></td>
<td>JSR PC, $S</td>
</tr>
<tr>
<td></td>
<td>MOV VALUE, -(SP)</td>
</tr>
<tr>
<td></td>
<td>:routine to type octal value pushed onto stack</td>
</tr>
</tbody>
</table>
|         | RTS PC, 06A | ; RETUR
PLACE: .WORD 0
.SBTL MEMORY ADDRESS TESTS

THIS TEST ADDRESS MEMORY UP TO 128K AND PROVES 'UNIQUENESS' OF A
MEMORY ADDRESS IN A 32K SEGMENT. THE TEST WRITES INTO EACH MEMORY
ADDRESS THE VALUE OF THAT ADDRESS AND THEN CHECKS FOR THE CORRECT?
DATA IN EACH ADDRESS.

THE TWELVE MOST SIGNIFICANT BITS OF THE LAST AVAILABLE MEMORY ADDRESS
IS STORED IN RS.

STARTING INSTRUCTIONS:
LOAD ADDRESS=200
PRESS START
STACK POINTER IS AT 500
**********RESTART AT 162 TO RESTORE LOADERS**************

MEMORY ADDRESS TEST
START: MOV #STARTI, &212 ;CHANGE START ADDRESS
MOV #STKPTR, SP ;SET UP STACK PTR
JSR PC SLOR ;GO SAVE MONITOR & LOADERS
JSR RS, EPRINT ;GO TO PRINT ROUTINE
RESLR

PERSTART: .WORD 0 ;CONTAINS RESTART ADDRESS AFTER PAR ERR
CLR #ICNT ;CLEAR PASS COUNT
CLR #RCOF ;CLEAR REALLOCATION FACTOR
MOV #PDWN, #MPFVEC+2 ;SET POWER FAIL TRAP VECTOR

:CHECK IF MEMORY MANAGEMENT IS AVAILABLE
CLR MMAYA ;CLEAR MGMT AVAILABLE INDICATOR
BIT #0112, #SWR ;CHECK IF TO RUN WITH MGMT
BNE 1$ ;DO NOT USE MGMT IF SW12 WAS SET

1$: JSR PC, MGMT ;SET INDICATOR TO -1 IF AVAILABLE

ROUTINE TO WRITE VALUE OF MEMORY ADDRESS INTO MEMORY ADDRESS
FOR EXAMPLE ROUTINE WRITES 200000 INTO LOCATION 200000

:RTUP: MOV #DONEC, #ERRORVEC ;SET TIME OUT TRAP VECTOR
MOV PC, R1 ;LOAD TRACE REGISTER
JSR PC, LDMCC
MOV #3MABTO, #MVMVEC ;SET MGMT ABORT VECTOR
MOV #200000, R2 ;FIRST ADDRESS
MOV R3, R3 ;LOAD CONSTANT
MOV R4,(R2)+ ;WRITE VALUE OF ADDRESS INTO ADDRESS
ADD R4, R3 ;NEXT VALUE
BR .-6 ;WRITE UNTIL DONE

1$: MOV #PC, R1
JSR PC, LDMCC
MOV #2MABTO, #MVMVEC ;SET MGMT ABORT VECTOR
MOV #200000, R2 ;FIRST ADDRESS
MOV R3, R3 ;LOAD CONSTANT
MOV R4,(R2)+ ;WRITE VALUE OF ADDRESS INTO ADDRESS
ADD R4, R3 ;NEXT VALUE
BR .-6 ;WRITE UNTIL DONE

1$: MOV #PC, R1
:ROUTINE TO CHECK THAT VALUE OF MEMORY ADDRESS WAS WRITTEN CORRECTLY
MOV PC,R1
LOAD TRACE REGISTER
MOV #20000.R2
SET R2
MOV #DONE1.3.*ERROR
SET TIME OUT TRAP
MOV R2,R0
SUB #2,R0
SUBTRACT 2
JSR PC;DDMM
ADD #2,R0
SET WRITTEN VALUE
MO. (R2)+,R3
CHECK
BEQ 1S
HLT
ERROR TO DETERMINE WHICH ADDRESS WAS WRITTEN IMPROPERLY EXAMINE R2. NEXT EXAMINE MEM MGMT REGISTER KIPARI
(IF MEM MGMT IS AVAILABLE). ADD R2 AND KIPARI TOGETHER AS SHOWN BELOW.
R2=2
000 XXX XXX XXX XXX
KIPARI (772342) 0000 YYY YYY YYY
ADDRESS ZZZ ZZZ ZZZ ZZZ ZZZ

:ROUTINE TO WRITE 1'S COMPLEMENT VALUE OF ADDRESS INTO ADDRESS
:FOR EXAMPLE ROUTINE WRITES 157777 INTO ADDRESS 20000

00006 005767 176140 TST MNAVA :MEMORY MANAGEMENT AVAILABLE?
00013 001764 172342 BEQ 3S :MEMO
00020 006303 MOV #KIPARI.R3 :FIND LAST ADDRESS IF MEM MGMT USED
00027 006303 ASL R3
00034 006303 ASL R3
00041 006303 ASL R3
00048 006303 ASL R3
00055 006303 ASL R3
00062 006303 ASL R3
00069 006303 ASL R3
00076 010246 MOV R2,+(SP) :DEVELOP COMPLEMENT OF LAST ADDRESS
00083 002000 BIC #20000.(SP) :SAVE BITS IF MEMORY IS NOT A MULTIPLE OF 4K
00090 002603 ADD (SP)+,R3
00097 008174 000250 MMAB11.3*MMVEC :SET ABORT VECTOR
000A4 000002 SUB #2,R2
000A7 000250 MOV R2,R3
000B4 005103 CMP R3
000B7 000002 ADD #2,R3
000C4 001034 MOV %X5, R2
000C7 102403 BVS DONE3
000D4 017776 CMP R2, #17776
000D7 01371 BNE 1S

:SET UP TO CHECK COMPLEMENT DATA WRITTEN DOWN
MO4

MEMO: ADDRESS TESTS

1909 00272 000340
1926 00272 0010777
1939 00272 001666
1951 00272 0012345
1963 00272 001823
1974 00272 002474
1987 00272 003035

DONE3: MOV PC,R1

LOAD ADDRESS REGISTER

MATCH: MOV *MEMA

CHECK IF MM IS AVAILABLE

BEGIN 1%

MOV *MEMB *MEMC

INIT KIPARK

MOV *MEMD, *MEMVE

SET ABLT.VECTOR

1%

MOV *MEMF, *MEMG

DONEH, *MEMVE

FIRST ADDRESS

MOV R2,R0

FIRST DATA (COM OF ADDRESS)

COM RO

ADD *RO

OVE R2

ADD *R0

SUB *R1

MOV RO,(R2)+,R3

GET VALUE

CHECK

BEQ 2%

BR 2%

DONE4: NOP

;ROUTINE TO WRITE BANK INTO ALL ADDRESSES IN A 4K BANK

1909 00272 012737 000000 000004
1926 00300 010701
1939 00300 004767 004036
1951 00300 012702 002000
1963 00300 001270 002000
1974 00300 001270 002000
1987 00300 001270 002000

DONE4: MOV *MEMA, *MEMVE

SET TIME OUT TRAP VECTOR

MOV PC,R1

JSR PC:LDMMO

MOV *MEMD, *MEMVE

CLR RO

INC RO

RO WILL BE DATA WRITTEN

MOV #4096, R4

SET 4K COUNTER

2%

MOV RO,(R2)+

WRITE BANK INTO ALL ADDRESSES

DONE4: CMP (SP)+(SP)+

ADJUST STACK PTR

;CHECK THAT DATA WRITTEN ABOVE CAN BE READ

1909 00304 012737 003110 000004
1926 00300 010701
1939 00300 004767 003766
1951 00300 012702 002000
1963 00300 003354 005200
1974 00300 003354 005200
1987 00300 003354 005200

DONE4: MOV *MEMA, *MEMVE

MOV PC,R1

JSR PC:LDMMO

MOV *MEMD, *MEMVE

CLR RO

INC RO

MOV #4096, R4

2%

MOV RO,(R2)+,R3

CMP RO,R3

BEQ .44

BR 1%

DONE4: MOV *MEMA, *MEMVE

;ROUTINE TO WRITE CONSTANT DATA INTO 4K

1909 00311 010701
1926 00311 012737 007:74 000250
1939 00311 012737 007:74 000250
1951 00311 005304
1963 00311 005304
1974 00311 005304
1987 00311 005304

DONE: MOV PC,R1

MOV *MEMA, *MEMVE

;BANK STARTING WITH LAST MEMORY LOCATION

MOV PC,R1

MOV *MEMA, *MEMVE
1959 003122 162702 000302 SUB $2, R2
1960 003126 005000 CLR R0
1961 003130 005300 DEC R0
1962 003132 012704 010000 MOV #4096, R4
1963 003136 010042 02: MOV R0, -(R2)
1964 00313A 012406 BYS DONE4C
1965 003140 020227 017776 CMP R2, $17776 ; CHECK IF DONE
1966 003144 001403 BEQ DONE4C
1967 003150 005304 DEC R4
1968 003156 00137A BNE 2$
1969 003164 000765 BR 1$
1970 003168 012737 003252 000004 DONE4C: MOV #DONE4D, SPERRVEC
1971 00316C 010701 MOV PC, R1
1972 003170 004767 003652 MOV PC, LDMMO
1973 003174 012737 007142 MOV #MABT0, #MMVEC ; SET ASORT VECTOR
1974 003200 012702 020000 MOV #20000, R2
1975 003204 022704 001000 1$: CMP #4096, R4 ; CHECK IF WRITE ABOVE STARTED ON
1976 003210 001406 BEQ 2$
1977 003214 012203 MOV (R2)+, R3
1978 003218 020003 CMP R0, R3
1979 00321C 001401 BEQ .+4
1980 003220 104000 HLT
1981 003224 005204 INC R4
1982 003228 013670 BNE 1$
1983 00322C 005200 010000 INC R0
1984 003230 012704 MOV #4096, R4
1985 003234 01263C MOV (R2)+, R3
1986 003238 020003 CMP R0, R3
1987 00323C 001401 BEQ .+4
1988 003240 104000 HLT
1989 003244 005304 DEC R4
1990 003248 001372 BNE 3$
1991 003250 000766 BR 2$
1992 003254 022626 DONE4D: CMP (SP)+, (SP)+ ; BRANCH IF PROGRAM WAS NOT
1993 003258 005373 TG0 #42
1994 00325C 001406 BEQ BEGIN1 ; LOADED VIA ACTII IN GV OR AA MODES
1995 003260 013300 BEGIN1
1996 003264 025700 TG0 LOGICAL+2 ; BRANCH IF NOT IN 3V MODE
1997 003268 100003 BEGIN1
2000 003270 012737 000036 000742 MOV #6.3*ICNT ; SET ICNT TO DO 1 PASS ONLY IN GV
2001 003274 005373
WRITE 2 XOR 12 TEST PATTERN STARTING AT ADDRESS 40000

DONE2: MOV SP, DPARPA
SET INDICATOR TO WRITE 2 XOR 12 PATTERN

REPEAT TEST USING PCDFlip 3:9 PATTERN

WRITE 1 XOR 13 TEST PATTERN WRITTEN ABOVE

DONE1: MOV #SP, DPARPA
SET INDICATOR TO WRITE 1 XOR 13 PATTERN

REPEAT TEST USING PCDFlip 3:9 PATTERN

RE-STATE PROGRAM TO CHECK ADDRESSES FROM 40000-01777 USING 1 XOR 2 PATTERN.

CHECK 1 XOR 13 TEST PATTERN WRITTEN ABOVE

DONE1: MOV SP, DPARPA
SET INDICATOR TO WRITE 1 XOR 13 PATTERN

RE-STATE PROGRAM TO CHECK ADDRESSES FROM 40000-01777 USING 1 XOR 2 PATTERN.

PROGRAM IS NOW EXECUTING CODE FROM PC AS SHOWN BELOW +20000

CAUTION: DO NOT ATTEMPT TO RESTART PROGRAM AT 200

DONE1: MOV PC, R1
RESTART ADDRESS TO LOOP TEST

RE-STATE PROGRAM TO CHECK ADDRESSES FROM 40000-01777 USING 1 XOR 2 PATTERN.

RE-STATE PROGRAM TO CHECK ADDRESSES FROM 40000-01777 USING 1 XOR 2 PATTERN.
CONTAINS PATTERN

TABLE OF Routines TO WRITE SELECTED PATTERNS

WORD $1X8       : 1 XOR 9 ROUTINE
WORD $3X9       : 3 XOR 9 ROUTINE
WORD $3X9       : 3 XOR 9 ROUTINE
WORD $5X9       : 5 XOR 9 ROUTINE
USER ROUTINE    : USER ROUTINE
WORD $0D00     : ROTATING '0' ROUTINE
WORD $0D01     : ROTATING '1' ROUTINE
WORD $3X9P      : PARITY 3 XOR 9 PATTERN
WORD $ALL       : ALL EXCEPT USER

$1X8: MOV .STBANK,-(SP)       ; GET STARTING BANK
       MOV .BANKS,-(SP)      ; GET # OF 4K BANKS
       RSL (SP)             ; MULTIPLY BY 32.
       TST $PCC3FLG         ; IF PROGRAM = STOP WITH EEH.
       BNE 15                ; WORD BLOCK COUNT
       ASL (SP)             ; TO FORM 128.
       RSL (SP)             ; WORD BLOCK
       RSL (SP)             ; COUNTER
       MOV 0           ; SAVE
      _word 128, WORD BLOCK COUNT
       MOV PC, 1X9         ; GO WRITE 1 XOR 9 TEST PATTERN
       JSR .PCC1          ; CONTAINS 128, WORD BLOCK COUNT
       JSR .PCC1          ; GO WRITE 1 XOR 9 TEST PATTERN

$2X9: MOV .STBANK,-(SP)       ; GET STARTING BANK
       MOV .BANKS,-(SP)      ; GET # OF BANKS
       RSL (SP)             ; MULTIPLY BY 16.
       TST $PCC3FLG         ; IF PROGRAM = 3 STOP WITH EEH.
       BNE 15                ; WORD BLOCK COUNT
       ASL (SP)             ; TO FORM 256.
       RSL (SP)             ; 256, WORD BLOCK COUNT
       ASL (SP)             ; COUNTER
       MOV 0           ; SAVE
      _word 256, WORD BLOCK COUNT
       MOV PC, 3X9         ; GO WRITE PATTERN
       JSR .PCC1          ; CONTAINS 256, WORD BLOCK COUNT

CHECK PATTERN WRITTEN ABOVE

$3X9: MOV .STBANK,-(SP)       ; GET STARTING BANK
       MOV .BANKS,-(SP)      ; GET # OF BANKS
       RSL (SP)             ; MULTIPLY BY 16.
       TST $PCC3FLG         ; IF PROGRAM = 3 STOP WITH EEH.
       BNE 15                ; WORD BLOCK COUNT
       ASL (SP)             ; TO FORM 256.
       RSL (SP)             ; 256, WORD BLOCK COUNT
       ASL (SP)             ; COUNTER
       MOV 0           ; SAVE
      _word 256, WORD BLOCK COUNT
       MOV PC, 3X9         ; GO WRITE PATTERN
       JSR .PCC1          ; CONTAINS 256, WORD BLOCK COUNT

$1X8: MOV .STBANK,-(SP)       ; GET STARTING BANK
       MOV .BANKS,-(SP)      ; GET # OF 4K BANKS
       RSL (SP)             ; MULTIPLY BY 32.
       TST .PCC3FLG         ; IF PROGRAM = 3 STOP WITH EEH.
       BNE 15                ; WORD BLOCK COUNT
       MOV PC, 1X9         ; GO WRITE 1 XOR 9 TEST PATTERN
       JSR .PCC1          ; CONTAINS 128, WORD BLOCK COUNT

$5X9: MOV .STBANK,-(SP)       ; GET STARTING BANK
       MOV .BANKS,-(SP)      ; GET # OF 4K BANKS
       RSL (SP)             ; MULTIPLY BY 32.
       TST .PCC3FLG         ; IF PROGRAM = STOP WITH EEH.
       BNE 15                ; WORD BLOCK COUNT
       MOV PC, 5X9         ; GO WRITE PATTERN

$3X9: MOV .STBANK,-(SP)       ; GET STARTING BANK
       MOV .BANKS,-(SP)      ; GET # OF BANKS
       RSL (SP)             ; MULTIPLY BY 16.
       TST .PCC3FLG         ; IF PROGRAM = 3 STOP WITH EEH.
       BNE 15                ; WORD BLOCK COUNT
       MOV PC, 3X9         ; GO WRITE PATTERN

CHECK PATTERN WRITTEN ABOVE
; ROUTINE TO WRITE 3 & CHECK USER CONSTANT

$570: LDR $R5,=PRINT  : GO TO PRINT ROUTINE
$571: JEQ $PC,PC,RECD  : ASK FOR USER CONSTANT
$572: LDR $R3,=PC,RECD  : CONTAINS USER CONSTANT

$540: MOV $R3,$(SP)  : GET BANK COUNT
$541: MOV $R3,$(SP)  : GET 4K BANK
$542: MOV $R3,$(SP)  : PRINT 3
$543: MOV $R3,$(SP)  : SAVE

$546: MOV $R3,$(SP)  : WRITE USER CONSTANT
$547: MOV $R3,$(SP)  : GET STARTING BANK
$548: MOV $R3,$(SP)  : AND 3 OF 256 WORD BLOCKS
$549: LDR $R3,=USER  : GO TO USER CHECK ROUTINE
$54A: LDR $R3,=USER  : CHECK IF USER HAS TYPED A CHARACTER
$54B: BPL $R3,TIMES  : CLEAR FLAG

$570: LDR $R5,=PRINT  : ROTATING '0' ROUTINE
$571: JEQ $PC,105,=3  : GET # OF 256 WORD BLOCKS
$572: JEB $PC,3  : GO CHECK PATTERN

$580: MOV $R2,$(SP)  : CHECK IF RUNNING ALL PATTERNS.
$581: MOV $R2,$(SP)  : GO START BX13 PATTERN IF COND - 4

$590: MOV $R5,$(SP)  : CANNOT BC BX13 PATTERN USING
$591: BNE $PC,3  : PROGRAM = 3

$5A0: MOV $R5,$(SP)  : GET STARTING BANK
$5A1: MOV $R5,$(SP)  : AND # OF 4K BANKS
$5A2: MOV $R5,$(SP)  : MUST BE AN EVEN # OF 4K BANKS

$5B0: MOV $R5,$(SP)  : PRINT ?
$5B1: JEQ $PC,105,=1  : PRINT

$5C0: MOV $R5,$(SP)  : RESTART

$5D0: MOV $R5,$(SP)  : GC WRITE PATTERN
$5D1: MOV $R5,$(SP)  : GC CHECK PATTERN

$5E0: MOV $R5,$(SP)  : CHECK IF RUNNING ALL PATTERNS
$5E1: BNE $PC,3  : DO ROTO PATTERN IF ALL SELECTED

$5F0: MOV $R5,$(SP)  : ROUTE TO WRITE 3 & CHECK USER CONSTANT

$600: MOV $R5,$(SP)  : ASK FOR USER CONSTANT
```
005263  05564  303020
005264  00004  000010
5: MOV   RO,(R2) ; WRITE USER CONSTANT INTO ADDRESS
005265  01203  000010
5: MOV   (R2),R3 ; GET DATA WRITTEN
005266  02003  001401
5: CMP   R0,R3 ; CHECK DATA
005267  01400  014000
5: BR    6$ ; REPORT ERROR
005268  00542  00571
6: TST   -R2 ; RESTORE ADDRESS
005269  01177  017777
6: MOV   *17777,R3 ; MOVES 4K
00526a  05094  050940
6: ADD   #17777,R3 ; SET TIME OUT TRAP
00526b  00330  003300
6: MOV   #1273,#ERRVEC ; CLEAR RELOCATION SUCCESS! INDICATOR
00526c  00040  000400
6: CLR   R4 ; CHECK IF MEMORY IS AVAILABLE
00526d  01202  012020
6: TST   (R3),R2 ; RELOCATE
00526e  02039  020390
6: CMP   R2,R3 ; RELOCATION COMPLETE?
00526f  01175  011750
6: BNE   1$ ; MOV   (R5)+,R3 ; GET FROM ADDRESS
005270  01563  015630
6: MOV   (R5)+,R3 ; GET TO ADDRESS
005271  01250  012500
7: MOV   R2,R3 ; MOVES 4K
005272  01600  016000
7: ADD   #1273,R2 ; SET TIME OUT TRAP
005273  00004  000040
7: MOV   #1273,#ERRVEC ; CLEAR RELOCATION SUCCESS! INDICATOR
005274  05094  050940
7: CLR   R4 ; CHECK IF MEMORY IS AVAILABLE
005275  01224  012240
7: TST   (R3),R3 ; RELOCATE
005276  02039  020390
7: CMP   R2,R3 ; RELOCATION COMPLETE?
005277  01413  014130
7: BEQ   5$ ; BRANCH IF DONE
005278  00424  004240
7: CMP   -R0,-R2 ; CHECK THAT DATA WAS RELOCATED PROPERLY
005279  01174  011740
7: BEQ   3$ ; CHECK IF RELOCATING BACK TO 000000
00527a  05702  057020
7: TST   R3 ; ERROR! CANNOT RELOCATE PROGRAM CODE TO UPPER MEMORY BANK PROPERLY
00527b  01403  014030
7: BEQ   3$ ; ERROR! CANNOT RELOCATE CODE BACK TO 000000 PROPERLY
00527c  00004  000040
8: HLT   2$ ; CONTINUE RELOCATING AT YOUR PERIL
00527d  00076  000760
8: HLT   3$ ; TOP OF MEMORY.
00527e  00000  000000
8: HALT  3$ ; TOP OF MEMORY.
00527f  00777  007770
8: BR    3$ ; TOP OF MEMORY.
005280  00004  000040
9: CMP   (SP)+,(SP)+ ; RESTORE STACK PTR
005281  00104  001040
9: CMP   R4 ; TOP OF MEMORY.
005282  00024  000240
9: NOP   5$ ; TOP OF MEMORY.
005283  00375  003750
9: MOV   #RELOC Factor,R2 ; GET ADDRESS OF RELOCATION FACTOR
005284  00302  003020
9: ADD   (R5),R2 ; ADD FACTOR
005285  00125  001250
9: MOV   (R5)+,R2 ; RELOCATED RELOC FACTOR NOW CONTAINS RELOCATION FACTOR
005286  00205  002050
9: RTS   5$ ; RETURN, R4=-1 IF NO RELOCATION
```

```
006263  01270  000000  000004
006264  02266  022660
006265  00104  001040
5: MOV   #20000,RO ; SET UP TO SCAN FOR TOP OF MEMORY
006266  00036  000360
5: MOV   #ERRVEC+2,#ERRVEC ; INCREMENT SCAN ADDRESS
006267  01270  012700
5: ADD   #20000,RO ; SET TIME OUT INDICATOR
006268  00026  000260
5: SEC   (RO) ; CHECK FOR EXISTANT MEMORY
006269  01274  012740
5: BCC   1$ ; "C" WILL BE CLEAR IF MEMORY EXISTS
00626a  10373  103730
5: MOV   #ERRTRP,#ERRVEC
```

```
: ROUTINE TO RELOCATE PROGRAM CODE FROM ORIGINAL POSITION (0-4K) TO TOP OF MEMORY.
```

```
: ROUTINE TO RELOCATE PROGRAM CODE FROM ORIGINAL POSITION (0-4K) TO TOP OF MEMORY.
```
SBDTL 3 X0R 9 ROUTINES

:ROUTINE TO WRITE XOR 9 WORST CASE NOISE TEST PATTERN

:CALL: MOV BANK 8,-15P: PUSH STARTING BANK 8 IN STACK

: MOV BLKCON,-15P: PUSH 256, WORD BLOCK INKIN+15, STACK

:JSR PC,-3X9: CALL ROUTINE

: 3X9: MOV 4,-SP,R2: GET STARTING BANK 8

:JSR PC,STAT52

: MOV RC, R3

:COM R3

:TST PARPAT

:BRANCH IF PARITY MEMORY PATTERN IS

: NOT TO BE WRITTEN

: 3: MOV #401,R0

: WRITE PARITY 3X9 PATTERN

: EACH LOOP WRITES 256, XOR 9
CALL: MOV BANK#, -(SP) ; PUSH STARTING BANK # INTO STACK
        MOV BLKCNTR, -(SP) ; AND 256, WORD BLOCK COUNT
        JSR PC, .X3X9 ; CALL ROUTINE

.X3X9: NOP PC, CKSWR ; GO CHECK SWITCH REGISTER

; CHECK WORST CASE PATTERN
3%: MOV 2(SP), R4 ; GET 256, BLOCK WORD COUNT
    MOV 4(SP), R2 ; GET FIRST BANK #
    JSR P1, STMM2 ; SET UP MEM MGMT
    JSR R5 ; SET CHECK WORD
    TST ICOUNT ; IF ICOUNT IS NEG AM CHECKING COMPLEMENTED PATTERN
    BPL ICOUNT ; IF ICOUNT IS NEG AM CHECKING COMPLEMENTED PATTERN
    CMP R0 ; SO COMPLEMENT CHECK WORD
    MOV #64, R5 ; SET 256, WORD COUNTER

3%: TST PARPAT ; BRANCH IF PARITY MEM R1 PATTERN IS
    BEQ 30% ; BRANCH IF PARITY MEM R1 PATTERN IS

30%: MOV (R2), R3 ; SET TEST DATA
<table>
<thead>
<tr>
<th>Line</th>
<th>Address</th>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0100</td>
<td>320000</td>
<td>MOV R3,R0</td>
<td>SET R3 to R0</td>
</tr>
<tr>
<td>0102</td>
<td>080000</td>
<td>MOV R3,R0</td>
<td>SET R3 to R0</td>
</tr>
<tr>
<td>0104</td>
<td>180000</td>
<td>MOV R3,R0</td>
<td>SET R3 to R0</td>
</tr>
<tr>
<td>0106</td>
<td>280000</td>
<td>MOV R3,R0</td>
<td>SET R3 to R0</td>
</tr>
<tr>
<td>0108</td>
<td>380000</td>
<td>MOV R3,R0</td>
<td>SET R3 to R0</td>
</tr>
<tr>
<td>010A</td>
<td>480000</td>
<td>MOV R3,R0</td>
<td>SET R3 to R0</td>
</tr>
<tr>
<td>0110</td>
<td>580000</td>
<td>MOV R3,R0</td>
<td>SET R3 to R0</td>
</tr>
<tr>
<td>0112</td>
<td>680000</td>
<td>MOV R3,R0</td>
<td>SET R3 to R0</td>
</tr>
<tr>
<td>0114</td>
<td>780000</td>
<td>MOV R3,R0</td>
<td>SET R3 to R0</td>
</tr>
<tr>
<td>0116</td>
<td>880000</td>
<td>MOV R3,R0</td>
<td>SET R3 to R0</td>
</tr>
<tr>
<td>0118</td>
<td>980000</td>
<td>MOV R3,R0</td>
<td>SET R3 to R0</td>
</tr>
<tr>
<td>011A</td>
<td>A80000</td>
<td>MOV R3,R0</td>
<td>SET R3 to R0</td>
</tr>
<tr>
<td>011C</td>
<td>B80000</td>
<td>MOV R3,R0</td>
<td>SET R3 to R0</td>
</tr>
<tr>
<td>011E</td>
<td>C80000</td>
<td>MOV R3,R0</td>
<td>SET R3 to R0</td>
</tr>
<tr>
<td>0120</td>
<td>D80000</td>
<td>MOV R3,R0</td>
<td>SET R3 to R0</td>
</tr>
<tr>
<td>0122</td>
<td>E80000</td>
<td>MOV R3,R0</td>
<td>SET R3 to R0</td>
</tr>
<tr>
<td>0124</td>
<td>F80000</td>
<td>MOV R3,R0</td>
<td>SET R3 to R0</td>
</tr>
</tbody>
</table>

The above code snippet appears to be a computer program written in assembly language. It seems to involve operations such as moving data between registers and performing conditional branches and jumps based on the content of the registers.
GET BLOCK

COUNT

PUSH FAKE STATUS ON - E.S.

COMPLEMENT CHECK WORD COUNT

COMPLEMENT WORD COUNT

PUSH WORD COUNT

COMPLEMENT BLOCK COUNT

COMPLEMENT CHECK WORD COUNT

PUSH WORD COUNT

COMPLEMENT BLOCK COUNT

SET BLOCK ON TEST

BRANCH IF LOOP ON TEST

STOP
; ROUTINE TO WRITE 8 XOR 13 WORST CASE NOISE TEST PATTERN
SBTL 0 XOR 13 ROUTINES
; CALL: MOV BANK *, -(SP)
;       MOV $40Banks,- (SP)
;       JSR PC, $B13

010160 0160D8 0060D6 0060D5 00C1B6

010160 0056D6 0056D6 0056D6 0056D6

010160 0056D6 0056D6 0056D6 0056D6

010160 0056D6 0056D6 0056D6 0056D6

010160 0056D6 0056D6 0056D6 0056D6

010160 0056D6 0056D6 0056D6 0056D6

010160 0056D6 0056D6 0056D6 0056D6

010160 0056D6 0056D6 0056D6 0056D6

; ROUTINE TO CHECK 8 XOR 13 WORST CASE NOISE TEST PATTERN
; CALL: MOV BANK *, -(SP) ; PUSH FIRST BANK * ON THE STACK
;       MOV $40Banks,- (SP) ; PUSH 8 OF 4K BANKS TO CHECK ON THE STACK
;       JSR PC, $B13 ; CALL ROUTINE
MOV (R2)+, R3 ; GET TEST DATA
CMP R0, R3 ; COMPARE WITH CHECK WORD
BEQ $.10 ; PUSH FAKE STATUS ON THE STACK
CLR -(SP) ; ERROR! MEM DATA (R3) NOT = TEST DATA
JSR PC, ERROR ; (R0), ADDRESS=(R2)-2

MOV (R2)+, R3 ; GET TEST DATA
CMP R0, R3 ; COMPARE WITH CHECK WORD
BEQ $.10 ; PUSH FAKE STATUS ON THE STACK
CLR -(SP) ; ERROR! MEM DATA (R3) NOT = TEST DATA
JSR PC, ERROR ; (R0), ADDRESS=(R2)-2

MOV (R2)+, R3 ; GET TEST DATA
CMP R0, R3 ; COMPARE WITH CHECK WORD
BEQ $.10 ; PUSH FAKE STATUS ON THE STACK
CLR -(SP) ; ERROR! MEM DATA (R3) NOT = TEST DATA
JSR PC, ERROR ; (R0), ADDRESS=(R2)-2

MOV (R2)+, R3 ; GET TEST DATA
CMP R0, R3 ; COMPARE WITH CHECK WORD
BEQ $.10 ; PUSH FAKE STATUS ON THE STACK
CLR -(SP) ; ERROR! MEM DATA (R3) NOT = TEST DATA
JSR PC, ERROR ; (R0), ADDRESS=(R2)-2

DEC R5 ; DECREMENT WORD COUNT
BNE R3 ; DECREMENT BLOCK COUNT

BIT #40000, 3#SU ; CHECK LOOP SWITCH
BNE 1$ ; LOOP CHECKING THIS PATTERN
HSL ICOUNT ; SHIFT PATTERN INDICATOR
BNE 1$ ;
MOV (SP)+, (SP) ; ADJUST STACK
MOV (SP)+, (SP)
RTS ; RETURN TO CALLER

SBTTL GETTTY INPUT ROUTINE
; ROUTINE TO GET ASCII INPUT FROM TTY AND CONVERT TO OCTAL
; ROUTINE LEAVES THE FIRST 16 BITS IN ADDRESS FOLLOWING THE CALL
; AND THE LAST 2 BITS IN 1617 BELOW.
; CAL: JSR PC, RECO
; ECOD: JSR PC, $SAVR ; GO SAVE REGISTERS ON THE STACK

JSR PC, INUM
MOV R2, TEMP3
MOV R3, TEMP3
JSR PC, $RESTR
MOV (SP), TEMP1
MOV TEMP3, TEMP1
MOV TEMP2, 1617
:ROUTINE TO CHECK THE SWITCH REGISTER
:CHECK SWITCH 9 :IF SET, LOAD ERROR COUNT INTO THE DISPLAY REGISTER:
:IF NOT SET, LOAD PASS COUNT INTO THE DISPLAY REGISTER
:9KSWR: BIC #17777, LDDISP ;SAVE RELOCATION BITS
:BIT #BITB,3,#SWR ;CHECK SWITCH B
:BEQ 10$ ;BRANCH IF SET
:JSR PCREL24K ;GO RELOCATE PROGRAM BACK TO #44K END
:10$: BIT #BIT9,3,#SWR ;SWITCH 9 SET?
:BEQ 1$ ;BYE
:1$: BIS ERCTN, LDDISP ;LOAD ERROR COUNT
:BR 2$ ;BYE
:2$: BIS ICNT, LDDISP ;LOAD PASS COUNT
:MOV LDDISP, #DISPLAY ;LOAD THE DISPLAY REGISTER
:MOV #0017, ICONT ;LOAD ITERATION COUNT WORD
:BIT #4000, #SWR ;CHECK SW11
:BEQ 6$ ;BYE
:6$: CLRIB ICONT ;ICOUNT =040000 IF SW11 =1
:RTS PC

:MESSAGES
:ANESLR: .ASCIZ "15:12" 'TO RESTORE LOADERS START AT 162'15:12'

:PARITY: .ASCIZ "15:12" 'ENABLE PARITY? 1=O=YES:NO'

:STBANK: .ASCIZ "15:12" 'STARTING BANK #9' '

:STBANKS: .ASCIZ "15:12" 'OF #4K BANKS TO TEST(S)'

:PAT: .ASCIZ "15:12" 'PATTERN #'

:GUEST: .ASCIZ "15:12" 'GUEST'

:CON: .ASCIZ "15:12" 'TYPE CONSTANT'

:PRG3M: .ASCIZ "15:12" 'INPUT # OF 256 WORD BLOCKS TO TEST INSTEAD OF'
PROGRAM DESCRIPTION

This is a pseudo-modified version of the Branch Gobble MOS Memory Exerciser. Pseudo-modified because the original code, taken from the DZQA-A instruction exerciser which is Branch Gobble is included. Here in its original form. But memory management capabilities have been added to give it operating abilities in a 0-128K memory environment.

OPERATING PROCEDURE

When loaded, this program's starting address is XXXX.

When running the following steps are taken:

1. A program ID is typed on the TTY:
   - Branch Gobble MOS TEST

2. The program determines if there is MOS parity. If yes it is enabled.
AND THE USER IS TOLD:

"THE USER IS ASKED:"

"USE MEMORY MANAGEMENT".

"Y / N"

"IF THE USER TYPES Y, THE MEMORY MANAGEMENT"

"WILL BE USED TO RUN THE TEST. IF HE TYPES"

"N THEN MEMORY MANAGEMENT WILL NOT BE USED"

"IN EITHER CASE THE ACTUAL TEST WILL BE"

"PERFORMED BY THE UNMODIFIED ORIGINAL VERSION"

"OF BRANCH Gobble."

"NOTE THAT WHEN THE TEST OF MEMORY"

"LOCATED IN UNIBUS ADDRESSES HIGHER THAN"

"17000H IS DESIRED, THE ENABLING OF MEMORY"

"MANAGEMENT IN THIS PROGRAM IS MANDATORY."

"NOTE ALSO THAT WHEN THE TEST IS TO BE"

"IN THE 0 TO 17000H RANGE OF UNIBUS ADDRESSES"

"IT IS RECOMMENDED THAT MEMORY MANAGEMENT"

"BE DISABLED."

"THE PROGRAM WILL THEN ASK THE USER"

"FOR THE HIGH ADDRESS LIMIT FOR THE "TEST":"

"WHAT IS THE HIGH LIMIT?

"AND THEN FOR THE LOW LIMIT:

"WHAT IS THE LOW LIMIT?

"BOTH OF THE ADDRESSES SHOULD BE SPECIFIED IN"

"FORM XXXXXX, THAT IS THE"

"ADDRESSES MUST BE THE BEGINNING OF 16-BIT"

"BLOCKS OF MEMORY. IF THEY ARE NOT THEN THEY"

"WILL BE TRUNCATED."

"THE ADDRESSES WILL BE INTERPRETED AS FULL 19-BIT"

"UNIBUS ADDRESSES."

"VALID ADDRESSES MEET THE FOLLOWING CONDITIONS:

"1. THE SPECIFIED SPAN OF THE "TEST"

"SHOULD NOT ENCOMPASS THE ACTUAL"

"MEMORY LOCATIONS OCCUPIED BY THIS"

"PROGRAM. THIS PROGRAM IS RELOCATABLE"

"AT LOADING TIME, SO THAT NO LIMITATIONS"

"ARE THUS IMPOSED."

"2. THE HIGH LIMIT SHOULD BE GREATER"

"THAN THE LOW LIMIT."

"3. THE NUMBER MUST BE SPECIFIED BY"

"32 DIGITS."

"4. IF THE USER HAS DISABLED MEMORY"

"MANAGEMENT THE ADDRESSES SHOULDN'T"

"BE IN LOW MEMORY, I.E. LESS THAN 17000H."

"IF ANY OF THESE CONDITIONS IS NOT MET"

"THE USER WILL BE ASKED TO INPUT ANOTHER"

"ADDRESS."

"NOTE THAT IF ANY OF THE ADDRESSES IN THE TEST"

"SPAN IS A NON-EXISTANT MEMORY LOCATION"

"A TIME OUT ERROR WILL OCCUR FROM WHICH"

"NO RECOVERY CAN BE MADE EXCEPT "O RES AR""

"AT THE STARTING ADDRESS."

"WHEN THE ABOVE INFORMATION HAS BEEN"

"SUCCESSFULLY GATHERED FROM THE USER, THE
TEST WILL BEGIN. IF BRANCH JUMPS ENCOUNTERS A MEMORY FAULT DURING THE TEST, THE PROCESSOR WILL BE HALTED AT THE "LOCATION" OF THE MEMORY FAULT.

5. IF THE TEST IS COMPLETE WITHOUT AN ENCOUNTER WITH A MEMORY FAULT ANY WHERE IN THE TESTED SPAN, THEN THE TEST WILL BE REPEATED BY RETURNING TO STEP 4.

B. IF THE USER WISHES TO STOP THE TEST IN PROGRESS, 1. SET ANOTHER WITH A DIFFERENT RANGE THEN HE SHOULD EXIT THE CALL AND START THE TEST AGAIN IN A SEC.

TOP:

CONSTANTS:

A8R=0774D6

THIS ROUTINE TAKES CARE OF THE IDENTIFICATION, TASK THE USER IF MEMORY MANAGEMENT SHOULD BE ENABLED, AND IF NOT DOES THE SET UP FOR THE ACTUAL TEST. IF MEMORY MANAGEMENT IS DESIRED THE ROUTINE YMNBGO IS GIVEN CONTROL.

012436 000000

LODFGLG: .WORD 0

BRANCH:

CLR RELFL ; INITIALIZE THE RELLOCATION INDICATOR.

MNAV ; INITIALIZE THE MEMORY MANAGEMENT FLAG.

JSR RS$PRINT ; GO TO PRINT ROUTINE

JSR PC$SR

JSR PC$OR

JSR PC$PO

JSR RS$PRINT ; GO TO PRINT ROUTINE

JSR CMP ; MEMORY MANAGEMENT SHOULD BE ENABLED.

JSR CMP ; WAIT FOR A CHAR.

JSR CMP ; GET THE CHARACTER.

JSR CMP ; ECHO THE CHARACTER.

JSR CMP ; RESET THE STACK.

CMP ; IF IT IS Y, THEN 33

BEQ YMNBGO ; TO YMNBGO TO ENABLE MEMORY MANAGEMENT.

CMP ; IF IT WAS NEITHER

MOV ;

3S$
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0110</td>
<td>RETURN</td>
</tr>
<tr>
<td>0111</td>
<td>SET</td>
</tr>
<tr>
<td>1001</td>
<td>CLEAR</td>
</tr>
<tr>
<td>0001</td>
<td>HALT</td>
</tr>
<tr>
<td>0000</td>
<td>CLEAR</td>
</tr>
<tr>
<td>0010</td>
<td>RETURN</td>
</tr>
<tr>
<td>1010</td>
<td>CALL</td>
</tr>
<tr>
<td>1100</td>
<td>BRANCH</td>
</tr>
<tr>
<td>0101</td>
<td>NEGATE</td>
</tr>
<tr>
<td>0110</td>
<td>RETURN</td>
</tr>
<tr>
<td>0111</td>
<td>SET</td>
</tr>
<tr>
<td>1001</td>
<td>CLEAR</td>
</tr>
<tr>
<td>0001</td>
<td>HALT</td>
</tr>
<tr>
<td>0000</td>
<td>CLEAR</td>
</tr>
<tr>
<td>0010</td>
<td>RETURN</td>
</tr>
<tr>
<td>1010</td>
<td>CALL</td>
</tr>
<tr>
<td>1100</td>
<td>BRANCH</td>
</tr>
<tr>
<td>0101</td>
<td>NEGATE</td>
</tr>
</tbody>
</table>

**Note:**
- The routine is called to shift RC to the left six bits.
- The setup routine initializes the registers used in the test pattern generator.
THE LIMITS ARE THEN GOTTEN AND CHECKED FOR VALIDITY.
THE TRAP INTERRUPT VECTORS ARE SET AND (MM831)
IS CALLED. MMGB1 IS A ROUTINE THAT SETS THOSE
MEMORY MANAGEMENT REGISTERS WHICH NEED TO
BE CHANGED DYNAMICALLY DURING THE TEST.

JSR PC.LIMITS

JSR PC.LOSEG4

MOV @-SP	; SET UP THE STACK

MOV 0

MOV 0

SET UP THE TRAP INTERRUPT LINES

WHILE A TEST IS IN PROGRESS, WORKING UPWARDS FROM THE LOWEST
MEMORY MANAGEMENT IR SET TO ENABLE BRGB TO WORK THROUGH AS
MUCH OF THE TEST SPAN AS POSSIBLE IN A SINGLE MANAGEMENT SET UP
BEFORE HAVING TO RESET THE MANAGEMENT REGISTERS.
BRGB ALWAYS IS IN LOW VIRTUAL MEMORY AND UPPER VIRTUAL
ADDRESSES ARE ALWAYS MAPPED INTO UNIBUS DEVICE ADDRESSES.

JSR PC.LIMITS

JSR PC.LOSEG4

MOV 0

SET UP THE STACK

MOV 0

SET UP THE TRAP INTERRUPT LINES

WHICH WILL BE USED TO RETURN FROM THE TESTING ROUTINE.

MMGB1 IS USED TO DYNAMICALLY ALLOCATE MEMORY UNDER MEMORY MANAGEMENT
WHILE A TEST IS IN PROGRESS, WORKING UPWARDS FROM THE LOWEST
MEMORY MANAGEMENT IR SET TO ENABLE BRGB TO WORK THROUGH AS
MUCH OF THE TEST SPAN AS POSSIBLE IN A SINGLE MANAGEMENT SET UP
BEFORE HAVING TO RESET THE MANAGEMENT REGISTERS.
BRGB ALWAYS IS IN LOW VIRTUAL MEMORY AND UPPER VIRTUAL
ADDRESSES ARE ALWAYS MAPPED INTO UNIBUS DEVICE ADDRESSES.
GO7

TEST COMM-3 XORiN MEMORY EXERCISES  MAK 31 27 780 10-SEP-73 11:59 PAGE 66

132156 012767 023000 023272
132160 010530 002000 002272 13: MOV R5, R0(1)
132164 006200 000000 000272 25: ADD #200, R5
132168 013697 001A00 001B66 25$: CMP HIIM, R5
132172 014091 000066 000066 25$: BCS 35
132176 009201 000000 000000 25$: INC R1
132180 002770 000000 000000 3$: BLT 13
132184 012567 002244 000000 3$: MOV R5, R1
132188 015705 000000 002230 3$: SUB #8, R5
132192 015767 000000 002230 3$: BR 48
132196 004303 000000 000230 3$: MOV HIIM, HI
1321A0 015767 000538 002222 4$: MOV LOIM, HI
1321A4 015700 002216 000000 4$: MOV R5, R0(1)
1321A8 004767 177044 000000 4$: JSR P, #0(RC)
1321AC 005067 000000 000000 4$: ADD $DC00, R0(1)
1321B0 005067 000000 000000 4$: MOV R5, R0(1)
1321B4 015667 000537 177542 4$: INC #SRC
1321B8 00C000 000000 000000 4$: JMP R1
1321BC 027F05 000000 000000 000000 5$: JMP #0
1321C0 002300 002290 177550 5$: JMP #0
1321C4 002300 002290 177550 5$: JMP #0
1321C8 002300 002290 177550 5$: JMP #0
1321CC 002300 002290 177550 5$: JMP #0
1321D0 002300 002290 177550 5$: JMP #0
1321D4 002300 002290 177550 5$: JMP #0
1321D8 002300 002290 177550 5$: JMP #0
1321DC 002300 002290 177550 5$: JMP #0
1321E0 002300 002290 177550 5$: JMP #0
1321E4 002300 002290 177550 5$: JMP #0
1321E8 002300 002290 177550 5$: JMP #0
1321EC 002300 002290 177550 5$: JMP #0
1321F0 002300 002290 177550 5$: JMP #0
1321F4 002300 002290 177550 5$: JMP #0
1321F8 002300 002290 177550 5$: JMP #0
1321FC 002300 002290 177550 5$: JMP #0
132200 002300 002290 177550 5$: JMP #0
132204 002300 002290 177550 5$: JMP #0
132208 002300 002290 177550 5$: JMP #0
13220C 002300 002290 177550 5$: JMP #0
132210 002300 002290 177550 5$: JMP #0

RESET THE RIPAR.

REACHED HIIM.

YES, GOTO 35.

NO, INCREMENT R1, AND SEE IF ALL THE RIPAR'S HAVE BEEN SET.

ALL THE RIPAR'S HAVE NOT BEEN SET:

LOOP TO GET THE NEXT ONE.

DO THIS IF ALL THE TEST SPAN HAS NOT BEEN ALLOCATED TO SOME APP.

IS THE KERNAL INSTRUCTION SPACE.

IS THE "ES" SCAN.

IS THE "ES" SCAN.

RETURN TO BADOBJ 73.

PERFORM THE TEST IN THE SCAN INDICATED.

BY THE RESULT OF THE

ABOVE.

RESTART.

LOSEG1 IS CALLED TO DECIDE WHETHER OR NOT THE LIMITS ARE SUCH THAT THE FIRST 4K OF MEMORY WILL HAVE TO BE CHECKED IN THE TEST. THAT IS, DO THE LIMITS INCLUDE ADDRESSES WHICH LIE IN THE FIRST 4K BLOCK OF MEMORY? REQUESTING THAT THE CONTENTS OF THIS FIRST BLOCK OF MEMORY BE MOVED INTO THE SECOND 4K BLOCK SO THAT THE TEST CAN BE RUN THROUGH THE FIRST 4K BLOCK IF RELOCATION IS NECESSARY.

LOSEG1: CLR LOSFL, #200.

LOSEG1: SLR LOSAV, #200.

LOSEG1: BHS 2%.

LOSEG1: BHS 2%.

LOSEG1: DEC LOSFL.

LOSEG1: CMP #200, HIIM.

LOSEG1: BR 1%.

LOSEG1: JMP #0, LOIM.

LOSEG2 IS CALLED TO DO THE ACTUAL RELOCATION OF THE FIRST 4K MEMORY.
BANK INTO THE SECOND 4K MEMORY BANK, THE RUN THE TEST.
THROUGH THE DESIGNATED PARTS OF THE FIRST BANK ARE RELOCATED.
THE CONTENTS OF THE FIRST BANK BY MOVING THE SECOND BANK.
CONTENTS BACK INTO THE FIRST BANK.

LOSEG8: TST  LOSFL : SEE IF RELOCATION IS NECESSARY.
       BMI 1$ : IF NOT PERFORM.
       RTS P.
       1$: MOV  #EP,SAVF.
       TST #120 : SEE IF THE PROGRAM IS ALREADY RELOCATED.
       BMI 1$.

WORD 0.
WORD 20000.
MOV #1,2128.
MOV #20000,R1.
ADD R1,SP.
ADD R1,PC.
ADD R1,EP.
ADD R1,SAVF.
MOV #1,RELFL.
MOV LOSAV,RO : ESTABLISH VALID LIMITS FOR THE TEST.

JMP PC,ROT : THROUGH THE FIRST 4K MEMORY BANK.

ADD PC,R1.
ADC #6,R1.
MOV R1,#34.
CLR #34.
RTI.

MOV #LOSEG3,R1.
ADD PC,R1.
ADC #6,R1.
MOV R1,#34.
CLR #34.
RTI.

MOVE #LOSEG3...R1.

LOSEG3 RELOCATES BACK INTO THE FIRST 4K MEMORY BANK.
LOSEG8: MOV  SAVPC - SP.
       TST  PERRFL .
       BMI 1$.
       JSR  RS,RELOC.
       WORD 20000.
       WORD 0.
       CLR #120.
       MOV #20000,R1.
       SUB R1,SP.
LIMITS IS CALLED TO ASK THE SER FOR BOTH THE HIGH AND LOW LIMIT BUS ACCESS LIMITS. FOR THE IMPENDING TEST, THE TWO LIMITS ARE LEFT IN BLOCK NUMBER FORM AT LOCATIONS HLIM AND LOLIM (THEY ARE ALSO PUT IN LOSAV AND HISAV FOR LATER USE BY THE ROUTINE GONE). A VALIDITY CHECK IS MADE TO MAKE SURE THE INDICATED SPAN IS A VALID TEST SPAN.

LIMITS:

13646 JSR R5.SPRINT : GO TO PRINT ROUTINE
004567 165106
13652 WORD HILIM
13658 JSR PC.INUM : ASSEMBLE THIS NUMBER.
004767 000110
13664 JSR PC.THAR
13670 MOV R3.HLIM
13676 JSR R5.SPRINT
13682 004567 165060
13688 JSR PC.INUM
13694 JSR PC.THAR
13696 MOV R3.LOLIM
13698 CMP HLIM,LOLIM : IF LOLIM IS GREATER THAN HLIM THEN GSTC LIMERR, ERROR :
13700 BLE LIMERR : STORE THE LIMITS IN SAVE REGISTERS.
13706 RTS PC : RETURN
13712 MOV HLIM,HILIM
13716 JSR R5.SPRINT
13722 004767 000032
13728 00032
13734 CMP LOLIM,LOLIM
13740 BLE LIMERR
13746 RTS PC
13752 MOV LOLIM,LOSAV
13758 JSR R5.SPRINT
13764 004567 165004
13766 JSR ERANESS : WRITE AN ERROR MESSAGE
13772 BR LIMIN : AND TRY AGAIN.
13778 00033
13784 KILIM : WORD 0
13790 LOM : WORD 0
13796 : THESE ARE INTERMEDIATE STORAGE REGISTERS:
13802 LOSAV : WORD 0
13808 HISAV : WORD 0
THIS ROUTINE IS CALLED TO ASSEMBLE AN 18-BIT NUMBER FROM THE TTY AND TRUNCATE IT DOWN TO 12-BITS.
A CALL IS MADE THUS:

JMP PC, INUM

RES: WORD 0
THE NUMBER IS ASSEMBLED AND THE RESULTING 12-BIT TRUNCATED NUMBER IS LEFT IN RES. WHEN AND
RTS RETURN IS MADE.
NOTE THAT THE NUMBER SHOULD BE SPECIFIED
IN OCTAL DIGITS. AN CHARACTERS WHICH DO NOT
MEET THIS SPECIFICATION IN THE INPUT STRING
WILL CAUSE AN ERROR WHICH WILL BE SIGNALIZED BY
A ? ON THE TTY FOLLOWED BY A RETRY.

INUM: CLR (SP); PUT A ZERO MARKER ON
CLR R2; SET UP THE TEMPORARY
CLR R3; STORAGE AND COUNTER
CLR R5; REGISTERS.
MOV $7, R5
TST @TKB
MOV @T, @SP
JSR PC, TYPIT
TST (SP)+
BPL 1$;
MOV @TKB, (SP)
BIC 177600, (SP);
CMP 177600, (SP);
BIT 2$;
BNE 1$;
TST (SP)+;
TST (SP); FIRST SEE IF THERE
IS A PREVIOUS
CHARACTER ON THE STACK;
IF THERE WAS NO PREVIOUS
CHARACTER TAKE NO
ACTION AND GO BACK
TO WAIT AT 1$ FOR
THE NEXT CHARACTER
IF THERE WAS
A PREVIOUS CHARACTER
PRINT A SLASH

1$: MOV @T, @SP
JSR PC, TYPIT

2$: JSR PC, TYPIT

BPL 1$: IF THE LAST INPUT

CHARACTER WAS NOT

RUBOUT ECHO IT

IS IT CR

THEN BRANCH TO 1$ FOR

THE NEXT CHARACTER.

YES, PRINT A LF

3$: CMP @15, @SP
BNE 1$;
MOV @12, @SP
JSR PC, TYPIT
TST (SP)+;
IST (SP)+;
BEQ 4$;
START TO ASSEMBLE
THE NUMBER. IF THE
STACK IS AT THE
ZERO MARKER WE ARE
DONE, SO GOTO 4%
CHECK TO SEE IF
THE CHARACTER IS A
VALID OCTAL DIGIT.
IF NOT GOTO INERR.
ARE THERE TO MANY DIGIT?
MORE THAN 6
ROTATE THE DOUBLE
LENGTH "WORD" MADE JF
OF THE DIGITS PROCESSED
YET FAR
MAKE THIS NEW DIGIT
PART OF THE NUMBER.
LOOP TO GET THE REST OF
THE NUMBER.
MAKE SURE THE NUMBER
HAS BEEN RIGHT JUSTIFIED
PROPERLY.
AND RETURN CONTROL.

ERROR HANDLER FOR THE INUM ROUTINE.
RETURNS TO THE CALLING ROUTINE, LIMITS TO ASK
FOR THE PARAMETER AGAIN.

INERR:

IS: TST (SP)+
CLEAN JUNK OFF THE
STACK

ONER: JMP INUM
THE ROUTINE LIMITS
ROUTINE WHICH ROTATES
FOR RE
FOR R3
BIT TO THE RIGHT
TREATING THEM AS A 32-BIT
OPERAND.

THRA:

SJ: TSTB @TPS
TYPIT TAKES THE
WORD 2 BYTES UP IN
MOV B (SP),@TPB
THE STACK AND "PRINTS"
THE REAL REGISTERS ARE LOCATED USING A TIME OUT PLAN. WHERE THE POSSIBILITIES ARE REFERENCED AND IF A TIME OUT OCCURS THE, ARE "NONE." OR IF NO TIME OUT OCCURS THEN THE REGISTER IS REAL." WHEN A REAL REGISTER IS FOUND IT IS WRITTEN INTO AND READ TO DETERMINE WHAT KIND OF PARITY REGISTER IT IS, CORE OR MOS. IF IT IS MOS THEN THE REGISTER IS SET TO ENABLE PARITY.

MP1: TST (RO) IF THIS INSTRUCTION TIMES OUT THEN.

MP2: CMP (SP)+(SP)+ IF A TIME OCCURS COME HERE OR IF THE REGISTER WAS NOT MOS PARITY.

MP3: MOV #1(RO) ; OUTPUT AN ENTRY INTO THAT TABLE.

MP4: BR MP25 ; BRANCH IF NOT DONE.

MP5: ADD #2, RO IF ANY MOS PARITY REGISTERS HAVE BEEN FOUND.

MP6: BMI MP3 IF NOT DONT OUT THE NO PARITY MESSAGE.


MPFL: MOV #1(RO); GO TO PRINT ROUTINE.

MPFL: MOV RD.-(SP); GO TO PRINT ROUTINE.

MPFL: MOV #1(RO); GO TO PRINT ROUTINE.
; TELL THE USER A PARITY ERROR OCCURRED.
JSR R5,$PRINT; GO TO PRINT ROUTINE
; PERMIS ; TELL THE USER WHAT INSTRUCT CAUSE THE ERRCR.
JSR PC,02A
JSR R5,$PRINT; GO TO PRINT ROUTINE

MOV *PARR3,-,RC; SET THE TIME AND TRAP VECTORS SO
ADD PC,RO; THAT THE SCAN CAN BE MADE
ADD $5,R0; 
MOV RO,*PARR3-1,000100; 
CLR *PARR3-1,000000; 
MOV *PARR3-1,RO; 
ADD PC,RO; 
ADD $5,R0; 
MOV RO,*PARR3-2,000000; 
CLR *PARR3-2,000000; 
MOV *PARR3-2,RO; 
CLR R3

; PERFORM THE SCAN, EITHER THIS INSTRUCTION.
I$: MOV .R31+,R1; WILL TIME OUT OR THE PARITY ERROR WILL
BR 1$; ONCE AGAIN OCCUR SO THAT CONTROL WILL
; BE TRANSFRESHED OUT OF THIS LOOP ONE WAY
; OR ANOTHER.

; IF THE PARITY ERROR IS AGAIN ENCOUNTERED THEN THIS PROGRAM WILL
; RECEIVE CONTROL THROUGH A TRAP THROUGH VECTOR 114. HERE THE USER
; IS GIVEN THE ADDRESS OF THE LOCATION WHICH CAUSED THE PARITY ERROR
; DURING THE SCAN. THE USER IS ALSO TOLD WHETHER OR NOT MEMORY MANAGEMENT
; IS ENABLED OR NOT. IF IT IS ENABLED THEN THE PAR INVOLVED WITH
; THE ADDRESS RELOCATION IS ALSO GIVEN TO THE USER.
PARRA: ADD $2,R3; R3 CONTAINS THE ADDRESS PLUS 2
MOV R3,(SP); OF THE LOCATION CAUSING THE ERRCR.
ADD $2,R3
MOV R3,SPER
JSR PC,02A; GIVE THE USER THIS ADDRESS
JSR R5,$PRINT; GO TO PRINT ROUTINE
JSR PC,02A; GIVE THE USER THIS ADDRESS AND
JSR R5,$PRINT; GO TO PRINT ROUTINE
JSR PO,02A; GIVE HIM THE 3RD DATA.

; SEE IF MEMORY MANAGEMENT WAS USED
BIT #SR0,#1; DURING THE SCAN THROUGH MEMORY.
BEQ 2$; IF IT WAS THEN ESTABLISH WHICH KIPAR
1$: JSR PC,THRR; IS INVOLVED WITH THE RELOCATION OF
JSR PC,THRR; THIS BAD ADDRESS.
JSR PC,THRR
JSR PC,THRR
JSR PC,THRR

JSR R5,$PRINT; GO TO PRINT ROUTINE
JSR PC,02A
JSR PC,02A
JSR PC,02A
JSR PC,02A
JSR PC,02A
BR RETFR ;GO RESTART THE TEST WHICH WAS IN PROGRESS.

JSR R5, $PRINT ;GO TO PRINT ROUTINE

WORD NOTIND ;ME SCAN SO TELL

JSR R5, $PRINT ;THE ADDRESS GIVEN FOR THE ERROR. THEY
 WORD $CRLF ;RESTART THE TEST WHICH WAS IN PROGRESS
 MOV $AVPER, R3 ;WHEN THE ERROR OCCURRED.
 BR CONT

SAYPER: .WORD 0

PARR: JSR R5, $PRINT ;GO TO PRINT ROUTINE
 .WORD NOTIND

JSR R5, $PRINT ;TELL THE USER THAT IS THE CASE AND
 .WORD $CRLF

MOV $500, SP ;RESTART THE SCAN WHICH WAS IN PROGRESS

RESET JR PC, 3 WHERE

JSR PC, 3 WHERE

MOV #16, PC

TST MVA

MOSPAR

BEQ JSR PC, 3 WHERE

REPET2 ;WHEN THE ERROR WAS ORIGINALLY

JMP SPOT2 ;ENCOUNTERED.

MOV 000500

1470 014706 014707 005000

1470 014708 000005

JSR PC, 3 WHERE

MOV #16, PC

MVA

JMP REPE

JSR PC, 3 WHERE

MOSPAR

JMP SPOT2

ROUTINE USED TO SET THE TRAP VECTOR 114.

MP-VECT: MOV #PARERZ-.RO

ADD PC, RO

ADD #6, RO

MOV RO, #114

CLR #116

RTS PC

MESSAGES USED FOR THESE PARITY ROUTINES.

NOTIND: .ASCIZ 'SCAN COMPLETE'(15)(12)

MPMES: .ASCII (15)(12)'PARITY ENABLED'

$CRLF: .ASCIZ (15)(12)

PERMES: .ASCIZ 'PC=

NOMPAR: .ASCIZ (15)(12)'NO PARITY'(15)(12)
| .137  | 0.1640  | 2520  | 3375* | 3378* |
| .375  | 0.1296  | 2968  | 2972* | 3378* |
| .5414 | 0.1646  | 2969* |       |       |
| .5489 | 0.1534  | 2970  | 2973  | 2977* |
| .5/9  | 0.1338  | 2040  | 2164  | 2300  | 2980* |
| .9/13 | 0.12846 | 2068  | 2183  | 2412  | 3129* |