ALTOS

COMPUTER SYSTEMS

P R E L I M I N A R Y

ACS 8600 COMPUTER SYSTEM

SUPPLEMENT 3. EXECUTING DIAGNOSTIC PROGRAMS

REVISION C

APRIL 22, 1982

ACS Manual Part Number: 690-11853-001-C

TABLE OF CONTENTS

ALTOS ACS 8600 COMPUTER SYSTEM

SUPPLEMENT 3. EXECUTING DIAGNOSTIC PROGRAMS

PART 1. EXECUTING THE ALTOS DIAGNOSTIC EXECUTIVE PROGRAM (ADX)

GENERAL INFORMATION

This supplement provides instructions for executing diagnostic programs to prepare your system for installing the operating system and the selected application programs.

This supplement also presents the ADX program diskette loading procedure, and reviews ADX program functions.

KNOWN SYSTEM DEFICIENCIES

At the time this preliminary manual was printed, the following system deficiencies were known:

1. The COPY utility for copying diskettes is not presently available.

2. The Serial test is not presently available.

ADX PROGRAM FUNCTIONS

The ADX master diskette that came with the system contains a group of command programs that test ACS 8600 computer system components, format diskettes, and copy programs. These programs comprise the ALTOS Diagnostic Executive program (ADX). The ADX program must be run the first time the computer system is powered up, and every time a new component is added to the system.

WARNING

> **You should make a backup copy of this diskette before proceeding to test system components. To insure protection of your master diskette, you should make at least two copies. One copy, the ADX system diskette, is for daily use. The other, the back-up master, is only for making additional copies for daily use. You should never use your Altos master diskette for daily operations. It should be stored away from your computer area in a secure location to prevent accidental use.**

The utilities and tests comprising the ADX program are grouped into three categories, described in this section. Each program prompts you for every operation you need to perform. The programs perform these following functions:

1. They format diskettes and copy diskette utilities.

2. They test computer memory, printer, terminal, and floppy drives.

3. They test hard disk functions.

SUGGESTED INSTALLATION TEST ORDER

Run the memory test, MEM86, using the Short Comprehensive program.

Run the floppy diskette test for one write/read pass.

Run the hard disk test, HARD86, using the Write/Read test. Allow it to run through at least four test patterns, writing and reading.

These suggestions are for a minimum checkout. If you have time available, you may wish to run the comprehensive memory test longer and run a hard disk test overnight, as well as running some of the other tests.

LOADING AND RUNNING THE ADX SYSTEM

With your system powered on, place the ADX diskette (label up) into the floppy disk drive. Close the drive door and press the Reset button to load the ADX system. Your screen will display the monitor message:

```
     ALTOS COMPUTER SYSTEMS
      Monitor Version x.x

     Press any key to interrupt boot
```

Do not press a key. (If you do so accidentally, you will be given a choice of where to boot from. Choose the floppy.)

After some other information displays, the basic ADX identification shows, followed by the command directory:

```
   ALTOS DIAGNOSTIC EXEC 8600 - Version x.xx
    COPYRIGHT (c) xxxx ALTOS COMPUTER SYSTEMS


   ** D I A G N O S T I C   C O M M A N D   D I R E C T O R Y **

      DTEST        SER86        MEM86        HARD86
      FORMAT       COPY


     REQUEST:
```

To select one of these programs, enter the name and press Return. For example:

```
     FORMAT<CR>
```

Here is a brief description of these utilities and diagnostics:

COPY	Diskette copy utility. /* To be furnished. */
DTEST	Basic test for diskette drive, diskettes.
MEM86	Memory test routines, including quick tests, fault isolation tests, and long exercise routines.
HARD86	A collection of hard-disk utility and test routines. Utilities include formatter, routine to flag bad sectors; diagnostics include quick tests, fault isolation tests, and long exercise routines.
SER86	Serial I/O tests for the serial ports. Includes a printer test. /* To be furnished */

DATA ENTRY CONVENTIONS

In this document, data that the user enters is shown by underlining. For example, y. An alternate way is to name the key, such as "Press the Escape key". The key name is capitalized. Pressing the Return key is also shown by "<CR>". Examples:

Reply Y or N and press the Return key.

Reply y<CR> or n<CR>.

Select 1<CR> for continuous display, 2<CR> for display at the end of the pass.

DATA ENTRY; CHANGING INVALID DATA

The ADX programs accept alphabetic data in either lower case or upper case. For example, either mem86<CR> or MEM86<CR> is acceptable.

The ADX programs use, in general, the same conventions used by the CP/M operating system for cancelling or changing data before it is entered. (Pressing the Return key enters data.) Here are some basic ways of modifying or correcting entries:

To erase the last character typed: Depending on your keyboard, use the Rubout or Delete key, or Cntrl-H. The erased character may be echoed; use Cntrl-R to see the true entry.

To erase the whole entry, use Cntrl-U or Cntrl-X.

To re-display the current entry, use Cntrl-R.

DISKETTE FORMAT PROGRAM: FORMAT

The FORMAT utility program formats, or reformats, diskettes. This program erases all data previously stored on a diskette.

Follow this procedure to execute FORMAT:

Insert the ADX diskette in the diskette drive, and press the Reset button. The ADX menu displays on the terminal screen. Enter FORMAT<CR> after the "REQUEST" prompt.

The terminal displays the FORMAT prompt message:

```
.... ALTOS FLOPPY DISK FORMAT ROUTINES ....

  1.  Standard single density format
  2.  Double density format for CP/M, XENIX, or diagnostic disk
  3.  Double density format for MP/M
  4.  End this program

Select format option by number: n
```

Explanation: Select according to the intended use of the diskette. The only difference between selections 2 and 3 is the skewing factor.

After you have selected 1, 2, or 3, this appears:

```
     Place disk to be formatted in drive A:,
     remove diagnostic disk if necessary.
     When ready to proceed,
     Reply with <CR>:
```

Insert the diskette into the drive, close the loading door, and press Return. The system will start formatting the diskette, displaying the cylinder numbers as it proceeds.

```
     Cylinder xx    (cylinder number goes from Ø to 77)
```

When formatting is complete, the program repeats the selection choice (1-4) shown above. You can format as many diskettes as you wish.

To exit, select 4. This displays:

```
     Place a system or diagnostic disk in 'A:'
     Hit <CR> when ready
```

Insert the appropriate diskette and press Return to boot from that disk.

FLOPPY DISK DRIVE TEST PROGRAM: DTEST

This is a simple test to verify the basic functioning of the floppy disk drive, or the diskette in it.

Follow this procedure to execute DTEST:

Insert the ADX diskette in the diskette drive, and press the Reset button. The ADX menu will display on the terminal screen. Enter DTEST<CR> after the "REQUEST" prompt.

The terminal will display the DTEST prompt message:

```
ALTOS COMPUTER SYSTEMS
CP/M-86 FLOPPY DISK TEST AND ANALYSIS, VERS: x.xx

**** HIT "ESC" TO EXIT ****

LOAD SCRATCH DISKETTES IN DRIVE(S) TO BE TESTED
HIT <CR> WHEN READY TO PROCEED
```

Use either a formatted blank diskette or a diskette with data that you are willing to have written over. When you press Return, you see:

```
ARE DISKETTES REALLY SCRATCH?? Y OR N
```

If you reply N, you are told to load a scratch diskette. When you reply Y, you see:

```
ENTER 'S' FOR SINGLE DENSITY, 'D' FOR DOUBLE.
```

After you reply S or D, the test begins. First a data pattern is written on all sectors of the diskette, and then the entire diskette is read and compared. The pattern is 'E5' in hexadecimal, which is 1110 0101 in binary. The screen displays:

```
STARTING WRITE/READ DATA INTEGRITY TESTS
..        WRITE PHASE
..        READ PHASE
```

The "WRITE PHASE", "READ PHASE" indicate the start of each operation. The test continues to alternate these two phases until you press the Escape key (ESC). The minimum complete test is to do one write phase and one read phase. When "WRITE PHASE" displays for the second time, this minimum test is complete.

Pressing Escape returns to the ADX menu.

If an error message could possibly be caused by a bad diskette, try another diskette to see whether it is the diskette or the drive.

DISKETTE COPY UTILITY: COPY

This utility copies a diskette by reading the entire contents into memory and, when that diskette has been replaced in the drive, writing those contents to the new diskette.

/* To be furnished */

HARD DISK UTILITIES AND DIAGNOSTICS: HARD86

HARD86 is a collection of hard-disk utility and diagnostic programs. All ADX programs for the hard disk are in HARD86.

Follow this procedure to execute HARD86:

Insert the ADX diskette in the diskette drive, and press the Reset button. The ADX menu will display on the terminal screen. Enter HARD86<CR> after the "REQUEST" prompt.

The terminal will display the HARD86 prompt message:

```
     *** Hard Disk (8") Test Facility Vx.x   Mon dd yyyy ***
     Specify the hard disk to be used.  Press <CR> for defaults.
       First disk (default) = 1; Second = 2:
```

Select the first disk or the second (add-on) disk. This selection is done only at the start of of HARD86, and is used for any program selected from the menu. To change the disk selection, return to the ADX menu and request HARD86 again.

The next prompt is:

```
     Specify size.
       20 Megabytes (default) = 2; 40 megabytes = 4:
```

If you reply incorrectly to either prompt, you are given an error message and the prompt is repeated so you can enter a correct response. After you have specified the disk and its size, the menu is displayed for you to select a program. At the completion of any program, this menu is displayed for another choice.

```
     *** Hard Disk (8") Test Facility Vx.x mon dd yyyy ***

     1.  Format Disk Drive
     2.  Verify Addresses for all Sectors on Disk
     3.  Seek test with optional Verify
     4.  Write entire Disk
     5.  Read entire Disk
     6.  Set Flag Byte for a Specific Sector
     7.  Hard Disk Write/Read Error Test
     8.  Miscellaneous Functions
     9.  Terminate this Test Series.

     Select required function by number:
```

Enter a choice from 1-9 and press <CR>. Here is a brief description of the functions. Operating instructions for program choices 1 to 8 follow this brief description, in the same order as the menu above.

1. Format Disk. Usually, the disk has been formatted before the system is shipped from the factory. Formatting places the

cylinder and sector addressing on the disk and blocks out data areas. It must be done before data can be written on the disk. It is not usually necessary to run this program in the field. Formatting destroys any prior data on the disk. If the disk is re-formatted, "Flag Bad Sector" (selection 6) would usually have to be run also.

2. Verify Addresses. This reads the addressing information for every sector of every cylinder on the disk to verify their availability. No data is read.

3. Seek Test. This test allows you to specify two cylinders which the disk controller accesses alternately and continually. The main use of this test is for fault isolation. It would typically be run while using electronic test equipment.

4. Write Entire Disk. This writes a specified pattern to all sectors of the disk.

5. Read Entire Disk. This reads the data from every sector on the disk, and if desired can compare that data to a specified pattern. It can be used after the program above, "Write Entire Disk", to verify that the write operation was successful.

6. Set Flag Byte for a sector. This utility flags a sector as bad, that is, not to be used for storing data. Usually, any bad sectors have already been flagged before the system was shipped. This utility can be used in the field in case a bad spot develops on the disk, or if the disk has been re-formatted.

7. Hard Disk Write/Read Test. This test writes a specified pattern over the entire disk, reads it back, and compares. You can specify one pattern or have the program use a comprehensive series of patterns. The test can also be used as an exerciser, to work the disk for a long time (such as overnight), and report on the results at the end.

8. Miscellaneous Functions. There are two funtions. The first suppresses or enables the display of the disk status error message. This is ordinarily enabled, but might be disabled while running some repetitive operation and checking it with electronic test equipment. The second function displays the contents of a selected sector on the screen in hexadecimal and ASCII.

9. Terminate Test. This returns to the basic ADX menu.

Format Disk Drive

```
**************************************
*  WARNING: THIS HARDTEST FUNCTION *
*  CHANGES DATA ON THE HARD DISK   *
*  AND MAY CAUSE LOSS OF USER DATA *
**************************************
```

Usually, the disk has been formatted before the system is shipped from the factory. Formatting places the cylinder and sector addressing on the disk and blocks out data areas. It must be done before data can be written on the disk. It is not usually necessary to run this program in the field. Formatting destroys any prior data on the disk.

Formatting also erases flag byte indications of bad sectors. These sectors were flagged when the system was prepared for shipment, using the error map provided with the hard disk. The system does not write data on flagged sectors. Sectors previously marked as bad will now be considered as valid. Unless these sectors are re-marked as bad sectors, data written on these bad sectors may be lost. The error map is shipped with your system, taped inside.

If you re-format your hard disk, you will probably also need to run the "Set Flag Byte" program, selection 6.

When you select program 1 and press Return, the screen displays:

```
*** DO NOT RUN THIS TEST WITHOUT PERMISSION FROM -ALTOS-
CUSTOMER SERVICE *** CALL 408 946 6700
Do you want to continue?
```

Password entry is used as protect the system and the data stored on it. To obtain the password call your distributor. Call ALTOS customer service only if your distributor suggests it for some reason. Before a password is given to you, the distributor or Altos will attempt to determine whether formatting the hard disk is actually necessary.

If you are not continuing, enter n<CR>. You will return to the HARD86 menu.

If you are continuing, enter the password and press <CR>. The screen will display:

```
*** THIS TEST WILL ERASE FILES ON THE HARD DISK. ***
Do you want to continue? (Y or N):
```

To continue, enter y<CR>.

The format process starts. The cylinder number is shown as each cylinder is formatted. The count goes from 0 to 255.

When formatting is complete, the program returns to the HARD86 menu.

NOTE: Flagging Bad Sectors

Before putting data on the newly-formatted hard disk, flag any bad sectors. The error map for your hard disk is taped to the unit. It is usually found taped to the bottom cover under the hard disk drive; however, it may be taped in some other location near the drive. Opening the system and removing the map is a job for an experienced technician. To flag the bad sectors, see selection 6 of HARD86, "Set Flag Byte".

Verify Addresses for All Sectors on Disk

This test reads the addressing information for every sector of every cylinder on the disk to verify their availability. No data is read; nothing is written or erased.

The identification area of every sector on the hard disk contains addressing information which consists of the cylinder, head/drive, and sector numbers. This test reads and verifies those numbers. If those numbers cannot be read, the sector cannot be used.

The identification area also contains a flag byte location where sectors can be flagged as bad, that is, as not reliable for storing data, and a CRC value. The CRC is a Cyclic Recundancy Check value which is developed by the circuitry when it writes data on the sector. When data is read from the sector, a CRC value is developed and then compared to the stored value as a check on data integrity.

When you select program 2 and press Return, the screen displays:

```
     Press any key when ready to start this test.
```

The verification process starts. The cylinder number is shown as each cylinder is verified. The count goes from Ø to 255.

When verification is complete, the program returns to the HARD86 menu.

Error conditions:

BAD SECTOR. The sector has been flagged as a bad sector.

SECTOR NOT FOUND. Bad address information. See the note below.

CRC ERROR. The CRC character itself is not valid; it has a parity error. (Note that this is different from the usual meaning of "CRC ERROR". The data on the sector has not been read to develop a CRC character for checking against the stored CRC character. "CRC ERROR" usually means a mis-comparison of CRC values.) See the note below.

Note: Errors in the Identification Area

An SECTOR NOT FOUND or CRC ERROR suggests two possibilities. First, the location where the information is stored in the identification area is not reliable for data storage. Two, the addressing or CRC data has been corrupted by some outside cause, such as an electrical spike or magnetic field.

The CRC ERROR, if caused by outside interference, should go away when new data is written to the sector. If it persists for that sector, the sector should be flagged as bad.

A sector which is "Not Found" can be found in order to flag it as bad. The only other way to deal with a sector which has incorrect identification information is to reformat, which means reformatting the entire disk. Reformatting is a major undertaking. See "Format Disk", selection 1, for guidelines as to when reformatting would be appropriate.

To flag a sector as bad use the "Set Flag Byte" program, selection 6.

Seek Test with Optional Verify

This test allows you to specify two cylinders which the disk controller accesses alternately and continually. It verifies the address information at head Ø, sector Ø, of each cylinder specified, unless verification is disabled. The main use of this test is for fault isolation. It would typically be run to check for problems in doing the seek operation or while using electronic test equipment to check details of circuit functioning.

When you select program 3 and press Return, the screen displays:

```
     Press any key when ready to start this test.
```

The program then prompts you to specify two cylinder addresses to set the test boundaries.

```
     Enter first cylinder number to seek:
```

Enter a number from Ø to 255, and <CR>. The next prompt is:

```
     Enter last cylinder number to seek:
```

Enter a number from Ø to 255.

The maximum seek distance is from Ø to 255. The minimum seek distance would be to specify the same cylinder; usually there would be no reason to do this.

The next prompt is:

```
     Do you want test verification of the Cylinder Numbers?  (y
     or n):
```

In general use, select yes by entering y<CR>. As the program accesses each cyliner, it reads and verifies the sector Ø address information of each cylinder. Disabling the verification by replying n<CR> causes the test to run somewhat faster, and would be done only when using electronic test equipment to check some part of the seek process where maximum speed is desired and the verification of address does not matter.

The seek process starts as soon as "y" or "n" is specified. The cylinder numbers are displayed alternately as each seek is performed.

The seek test continues until you press the Escape key, "ESC", which cause return to the ADX menu.

Error conditions:

Bad Sector. Sector Ø of the specified cylinder has been flagged as a bad sector. This message is disabled if "no verification" is specified.

Record Not Found. There are two possible cases.

1. The identification information for sector 0 of the specified cylinder is bad, and has not been flagged as bad. (The identification consists of cylinder, drive/head, and sector addresses.)

2. The system is not performing the seek operation properly. To verify that it performance, select another cylinder and retry.

These messages are disabled when "no verification" is specified.

Write Entire Disk

This writes a specified pattern to all sectors on the disk. A companion program, "Read Entire Disk", allows verification of the success of this program.

When you select program 4 and press Return, the screen displays:

```
*** THIS TEST WILL ERASE FILES ON THE HARD DISK ***
Do you wish to continue? (y or n):
```

If you reply n<CR>, the program returns to the HARD86 menu. If you reply y<CR>, this prompt displays:

```
Do you want to write a specific pattern? (y or n):
```

If you reply n<CR>, the default pattern is E5E5 in hexadecimal. E5 is 1110 0101 in binary. If you reply y<CR>, you are prompted to choose a pattern:

```
Patterns can be specified by entering:
  *1 - for 256 pattern (hex 00....FF)
  one of two byte pattern - enter pattern in binary, octal,
  decimal, or hex.
Select pattern:
```

Specifying *1<CR> selects a 256-byte block of all hexadecimal values from 00-FF as the pattern.

The one or two byte patterns are specified as follows. For all specifications, leading zeros are supplied:

Binary: Up to 8 or 16 binary digits, 0s and 1s, ending in "B" or "b". Example: 1100101001010011b<CR>.

Octal: Up to four octal digits (0-7) ending in "O" or "o" (alphabetic). Example: 7007o<CR>.

Decimal: One to three numbers, 0 - 255. The number is converted to its hexadecimal equivalent. Example: 127<CR>.

Hexadecimal: One to four hexadecimal values (0-9, A-F), ending in "H" or "h". Example: A55Ah<CR>.

When the pattern has been selected by specification or by default, the program expands it to fill a 512-byte sector and writes it to all sectors. It displays the cylinder number as it writes to the disk. The cylinder count goes from 0 to 511.

When the program is finished, it returns to the HARD86 menu. A companion program, "Read Entire Disk" (selection 5), allows reading and verifying the pattern just written.

Error conditions:

BAD SECTOR. Data was written to a sector flagged as bad.

RECORD NOT FOUND. The sector contains bad identification data. The sector has not been flagged as bad. (The identification data consists of cylinder, head/drive, and sector addresses.) If this error and a CRC Error (below) are both present, this error is shown.

CRC ERROR. The CRC is a Cylic Redundancy Check character developed to check the integrity of the data written. As the data is written to the sector, the CRC character is automatically developed by the circuitry and is written into the identification aread, following the address information. This message means that an error occured in the development of the CRC character.

Note: This test will not detect that a bad data block has been written, so long as the sector identification is good and the CRC character is developed correctly. The data may be stored incorrectly or the CRC may be stored incorrectly. Running the "Read Entire Disk" test will verify that the pattern was correctly written.

Read Entire Disk

This reads the data from every sector on the disk, and if desired can compare that data to a specified pattern. It can be used after the program, "Write entire disk", to verify that the write operation was successful.

Before each sector is read into a memory buffer, the buffer is set to all ones (hexadecimal FF). This is done to ensure that the contents are read accurately.

When you select program 5 and press Return, the screen displays:

```
Hard Disk read display options are:

1. DO NOT Display data if any error,
2. Display data only if a STATUS error,
3. Display data only if a COMPARE error,
4. Display data if a STATUS or COMPARE error.

Select option by number:
```

For selections 1 and 2 the sectors are only read; for selections 3 and 4, sectors are read and compared to a specified pattern. The pattern would usually be one written by the "Write Entire Disk" program, selection 4. Here are the details of the four options:

1. All sectors are read, no errors are displayed.

2. All sectors are read, but the data is not checked. Status errors are displayed. They occur when the controller cannot locate or properly identify a sector, or when there is a CRC error (the CRC value is invalid). Error display shows both status and sector data.

3. You select a pattern for comparison. Each sector is read and compared against that pattern. Comparison errors are displayed; the display shows both status and sector data. Status errors are not displayed, except for the unlikely case of a CRC error where the data is correct and the CRC value is invalid.

4. You select a pattern for comparison. Each sector is read and compared against that pattern. Both status and comparison errors are displayed; the display always includes sector data.

If you select option 3 or 4, you are prompted to specify a pattern. If you wrote a pattern with the "Write Entire Disk" program, choose the same pattern. The default pattern of the write program is hexadecimal E5; there is no default read pattern. If you wrote that pattern, specify ØE5h.

Here is the prompt sequence for specifying a pattern:

```
     Patterns can be specified by entering:
       *1 - for 256 pattern (hex 00....FF)
       one of two byte pattern - enter pattern in binary, octal,
       decimal, or hex.
     Select pattern:
```

Specifying *1<CR> selects a 256-byte block of all hexadecimal values from 00-FF as the pattern.

The one or two byte patterns are specified as follows. For all specifications, leading zeros are supplied:

Binary: Up to 8 or 16 binary digits, 0s and 1s, ending in "B" or "b". Example: 1100101001010011b<CR>.

Octal: Up to four octal digits (0-7) ending in "O" or "o" (alphabetic). Example: 7007o<CR>.

Decimal: One to three numbers, 0 - 255. The number is converted to its hexadecimal equivalent. Example: 127<CR>.

Hexadecimal: One to four hexadecimal values (0-9, A-F), ending in "H" or "h". Example: A55Ah<CR>.

When the pattern has been selected, the program expands it to fill a 512-byte buffer for comparison. Then it reads each sector and compares the pattern to the data.

The program displays each cylinder number as it reads the disk. The cylinder count goes from 0 to 255.

When the program is finished, it returns to the HARD86 menu.

Status Error Conditions:

BAD SECTOR. Data was read from a sector flagged as bad.

RECORD NOT FOUND. The sector contains bad identification data. The sector has not been flagged as bad. (The identification information consists of cylinder, head/drive, and sector addresses.) If this error and a CRC Error (below) are both present, this error is shown.

CRC ERROR. The CRC is a Cylic Redundancy Check character developed to check the integrity of the data stored on the disk. When the data on the sector is read, the circuitry computes a CRC character for it and compares it to the CRC stored at the time the data was written. This message means that the CRC character is not correct for the data.

COMPARE ERROR. The data does not match the pattern specified. The actual sector data is displayed.

Set Flag Byte for a Specific Sector

This utility flags a sector as bad, that is, not to be used for storing data. Usually, any bad sectors have already been flagged before the system was shipped. This utility can be used in the field in case a bad spot develops on the disk, or when the disk has been re-formatted and it is necessary to re-flag known bad sectors. The flag location is in the identification area, which also holds the sector addressing and the CRC characters.

Sectors to be flagged can be specified in two ways:

1. As shown on the error map provided with your system, by track, head, byte count, and length in bits.

2. By cylinder, head, and sector address.

For information on removing the error map from within your system, see the note at the end of this section.

When you select program 6 and press Return, the screen displays:

```
*** THIS TEST WILL ERASE FILES ON THE HARD DISK. ***
Do you want to continue? (y or n):
```

If you reply n<CR>, the program returns to the HARD86 menu. If you reply y<CR>, this prompt is displayed:

```
Press any key when ready to start this test.
```

When you press a key, the program displays:

```
Hard Disk "Flag Bad Sector" Options are:

1. Disk Error Map
2. Cylinder, Head, Sector
```

Enter 1 or 2 and press <CR>.

For option 1, you can read the information off the disk error map and enter it in the same form. You will be prompted to enter the track (TRK), head (HD), byte count and length in bits. Enter the information and press <CR>.

For option 2, you can enter the information in the form used by error messages given by other HARD86 programs, that is, by cylinder, head, and sector addresses.

When you have flagged a sector you are asked whether to continue or exit:

```
Do you want to continue this test? (Y or N):
```

If you reply y, the program will prompt you for the next bad sector to be flagged. When you have flagged all bad sectors you wish to, reply n<CR> and the program will return you to the HARD86 menu.

Note: The Hard Disk Error Map

This map is taped inside your system when it is shipped from the factory. Any bad sectors shown on the map were flagged before the system was shipped. The only time this map would be needed to flag sectors is after reformatting the hard disk.

The map is usually found taped to the bottom cover under the hard disk drive, but may be in other locations near the disk drive. Opening the system and removing it is a matter for a skilled technician.

Note: Assigning Alternate Sectors

This program flags bad sectors but does not allocate alternate sectors to be accessed in their place. This function is handled differently by different operating sytems. See the Operating System Supplement in this manual for information.

Hard Disk Write/Read Error Test

This test writes a specified pattern over the entire disk, reads it back, and compares. You can specify a pattern or have the program use a comprehensive series of patterns. The test can be used as an exerciser, to work the disk for a long time (such as overnight), and report on the results at the end.

This test has two phases. The first writes and reads a variety of patterns to all sectors of the disk. This phase continues until you end it by pressing the Escape key (ESC). The program then completes its current pass and begins the second phase.

In the second phase, the program erases all sectors and automatically flags all "bad sectors". It will display the final error count and give you an opportunity to print it.

Explanation of terminology:

Soft Error. An unsuccessful attempt to read data, shown as a CRC error. The operation is retried. If the operation succeeds on the first or second retry, each prior failure is counted as a "soft error".

Hard Error. If the third retry at a read fails, the sector is considered to have a "hard error".

Bad Sector. A sector that has a hard error is flagged as a "bad sector", not to be used for data storage. This is done in the final phase of the test. (Selection 6, "Set Flag Byte", can also be used.)

When you select program 7 and press Return, the screen displays:

```
*** THIS TEST WILL ERASE FILES ON THE HARD DISK ***
Do you want to continue? (y or n)
```

If you reply n<CR>, the program returns to the HARD86 menu. If you reply y<CR>, the display options are shown.

```
Display Options:

   1. Continuous display on terminal.
   2.  Display error summary at the end of each pass.
   3.  Display error summary only at the end of the test.

Select option by number
```

Option 1 maintains a display on the terminal, updating it with changes as they occur. Option 2 writes an updated display at the end of each write pass and read pass. Option 3 allows the test to be run without a terminal. You can remove the terminal after starting the test and reconnect it when you wish to end the test.

This prompt displays after you have selected your option:

```
     Do you want to display data if a CRC error? (y or n):
```

In case of a CRC error, you can choose whether to display the data buffer. If you intend to run the test without being present, reply n<CR>, because the program waits after displaying the buffer for you to tell it to continue.

The next prompt is:

```
     Do you want to write specific patterns? (y or n):
```

If you reply n<CR>, the program will use four pre-defined data patterns. If you reply y<CR>, this prompt shows:

```
     As many as four (4) patterns may be specified, as follows:
     Enter on or two byte pattern in binary, octal, decimal, or
     hex. Press RETURN to bypass a pattern selection:
```

Patterns are specified as explained in the instructions for one or two byte patterns in the Write test and the Read test preceeding these instructions. Pressing <CR> specifies one of the pre-defined patterns. When all patterns are specified, they are displayed in hexadecimal. The examples are the pre-defined patterns:

```
     Pattern #1 revisited: E5E5H
     Pattern #2 revisited: 5555H
     Pattern #3 revisited: AAAAH
     Pattern #4 revisited: FFFFH

     Press any key when ready to continue this test.
```

Press a key and the program starts writing the first pattern. It will continue writing and reading until you stop it by pressing the Escape key (ESC). At that point it displays:

```
     Finishing Pass
```

It will stop at the end of the write or read pass it is on and display the results.

```
  Pass count:
  Pattern:
  Cylinder:

  Soft Errors
  Chars: E5E5H    CMP Err   Ø   CRC Err    Ø    RNF Err Ø   BAD SEC   Ø
  Chars: 5555H    CMP Err   Ø   CRC Err    Ø    RNF Err Ø   BAD SEC   Ø
  Chars: AAAAH    CMP Err   Ø   CRC Err    Ø    RNF Err Ø   BAD SEC   Ø
  Chars: FFFFH    CMP Err   Ø   CRC Err    Ø    RNF Err Ø   BAD SEC   Ø

  Hard Errors
  Chars: E5E5H    CMP Err   Ø   CRC Err    Ø    RNF Err Ø   BAD SEC   Ø
  Chars: 5555H    CMP Err   Ø   CRC Err    Ø    RNF Err Ø   BAD SEC   Ø
```

```
 Chars: AAAAH   CMP Err  Ø  CRC Err  Ø   RNF Err Ø  BAD SEC  Ø
 Chars: FFFFH   CMP Err  Ø  CRC Err  Ø   RNF Err Ø  BAD SEC  Ø
```

The next prompt allows you to have a hard copy of the results:

```
     Do You want to print (LP) the errors? (y or n):
```

The program finishes by erasing the disk. It writes a pattern of hexadecimal E5s to all sectors. At this time it also flags as bad any sectors that had hard errors.

Miscellaneous Functions

There are two funtions. The first suppresses or enables the display of the disk status error message for HARD86 programs. This is ordinarily enabled, but might be disabled while running some repetitive operation and checking it with electronic test equipment. The second function displays the contents of a selected sector on the screen in hexadecimal and ASCII.

When you select program 8 and press Return, the screen displays:

```
     1.  Select disk error status display option
     2.  Display a sector
     3.  Terminate this series of tests

     Select required function by number:
```

1. If you select 1<CR>, this prompt shows:

```
     Do you want the disk error status message displayed? (y or
     n)
```

Reply y<CR> or n<CR> to enable or supress display of status errors for HARD86 programs. The program then returns to the HARD86 menu.

2. If you select 2<CR>, this prompt shows:

```
     *    DISPLAY HARD DISK SECTOR    *

     Enter Cylinder Number:
```

Enter a value and press Return. The range is 0-511.

```
     Enter Head Number:
```

Enter a value and press Return. The range is 0-3 for 20-megabyte disk drives, 0-7 for 40-megabyte drives.

```
     Enter Sector Number:
```

Enter a value and press Return. The range is 0-16.

The contents of the sector are then displayed on the screen, shown in hexadecimal on the left, ASCII on the right if displayable. Twenty four bytes are shown per line. Decimal numbers on the left aid in locating the exact displacement of any byte in the sector, from 0 to 511.

3. If you select 3<CR>, the program returns to the HARD86 menu.

PART 2. EXECUTING THE MEM86 PROGRAM

NOTE

This description of the MEM86 Program is identical to that presented in Revision B of the ACS 8600 Diagnotics Supplement dated March 22, 1980.

1.0 Overview

The MEM86-0 memory diagnostics are designed and written to help the user feel comfortable with this system and all of its' intergal parts, wheather he be a computer design engineer or a first time user of computers.

MEM86-0 executes a variety of dynamic tests to verify the integrity of the ACS8600 system RAM, ECC hardware, and memory management hardware. The first test that is usually excuted, upon system and/or RAM initialization, is a comprehensive memory test. This test allows the user to identify any faults, hard or soft, in the memory and associated hardware. Once this test has been accomplished and any faults identified, the user can then initate a number of selective tests that validate the hardware. The tests are broken down into moduals enabling the user to independently exercise any, or all, portion(s) of the system he may choose. Each modual is fully operator selectable. The operator can not only select the modual to be tested, but what type of error reporting action that should be taken to identify any errors within the system. This means of selective diagnostics allows the user to exercise the hardware anywhere within the system.

MEM86-0 assumes that the hardware is operating properly and that it passes the power-on self test. It assumes that any RAM used for program code or data variables is secure and error free for brief periods. MEM86 will scrub itself to prevent the accumulation of soft RAM errors in its program code/variable memory during prolonged program operation. Loading MEM86-0 distroys, or dumps, the proprietry inputs (e.g. segment register file, reset bootstrap program jump, monitor, etc.) loaded upon system "boot up" and allows the system memory to be fully checked out. As the user completes the MEM86-0 diagnostic testing,the system will automatically reboot the propriitary information into the approaite memory locations.

Operator Interaction

2.Ø Operator Interaction

The operator communicates with The ACS86ØØ via a series of menus. Once the operator has checked the power supplies and is assured that the system is viable, he will:

Load the system diskette into the diskette drive and boot the system up. One should expect to see the following appear on the screen:

Next, the user should bring up the directory of programs that are on the diskette. This is accomplished with the command (A>DIR). The directory will appear on the screen just below the banner that appeared for the initial boot up of the system.

Select the memory diagnostic test for the ACS86ØØ (command A>MEM86). The CRT screen will look like this:

```
A>MEM86

press any character to stop boot (this statement will be on the
screen during MEM86 boot up.  Once the system diagnostic is in
place in the system this statement will disappear)


             *** ACS86ØØ Memory Verification v1.X ***
                      Processor Mode Tested
                 Program Memory:XXXXtoXXXX
                 Test Memory:XXXXtoXXXX


Main Menu
      (A)  Continuous Comprehensive Test
      (B)  Short Comprehensive Test
      (C)  ECC Menu
      (D)  RAM Menu
      (E)  MGR Menu
      (F)  NDP Menu
      (G)  Relocate Program (not included in this revision)
      (H)  Error Handling Menu
      (I)  List Device Menu (not included in this revision)
      (J)  Message Severity Menu
      (K)  Exit to Diagnostic Disk
```

Message Severity Menu
- (A) print errors
- (B) print errors, warnings
- (C) print errors, warnings, information
- (D) print errors, warnings, information, traces

Error Handling Menu
- (A) Continue (ignore all errors)
- (B) Pause (for operator selection when error occurs)
- (C) Loop (retry all errors)

RAM Test Menu
- (A) All (default)
- (B) Checkout Test
- (C) Byte Ripple Test
- (D) Word Ripple Test
- (E) Odd Word Ripple Test
- (F) Check Bit Ripple Test

MGR Test Menu
- (A) All (default)
- (B) Page Base Ripple Test
- (C) Device Access Violation Test
- (D) User Mode Program Operation Exception Test

The main menu lists diagnostics that can be accomplished with this program. Definitions of these are as follows:

Continious Comprehensive Test-a continious test that will circulate a testing pattern, within the hardware of the system,to see if there are any hard or soft faults of any type. This test, and the short comprehensive test, use the message severity menu to pyrametrically denote the type of error, its' cause and where the fault lys in the system. This test will contine the test until the operator stops it by pressing the escape (ESC) key.

Short Comprehensive Test-this test is identical to thd Continious Comprehensive Test except that the test runs through its cycle once and stops.

ECC Menu-this menu allows the user to select additional tests to clear any ECC errors that might have occured within the system. (See section 4.0 for futher explination of the individual tests.)

RAM Menu-this menu allows the user to select additional tests to clear any RAM errors through exercise and/or find where, within the RAM, the error is located. (See section 3.0 for further explination of the individual tests.)

MGR Menu-this menu allows the user to select additional tests to clear any memory manager errors through exercise and/or find where, within the memory manager environment, the error is located. (See section 5.0 for further explination of these tests.)

NDP Menu-this menu allows the user to select additional tests to clear any 8087 errors that may have occured.
NOTE:This is an optional item and not included in this particular system.

Relocate Program-this program is not currently available

Error Handling Menu-this menu allows the user the flexibility to have the system react to any error in a predetermined manner. Selection of A, or return, tells the system to "Continue" and not pause if an error is discovered. The system will store any errors that may have occured and print them out at the completion of the current Continous/Short Comprehensive Test in progress. The detail of the error print out, in each part of this test, is determined by the users' Message Severity Menu selection. The selection of B will cause the system to stop each time an error is detected. The user has the option, at that time, to select another test or resume the test in progress. Selection of C causes the system to loop on any error until the error is cleared, or the user resumes the test in progress.

List Device Menu-this menu is not currently available

Message Severity Menu-this menu allows the user to select the depth of information, set up pyrametrically, that will be displayed for each error that occurs. The following is a sample of such a printout:

```
*** ERROR -- MGR boundary write violation on page 08 --
    align=05,loc=0807F, b/w=0
*** ERROR -- MGR boundary write violation on page 08 --
    align=06,loc=0807F, b/w=1
*** ERROR -- MGR CS seg read violation on page 09 --
    align=00, loc=09000, b/w=1
*** ERROR -- MGR CS seg write violation on page 09 --
    align=00,loc=09000, b/w=1
```

Exit to Diagnostic Disk-this statement is self explainitory

NOTE: We would encourage the user to look at Appendix A of this section to further understand what is normal and what can be expected during the operation of this system.

The escape (ESC) key may be pressed at any time to abort the current operation in progress and return to the main menu. The program may take up to 45 seconds to respond, depending upon the task it is currently involved in, to the escape key. If this does not release the test proceedure and bring it back to the main menu readout, it may be necessary to reboot the system.

3.0 RAM Tests

These tests are designed to locate and identify hard faults, e.g.opens and shorts, in the RAM data, address lines, RAM chips, etc., that may have occured anywhere within the RAM. The tested RAM, excluding 0000(H)-03FF(H) and memory containing the MEM86 program code or data (0400(H)-0C00(H) and FE000(H)-FFFF0(H)), is fully exercised and accessed. These tests are fully functional without the aid of the ECC hardware and will list any errors that have occured prior to the ECC hardware effecting a change or modification to the system. NOTE: The ECC hardware cannot be fully disabled by the user at this time.

3.1 Checkout Test

This test is designed to be a simple, overall checkout for the RAM and its' associated hardware. Not only is the RAM and its' associated hardware exercised, but the I/O capability of the microprocessor, its' Bus Interface Unit (BIU) and Execution Unit (EU) are also throughly tested. This test accomplishes this by supplying data to the processor, having it executed by the EU, interfaced to the RAM by the EIU and addressed to and retrieved from both the high and low banks of the 1 Mbyte RAM. This process blankets the RAM, its' address and data lines on both high and low banks, insuring that the odd/even storage/retrieval capability of the system is good and that there are no hard or soft errors within the RAM. The testing algorithm is as follows:

(1) Write a numerically corresponding 16bit word into all memory locations. (e.g. 0000(H) to location 0, 0001(H) to loc. 2, 0003(H) to loc. 4, ..., (N-1)/2 to loc. (N-1), N/2 to loc. N)

(2) Read each word in memory and verify that it is the correct value.

(3) Starting at the highest memory address and working in reverse order, write the complement, or opposite, of the numerically corresponding 16-bit word into all memory locations (e.g. (FFF(H)-N/2) to loc. N, (FFFF(H)-(N-1)/2) to loc. (N-1),

(4) Read each word in memory and verify that it is the correct value.

3.2 Byte Ripple Test

This test looks for byte exclusive data bit errors. The memory is accessed, one byte at a time, and verified. The testing algorithm is as follows:

(1) Write ØØ(H) into all memory locations.

(2) Verify, in ascending address order, that ØØ(H) is read from each memory location. Once this has been verified, write FF(H) into each memory location in ascending address order.

(3) Verify, in descending address order, that FF(H) is read from each memory location. Once this has been accomplished, write ØØ(H) into each memory location in descending address order.

3.2 Word Ripple Test

This test looks for word exclusive data bit errors. The memory is accessed, a word at a time, and verified. The testing algorithm is as follows:

(1) Write ØØØØ(H) into all memory locations.

(2) Verify, in ascending address order that ØØØØ(H) is read from each memory location. Once verified, write FFFF(H) into each memory location in ascending address order.

(3) Verify, in descending address order, that FFFF(H) is read from each memory location. Once verified, write ØØØØ(H) into each memory location in descending address order.

3.3 Odd Word Ripple Test

This test looks for data bit errors within odd words. Each word of data is fetched from the odd memory location boundaries within the high bank of the memory and verified. This test is not applicable if one only has 5ØØK of memory. The memory that would normally be tested would be the upper bank or second 5ØØK of RAM. The testing algorithm is as follows:

(1) Write ØØ(H) into all high bank memory locations.

(2) Verify that ØØ(H) has been read from memory location Ø. Write FF(H) into memory location Ø.

(3) Verify, in ascending address order, that ØØØØ(H) was read from each memory location. Once verified, write FFFF(H) into each memory location in ascending order.

(4) Verify that ØØ(H) was read from the last even memory location. Write FF(H) into this location.

(5) Verify that FF(H) was read from the last even memory location. Write ØØ(H) into this location.

(6) Verify, in descending address order, that FFFF(H) was read from every odd memory location. Once verified, write ØØØØ(H) into each odd memory location in descending order.

(7) Verify that FF(H) was read from the last even memory location. Write ØØ(H) into this location.

3.4 Check Bit Ripple Test

This test validates the ECC check bits as data bits. This is done to allow the user see if there are any problems with the ECC correction capabilities. The system can read in check bits to the memory but does not have the capability to see if this operation was successful. This test therefor "turns off" the error correction capability within the system and artificially forces the check bits to be specific values within memory. By doing this, the ECC hardware is validated to the users satisfaction.

(1) Write a 16-bit word oriented address into all memory locations. (e.g. ØØØØ(H) to location Ø, ØØØ1(H) to loc. 2, ØØØ3(H) to loc. 4, ..., (N-1)/2 to loc. (N-1), N/2 to loc. N)

(2) Read each word in memory and verify that it is the correct value.

(3) Starting with the highest memory location and working in reverse order, write the complement of the numerically corresponding word oriented address into all memory locations. (e.g. (FFFF(H)-N/2) to loc. N, (FFFF(H)-(N-1)/2) to loc. (N-1), ..., FFFE(H) to loc. 2, FFFFH to loc. Ø)

(4) Read each word in memory and verify that it is the correct value.

(5) Write ØØ(H) into all memory locations.

(6) Verify, in ascending address order, that ØØ(H) was read from each memory location. Once verified, write FF(H) into each memory location in ascending order.

(7) Verify, in descending address order, that FF(H) was read from each memory location. Once verified, write ØØ(H) into each memory location in descending order.

4.Ø ECC Tests

These tests are designed to validate all of the ECC hardware. The Check Bit Test,3.4, tested only the check bit portion of the ECC whereas these tests will check out the entire ECC system. The tests assume that the RAM has been fully tested and that some arbitrary RAM locations may be required to aid in the checkout of this area.

NOTE: These tests are not implemented in this revision.

4.1 Sliding Check Bit Test

This test slides forced data check bits, both 1's and Ø's, across the check bit data path, to locate any bits that may be faulty, e.g.open or crossed over.

4.2 Single Bit Error Test

This test artificially forces a single bit error in each of the 22 data/check bits. Once accomplished, it verifies the correct response of the ECC hardware.

4.4 Double Bit Error Test

This test artificially forces a double bit error in each of the 22 data/check bits. Once accomplished, it verifies the correct response of the ECC hardware. (Note: The number of patterns is 22!/2!2Ø! or 231.)

4.6 Syndrome Word Sliding Bit Test

This test will force bit errors to slide 1's and Ø's across the ECC syndrome word.

5.Ø Memory Manager Tests

These tests check out the memory manager hardware. This hardware is an I/O link from the memory to the central or master processor. The bus arbitration and memory lockout functions are both a function of hardware and software that must be throughly tested to insure the integity of the system, and its' means of communicating/processing information to the outside world, is real. Through testing of this hardware is essential to the field reliability of the system and that a minimum of down time is experienced by the user. As one exercises the memory management hardware, one will not only excerise the hardware but will check the software for soft errors, that might have inadvertantly appeared in the memory itself(e.g. disk containimination, demagnatizing the disk by accident, etc.) or other anomiallies (input power line surges, unscheduled maintance/"self passed training" done by unqualified personnel, etc.) that are normal "gliches" or problems that plauge the computer industry.

5.1 Page Base Address Ripple Test

This test will ripple the base addresses, of all virtual pages, to insure that they are able to span all physical pages.

5.2 Device Access Violation Test

This test will insure that the access restriction bits (system mode, user mode, DMA, arithmetic processor, code, data, stack, extra, stack bound) function correctly on all pages.

5.3 User Mode Program Operation Exception Test

This test insures that all 8Ø86 operations that are illegal in user mode (I/O operations, clearing the interrupt flag, or processor halt) generate the appropriate hardware interrupts.

```
 ALTOS COMPUTER SYSTEMS - 8600
     Monitor Version 1.02
     Release Date 12/24/81
Press any key to interrupt boot


CP/M-86 Version 1.0

Segment Address = 0080
    Last Offset = 2F7F


System Generated Sept 24, 1981
A>mem86



*** ACS8600 Memory Verification  v1.0 ***

  Processor Modes Tested.
  Program Memory = 00 (00400) to 06 (068C0)
  Test Memory = 07 (07000) to 7F (7FFFF)

Main Menu
      (A)  Continuous Comprehensive Test
      (B)  Short Comprehensive Test
      (C)  ECC Menu
      (D)  RAM Menu
      (E)  MGR Menu
      (F)  Relocate Program
      (G)  Error Handling Menu
      (H)  List Device Menu
      (I)  Message Severity Menu
      (J)  Exit to Diagnostic Disk
Enter: I


Message Severity Menu
      (A)  print errors
      (B)  print errors, warnings
      (C)  print errors, warnings, information
      (D)  print errors, warnings, information, traces
Enter: C
```

Appendix A -- Sample Session

```
Main Menu
      (A)  Continuous Comprehensive Test
      (B)  Short Comprehensive Test
      (C)  ECC Menu
      (D)  RAM Menu
      (E)  MGR Menu
      (F)  Relocate Program
      (G)  Error Handling Menu
      (H)  List Device Menu
      (I)  Message Severity Menu
      (J)  Exit to Diagnostic Disk
Enter: G


Error Handling Menu
      (A)  Continue  (ignore all errors)
      (B)  Pause  (for operator selection when error occurs)
      (C)  Loop  (retry all errors)
Enter: B


Main Menu
      (A)  Continuous Comprehensive Test
      (B)  Short Comprehensive Test
      (C)  ECC Menu
      (D)  RAM Menu
      (E)  MGR Menu
      (F)  Relocate Program
      (G)  Error Handling Menu
      (H)  List Device Menu
      (I)  Message Severity Menu
      (J)  Exit to Diagnostic Disk
Enter: A

Comprehensive test, pass #1
RAM test
RAM quick test
RAM byte ripple test
RAM word ripple test
RAM odd word ripple test
MGR test
MGR page mapping test
MGR access control test
*** ERROR -- MGR CS seg read violation on page Ø7 --
    align=ØØ, loc=Ø7ØØØ, b/w=1

Error Pause:
      (A)  Continue  (ignore this error)
      (B)  Loop  (retry this error)
      (C)  Error Handling Menu
      (D)  Message Severity Menu
```

```
Enter: C
Error Handling Menu
      (A)  Continue  (ignore all errors)
      (B)  Pause  (for operator selection when error occurs)
      (C)  Loop  (retry all errors)
Enter: A


Error Pause:
      (A)  Continue  (ignore this error)
      (B)  Loop  (retry this error)
      (C)  Error Handling Menu
      (D)  Message Severity Menu
Enter: B

*** ERROR -- MGR CS seg read violation on page Ø7 --
     align=ØØ, loc=Ø7ØØØ, b/w=1
*** ERROR -- MGR CS seg write violation on page Ø7 --
     align=ØØ,loc=Ø7ØØØ, b/w=1
*** ERROR -- MGR CS seg read violation on page Ø7 --
     align=Ø1, loc=Ø7FFE, b/w=1
*** ERROR -- MGR CS seg write violation on page Ø7 --
     align=Ø1,loc=Ø7FFE, b/w=1
*** ERROR -- MGR CS seg read violation on page Ø7 --
     align=Ø2, loc=Ø7ØØØ, b/w=Ø
*** ERROR -- MGR CS seg write violation on page Ø7 --
     align=Ø2,loc=Ø7ØØØ, b/w=Ø
*** ERROR -- MGR CS seg read violation on page Ø7 --
     align=Ø3, loc=Ø7FFF, b/w=Ø
*** ERROR -- MGR CS seg write violation on page Ø7 --
     align=Ø3,loc=Ø7FFF, b/w=Ø
*** ERROR -- MGR boundary write violation on page Ø7 --
     align=ØØ, loc=Ø7ØØØ, b/w=1
*** ERROR -- MGR boundary write violation on page Ø7 --
     align=Ø2, loc=Ø7ØØØ, b/w=Ø
*** ERROR -- MGR boundary write violation on page Ø7 --
     align=Ø4, loc=Ø7Ø7E, b/w=1
*** ERROR -- MGR boundary write violation on page Ø7 --
     align=Ø5, loc=Ø7Ø7F, b/w=Ø
*** ERROR -- MGR boundary write violation on page Ø7 --
     align=Ø6, loc=Ø7Ø7F, b/w=1
*** ERROR -- MGR CS seg read violation on page Ø8 --
align=ØØ, loc=Ø8ØØØ, b/w=1
*** ERROR -- MGR CS seg write violation on page Ø8 --
align=ØØ,loc=Ø8ØØØ, b/w=1
*** ERROR -- MGR CS seg read violation on page Ø8 --
align=Ø1, loc=Ø8FFE, b/w=1
*** ERROR -- MGR CS seg write violation on page Ø8 --
align=Ø1,loc=Ø8FFE, b/w=1
*** ERROR -- MGR CS seg read violation on page Ø8 --
align=Ø2, loc=Ø8ØØØ, b/w=Ø
```

```
*** ERROR -- MGR CS seg write violation on page Ø8 --
align=Ø2,loc=Ø8ØØØ, b/w=Ø
*** ERROR -- MGR CS seg read violation on page Ø8 --
align=Ø3, loc=Ø8FFF, b/w=Ø
*** ERROR -- MGR CS seg write violation on page Ø8 --
align=Ø3,loc=Ø8FFF, b/w=Ø
*** ERROR -- MGR CS seg read violation on page Ø8 --
     align=Ø4, loc=Ø7FFF, b/w=1
*** ERROR -- MGR CS seg write violation on page Ø8 --
     align=Ø4,loc=Ø7FFF, b/w=1
*** ERROR -- MGR CS seg read violation on page Ø8 --
     align=Ø5, loc=Ø8FFF, b/w=1
*** ERROR -- MGR CS seg write violation on page Ø8 --
     align=Ø5,loc=Ø8FFF, b/w=1
*** ERROR -- MGR boundary write violation on page Ø8 --
     align=ØØ,loc=Ø8ØØØ, b/w=1
*** ERROR -- MGR boundary write violation on page Ø8 --
     align=Ø2,loc=Ø8ØØØ, b/w=Ø
*** ERROR -- MGR boundary write violation on page Ø8 --
     align=Ø4,loc=Ø8Ø7E, b/w=1
*** ERROR -- MGR CS seg read violation on page Ø9 --
     align=Ø1, loc=Ø9FFE, b/w=1
*** ERROR -- MGR CS seg write violation on page Ø9 --
     align=Ø1,loc=Ø9FFE, b/w=1
*** ERROR -- MGR CS seg read violation on page Ø9 --
     align=Ø2, loc=Ø9ØØØ, b/w=Ø
*** ERROR -- MGR CS seg write violation on page Ø9 --
     align=Ø2,loc=Ø9ØØØ, b/w=Ø
*** ERROR -- MGR CS seg read violation on page Ø9 --
     align=Ø3, loc=Ø9FFF, b/w=Ø
*** ERROR -- MGR CS seg write violation on page Ø9 --
     align=Ø3, loc=Ø9FFF, b/w=Ø
*** ERROR -- MGR CS seg read violation on page Ø9 --
     align=Ø4, loc=Ø8FFF, b/w=1
***ERROR -- MGRCSsegwriteviolation on page Ø9 --
     align=Ø4,loc=Ø8FFF, b/w=1
*** ERROR -- MGR CS seg read violation on page Ø9 --
     align=Ø5, loc=Ø9FFF, b/w=1
*** ERROR -- MGR CS seg write violation on page Ø9 --
     align=Ø5, loc=Ø9FFF, b/w=1
*** ERROR -- MGR boundary write violation on page Ø9 --
     align=ØØ, loc=Ø9ØØØ, b/w=1
*** ERROR -- MGR boundary write violation on page Ø9 --
     align=Ø2, loc=Ø9ØØØ, b/w=Ø
*** ERROR -- MGR boundary write violation on page Ø9 --
     align=Ø4, loc=Ø9Ø7E, b/w=1
*** ERROR -- MGR boundary write violation on page Ø9 --
     align=Ø5, loc=Ø9Ø7F, b/w=Ø
*** ERROR -- MGR boundary write violation on page Ø9 --
     align=Ø6, loc=Ø9Ø7F, b/w=1
*** ERROR -- MGR CS seg read violation on page ØA --
     align=ØØ, loc=ØAØØØ, b/w=1
*** ERROR -- MGR CS seg write violation on page ØA --
     align=ØØ,loc=ØAØØØ, b/w=1
```

```
*** ERROR -- MGR CS seg read violation on page ØA --
      align=Ø1, loc=ØAFFE, b/w=1
*** ERROR -- MGR CS seg write violation on page ØA --
      align=Ø1, loc=ØAFFE, b/w=1
*** ERROR -- MGR CS seg read violation on page ØA --
      align=Ø2, loc=ØAØØØ, b/w=Ø
*** ERROR -- MGR CS seg write violation on page ØA --
      align=Ø2,loc=ØAØØØ, b/w=Ø
*** ERROR -- MGR CS seg read violation on page ØA --
      align=Ø3, loc=ØAFFF, b/w=Ø
*** ERROR -- MGR CS seg write violation on page ØA --
      align=Ø3, loc=ØAFFF, b/w=Ø
*** ERROR -- MGR CS seg read violation on page ØA --
      align=Ø4, loc=Ø9FFF, b/w=1
*** ERROR -- MGR CS seg write violation on page ØA --
      align=Ø4, loc=Ø9FFF, b/w=1
*** ERROR -- MGR CS seg read violation on page ØA --
      align=Ø5, loc=ØAFFF, b/w=1
*** ERROR -- MGR CS seg write violation on page ØA --
      align=Ø5, loc=ØAFFF, b/w=1
*** ERROR -- MGR boundary write violation on page ØA --
      align=ØØ, loc=ØAØØØ, b/w=1
*** ERROR -- MGR boundary write violation on page ØA --
      align=Ø2, loc=ØAØØØ, b/w=Ø
*** ERROR -- MGR boundary write violation on page ØA --
      align=Ø4, loc=ØAØ7E, b/w=1
*** ERROR -- MGR boundary write violation on page ØA --
      align=Ø5, loc=ØAØ7F, b/w=Ø
*** ERROR -- MGR boundary write violation on page ØA --
      align=Ø6, loc=ØAØ7F, b/w=1
*** ERROR -- MGR CS seg read violation on page ØB --
      align=ØØ, loc=ØBØØØ, b/w=1
*** ERROR -- MGR CS seg write violation on page ØB --
      align=ØØ,loc=ØBØØØ, b/w=1
*** ERROR -- MGR CS seg read violation on page ØB --
      align=Ø1, loc=ØBFFE, b/w=1
*** ERROR -- MGR CS seg write violation on page ØB --
      align=Ø1,loc=ØBFFE, b/w=1
*** ERROR -- MGR CS seg read violation on page ØB --
      align=Ø2, loc=ØBØØØ, b/w=Ø
*** ERROR -- MGR CS seg write violation on page ØB --
      align=Ø2,loc=ØBØØØ, b/w=Ø
*** ERROR -- MGR CS seg read violation on page ØB --
      align=Ø3, loc=ØBFFF, b/w=Ø
*** ERROR -- MGR CS seg write violation on page ØB --
      align=Ø3,loc=ØBFFF, b/w=Ø
*** ERROR -- MGR CS seg read violation on page ØB --
      align=Ø4, loc=ØAFFF, b/w=1
*** ERROR -- MGR CS seg write violation on page ØB --
      align=Ø4,loc=ØAFFF, b/w=1
*** ERROR -- MGR CS seg read violation on page ØB --
      align=Ø5, loc=ØBFFF, b/w=1
*** ERROR -- MGR CS seg write violation on page ØB --
      align=Ø5, loc=ØBFFF, b/w=1
```

```
*** ERROR -- MGR boundary write violation on page ØB --
     align=ØØ,loc=ØBØØØ, b/w=1
*** ERROR -- MGR boundary write violation on page ØB --
     align=Ø2,loc=ØBØØØ, b/w=Ø
*** ERROR -- MGR boundary write violation on page ØB --
     align=Ø4,loc=ØBØ7E, b/w=1
*** ERROR -- MGR boundary write violation on page ØB --
     align=Ø5,loc=ØBØ7F, b/w=Ø
*** ERROR -- MGR boundary write violation on page ØB --
     align=Ø6,loc=ØBØ7F, b/w=1
*** ERROR -- MGR CS seg read violation on page ØC --
     align=ØØ, loc=ØCØØØ, b/w=1
*** ERROR -- MGR CS seg write violation on page ØC --
     align=ØØ,loc=ØCØØØ, b/w=1
*** ERROR -- MGR CS seg read violation on page ØC --
     align=Ø1, loc=ØCFFE, b/w=1
*** ERROR -- MGR CS seg write violation on page ØC --
     align=Ø1,loc=ØCFFE, b/w=1
*** ERROR -- MGR CS seg read violation on page ØC --
     align=Ø2, loc=ØCØØØ, b/w=Ø
*** ERROR -- MGR CS seg write violation on page ØC --
     align=Ø2,loc=ØCØØØ, b/w=Ø
*** ERROR -- MGR CS seg read violation on page ØC --
     align=Ø3, loc=ØCFFF, b/w=Ø
*** ERROR -- MGR CS seg write violation on page ØC --
     align=Ø3, loc=ØCFFF, b/w=Ø
*** ERROR -- MGR CS seg read violation on page ØC --
     align=Ø4, loc=ØBFFF, b/w=1
*** ERROR -- MGR CS seg write violation on page ØC --
     align=Ø4, loc=ØBFFF, b/w=1
*** ERROR -- MGR CS seg read violation on page ØC --
     align=Ø5, loc=ØCFFF, b/w=1
*** ERROR -- MGR CS seg write violation on page ØC --
     align=Ø5, loc=ØCFFF, b/w=1
*** ERROR -- MGR boundary write violation on page ØC --
     align=ØØ, loc=ØCØØØ, b/w=1
*** ERROR -- MGR boundary write violation on page ØC --
     align=Ø2, loc=ØCØØØ, b/w=Ø
*** ERROR -- MGR boundary write violation on page ØC --
     align=Ø4, loc=ØCØ7E, b/w=1
*** ERROR -- MGR boundary write violation on page ØC --
     align=Ø5, loc=ØCØ7F, b/w=Ø
*** ERROR -- MGR boundary write violation on page ØC --
     align=Ø6, loc=ØCØ7F, b/w=1
*** ERROR -- MGR CS seg read violation on page ØD --
     align=ØØ, loc=ØDØØØ, b/w=1
*** ERROR -- MGR CS seg write violation on page ØD --
     align=ØØ, loc=ØDØØØ, b/w=1
*** ERROR -- MGR CS seg read violation on page ØD --
     align=Ø1, loc=ØDFFE, b/w=1
*** ERROR -- MGR CS seg write violation on page ØD --
     align=Ø1,loc=ØDFFE, b/w=1
*** ERROR -- MGR CS seg read violation on page ØD --
     align=Ø2, loc=ØDØØØ, b/w=Ø
```

```
*** ERROR -- MGR CS seg write violation on page ØD --
     align=Ø2,loc=ØDØØØ, b/w=Ø
*** ERROR -- MGR CS seg read violation on page ØD --
     align=Ø3, loc=ØDFFF, b/w=Ø
*** ERROR -- MGR CS seg write violation on page ØD --
     align=Ø3,loc=ØDFFF, b/w=Ø
*** ERROR -- MGR CS seg read violation on page ØD --
     align=Ø4, loc=ØCFFF, b/w=1
*** ERROR -- MGR CS seg write violation on page ØD --
     align=Ø4,loc=ØCFFF, b/w=1
*** ERROR -- MGR CS seg read violation on page ØD --
     align=Ø5, loc=ØDFFF, b/w=1
*** ERROR -- MGR CS seg write violation on page ØD --
     align=Ø5,loc=ØDFFF, b/w=1
*** ERROR -- MGR boundary write violation on page ØD --
     align=ØØ,loc=ØDØØØ, b/w=1
*** ERROR -- MGR boundary write violation on page ØD --
     align=Ø2,loc=ØDØØØ, b/w=Ø
*** ERROR -- MGR boundary write violation on page ØD --
     align=Ø4,loc=ØDØ7E, b/w=1
*** ERROR -- MGR boundary write violation on page ØD --
     align=Ø5,loc=ØDØ7F, b/w=Ø
*** ERROR -- MGR boundary write violation on page ØD --
     align=Ø6,loc=ØDØ7F, b/w=1
*** ERROR -- MGR CS seg read violation on page ØE --
     align=ØØ,loc=ØEØØØ, b/w=1
*** ERROR -- MGR CS seg write violation on page ØE --
     align=ØØ,loc=ØEØØØ, b/w=1
*** ERROR -- MGR CS seg read violation on page ØE --
     align=Ø1,loc=ØEFFE, b/w=1
*** ERROR -- MGR CS seg write violation on page ØE --
     align=Ø1,loc=ØEFFE, b/w=1
*** ERROR -- MGR CS seg read violation on page ØE --
     align=Ø2, loc=ØEØØØ, b/w=Ø
*** ERROR -- MGR CS seg write violation on page ØE --
     align=Ø2,loc=ØEØØØ, b/w=Ø
*** ERROR -- MGR CS seg read violation on page ØE --
     align=Ø3, loc=ØEFFF, b/w=Ø
*** ERROR -- MGR CS seg write violation on page ØE --
     align=Ø3,loc=ØEFFF, b/w=Ø
*** ERROR -- MGR CS seg read violation on page ØE --
     align=Ø4, loc=ØDFFF, b/w=1
*** ERROR -- MGR CS seg write violation on page ØE --
     align=Ø4,loc=ØDFFF, b/w=1
*** ERROR -- MGR CS seg read violation on page ØE --
     align=Ø5, loc=ØEFFF, b/w=1
*** ERROR -- MGR CS seg write violation on page ØE --
     align=Ø5, loc=ØEFFF, b/w=1
*** ERROR -- MGR boundary write violation on page ØE --
     align=ØØ,loc=ØEØØØ, b/w=1
*** ERROR -- MGR boundary write violation on page ØE --
     align=Ø2,loc=ØEØØØ, b/w=Ø
*** ERROR -- MGR boundary write violation on page ØE --
     align=Ø4,loc=ØEØ7E, b/w=1
```

```
*** ERROR -- MGR boundary write violation on page ØE --
     align=Ø5,loc=ØEØ7F, b/w=Ø
*** ERROR -- MGR boundary write violation on page ØE --
     align=Ø6,loc=ØEØ7F, b/w=1
*** ERROR -- MGR CS seg read violation on page ØF --
     align=ØØ,loc=ØFØØØ, b/w=1
*** ERROR -- MGR CS seg write violation on page ØF --
     align=ØØ,loc=ØFØØØ, b/w=1
*** ERROR -- MGR CS seg read violation on page ØF --
     align=Ø1,loc=ØFFFE, b/w=1
*** ERROR -- MGR CS seg write violation on page ØF --
     align=Ø1,loc=ØFFFE, b/w=1
*** ERROR -- MGR CS seg read violation on page ØF --
     align=Ø2, loc=ØFØØØ, b/w=Ø
*** ERROR -- MGR CS seg write violation on page ØF --
     align=Ø2,loc=ØFØØØ, b/w=Ø
*** ERROR -- MGR CS seg read violation on page ØF --
     align=Ø3, loc=ØFFFF, b/w=Ø
*** ERROR -- MGR CS seg write violation on page ØF --
     align=Ø3,loc=ØFFFF, b/w=Ø
*** ERROR -- MGR CS seg read violation on page ØF --
     align=Ø4, loc=ØEFFF, b/w=1
*** ERROR -- MGR CS seg write violation on page ØF --
     align=Ø4, loc=ØEFFF, b/w=1
*** ERROR -- MGR boundary write violation on page ØF --
     align=ØØ, loc=ØFØØØ, b/w=1
*** ERROR -- MGR boundary write violation on page ØF --
     align=Ø2, loc=ØFØØØ, b/w=Ø
*** ERROR -- MGR boundary write violation on page ØF --
     align=Ø4, loc=ØFØ7E, b/w=1
*** ERROR -- MGR boundary write violation on page ØF --
     align=Ø5, loc=ØFØ7F, b/w=Ø
*** ERROR -- MGR boundary write violation on page ØF --
     align=Ø6, loc=ØFØ7F, b/w=1
MGR user mode test
*** ERROR -- MGR user mode exception: interrupt disable
*** ERROR -- MGR user mode exception: user SVC call
Relocating program from ØØ4ØØ to 77ØØØ


*** ACS86ØØ Memory Verification  v1.Ø ***

  Processor Modes Tested.
  Program Memory = 77 (77ØØØ) to 7D (7D4CØ)
  Test Memory = Ø1 (Ø1ØØØ) to 76 (76FFF)
```

Appendix A -- Sample Session

```
Main Menu
      (A)  Continuous Comprehensive Test
      (B)  Short Comprehensive Test
      (C)  ECC Menu
      (D)  RAM Menu
      (E)  MGR Menu
      (F)  Relocate Program
      (G)  Error Handling Menu
      (H)  List Device Menu
      (I)  Message Severity Menu
      (J)  Exit to Diagnostic Disk
Enter: E


MGR Test Menu
      (A)  All (default)


      (B)  Page Base Ripple Test
      (C)  Device Access Violation Test
      (D)  User Mode Program Operation Exception Test
Enter: D

MGR user mode test, pass #1
*** ERROR -- MGR user mode exception: interrupt disable
*** ERROR -- MGR user mode exception: user SVC call
MGR user mode test, pass #2
*** ERROR -- MGR user mode exception: interrupt disable
*** ERROR -- MGR user mode exception: user SVC call
MGR user mode test, pass #3
*** ERROR -- MGR user mode exception: interrupt disable
*** ERROR -- MGR user mode exception: user SVC call
MGR user mode test, pass #4
*** ERROR -- MGR user mode exception: interrupt disable
*** ERROR -- MGR user mode exception: user SVC call
MGR user mode test, pass #5
*** ERROR -- MGR user mode exception: interrupt disable
*** ERROR -- MGR user mode exception: user SVC call
MGR user mode test, pass #6
*** ERROR -- MGR user mode exception: interrupt disable
*** ERROR -- MGR user mode exception: user SVC call
MGR user mode test, pass #7
*** ERROR -- MGR user mode exception: interrupt disable
*** ERROR -- MGR user mode exception: user SVC call
MGR user mode test, pass #8
*** ERROR -- MGR user mode exception: interrupt disable
*** ERROR -- MGR user mode exception: user SVC call
MGR user mode test, pass #9
*** ERROR -- MGR user mode exception: interrupt disable
*** ERROR -- MGR user mode exception: user SVC call
MGR user mode test, pass #1Ø
*** ERROR -- MGR user mode exception: interrupt disable
*** ERROR -- MGR user mode exception: user SVC call
```

```
MGR user mode test, pass #11
*** ERROR -- MGR user mode exception: interrupt disable
*** ERROR -- MGR user mode exception: user SVC call
MGR user mode test, pass #12
*** ERROR -- MGR user mode exception: interrupt disable
*** ERROR -- MGR user mode exception: user SVC call
MGR user mode test, pass #13
*** ERROR -- MGR user mode exception: interrupt disable
*** ERROR -- MGR user mode exception: user SVC call
MGR user mode test, pass #14
*** ERROR -- MGR user mode exception: interrupt disable
*** ERROR -- MGR user mode exception: user SVC call
MGR user mode test, pass #15
*** ERROR -- MGR user mode exception: interrupt disable
*** ERROR -- MGR user mode exception: user SVC call
MGR user mode test, pass #16
*** ERROR -- MGR user mode exception: interrupt disable
*** ERROR -- MGR user mode exception: user SVC call
MGR user mode test, pass #17



*** ACS8600 Memory Verification  v1.0 ***

  Processor Modes Tested.
  Program Memory = 77 (77000) to 7D (7D4C0)
  Test Memory = 01 (01000) to 76 (76FFF)

Main Menu
      (A)  Continuous Comprehensive Test
      (B)  Short Comprehensive Test
      (C)  ECC Menu
      (D)  RAM Menu
      (E)  MGR Menu
      (F)  Relocate Program
      (G)  Error Handling Menu
      (H)  List Device Menu
      (I)  Message Severity Menu
      (J)  Exit to Diagnostic Disk
Enter: I


Message Severity Menu
      (A)  print errors
      (B)  print errors, warnings
      (C)  print errors, warnings, information
      (D)  print errors, warnings, information, traces
Enter: D
```

Appendix A -- Sample Session

```
Main Menu
      (A)  Continuous Comprehensive Test
      (B)  Short Comprehensive Test
      (C)  ECC Menu
      (D)  RAM Menu
      (E)  MGR Menu
      (F)  Relocate Program
      (G)  Error Handling Menu
      (H)  List Device Menu
      (I)  Message Severity Menu
      (J)  Exit to Diagnostic Disk
Enter: G


Error Handling Menu
      (A)  Continue  (ignore all errors)
      (B)  Pause  (for operator selection when error occurs)
      (C)  Loop  (retry all errors)
Enter: B


Main Menu
      (A)  Continuous Comprehensive Test
      (B)  Short Comprehensive Test
      (C)  ECC Menu
      (D)  RAM Menu
      (E)  MGR Menu
      (F)  Relocate Program
      (G)  Error Handling Menu
      (H)  List Device Menu
      (I)  Message Severity Menu
      (J)  Exit to Diagnostic Disk
Enter: E


MGR Test Menu
      (A)  All (default)
      (B)  Page Base Ripple Test
      (C)  Device Access Violation Test
      (D)  User Mode Program Operation Exception Test
Enter: D

MGR user mode test, pass #1
trace: user interrupt disable test
*** ERROR -- MGR user mode exception: interrupt disable
TRAP Vector Dump: (count=1)
  seq=000001  int=02  err=0000  synd=00  addr=00018  cs:ip=7700:2AD2

Error Pause:
      (A)  Continue  (ignore this error)
      (B)  Loop  (retry this error)
      (C)  Error Handling Menu
      (D)  Message Severity Menu
Enter:
```