

SOFTWARE REFERENCE MANUAL

Digital Computer System

Model H8



Copyright © 1977
Heath Company
All Rights Reserved

HEATH COMPANY
BENTON HARBOR, MICHIGAN 49022

595-2048-01
Printed in the United
States of America



Contents

NOTE: An individual Table of Contents is included at the beginning of each of the following sections.

Introduction	Page 0-3
Panel Monitor (PAM-8)	Page 1-2
Console Debugger (BUG-8)	Page 2-2
Heath Text Editor (TED-8)	Page 3-2
Heath Assembly Language (HASL-8)	Page 4-2
Benton Harbor BASIC	Page 5-2

SPECIAL DISCLAIMER

Heath cannot provide consultation on user-developed programs or modified versions of Heath Software products.

These software products were developed for the Heath Company by the Wintek Corporation. Software copyrights reside with Wintek Corporation.

INTRODUCTION



TABLE OF CONTENTS

PANEL MONITOR (PAM-8)	0-7
CONSOLE DEBUGGER (BUG-8)	0-8
HEATH TEXT EDITOR (TED-8)	0-8
HEATH ASSEMBLY LANGUAGE (HASL-8)	0-10
BENTON HARBOR BASIC	0-11
TAPE FILES	0-12
System Record Structure.....	0-13
Label Record Format	0-14
System Data Formats	0-14
Reading the Displays	0-15
USING THE MAGNETIC TAPE SYSTEM	0-18
Recorder Operating Hints	0-18
USING THE PAPER TAPE SYSTEM	0-18
Paper Tape Operating Hints	0-19
PRODUCT INSTALLATION	0-19
Creating a Configured Tape	0-19
Loading From a Configured Tape.....	0-21
Copying an Existing Memory Tape	0-22
Installing a Pitch	0-22
Using an ASR Console	0-23
Using a 110 Baud Console Terminal	0-23
Console Interface.....	0-23
Reporting Software Problems	0-25
APPENDIX A (ASR Patches)	0-26
APPENDIX B (Console Driver Listing)	0-36
APPENDIX C (I/O and Memory Maps)	0-50
APPENDIX D (ASCII Characters)	0-52
APPENDIX E (Decimal to Octal Tables)	0-56
APPENDIX F (Memory Table, Offset Octal & Decimal Boundaries)	0-58
INDEX	0-59



This Software Reference Manual includes all the information you will need to be thoroughly familiar with the software products supplied with your H8 Computer. These software products are: the front Panel Monitor, PAM-8; the Console Debugger routine, BUG-8; the Heath Test Editor, TED-8; the Heath Assembly Language, HASL-8; and Benton Harbor BASIC, Heath Company's version of Dartmouth BASIC. Extended Benton Harbor BASIC, which is available as an optional accessory and includes such additional features as string manipulation, is also included in this Manual.

This book is intended as a reference manual, and, as such, it is as complete as possible. Examples are included to help you understand exactly how the Heath software products carry out their instructions; but they are not designed to teach you programming. If you have never used a text editor and assembler, for example, we recommend that you obtain some instruction from other sources, such as the "Heathkit Continuing Education" courses, prior to reading this material. If you have used editors and assemblers, this Manual will tell you about the special features in the Heath Text Editor (TED-8) and the Heath Assembly Language (HASL-8).

This introduction describes each product briefly and covers those aspects of the packages that are common to all. A separate section then follows for each software product. Each section provides detailed reference information and is followed by one or more Appendices for that product. Be sure to read all of this introductory section so you have a good overview of all of the products.

Heath software products feature a high degree of commonality in many of the modules which make up the individual products. For example, all software products which use the console terminal employ a software module called the Console Terminal Driver. This common usage of the console terminal driver permits you to move easily from one software product to the other, as the operating features are similar. Likewise, all tape handling is carried out through a common tape handling package, and once these features are understood, they are applicable to all products.



Heath software is supplied in three forms: cassette magnetic tape, paper tape, and read-only memory (ROM). The Panel Monitor (PAM-8) is supplied in a ROM (programs supplied in ROM cannot be modified by the user). The Console Debugger (BUG-8), the Heath Text Editor (TED-8), the Heath Assembly Language (HASL-8), and BASIC are supplied with the H8 in cassette form. They are optionally available in paper tape form. The cassettes and the paper tapes are compatible with the required error checking and synchronizing characters used by the front panel monitor system.

A printed copy of the panel monitor source listing is provided to aid you in using PAM-8. The Console Driver Listing and the partial listing (including entry points) of the BASIC floating point package and other BASIC utility packages are also included. All other programs are supplied in binary object forms and listings are **not** available.



PANEL MONITOR (PAM-8)

The ROM Panel Monitor, which is permanently located in the lower 1024 bytes of memory, permits you to load, execute, and debug programs written in 8080 machine language. The Heath Panel Monitor also makes use of the first 64 locations of random access memory. The H8 front panel is used as an I/O device, and it is assigned port numbers 360 and 361. With the Heath Panel Monitor, you can:

1. Examine the contents of a memory location.
2. Change the contents of a memory location (enter a new program, for example, or modify an old program).
3. Examine the contents of any of the 8080 registers.
4. Change the contents of any of the 8080 registers.
5. Start or stop the execution of a user-written program.
6. Execute a user program, a single instruction at a time.
7. Dump a program onto either magnetic or paper tape, with error detection codes and synchronization data.
8. Load a program from paper or magnetic tape into the desired memory locations.
9. Breakpoint a user program.
10. Reinitialize to a power up status.

The Heath Panel Monitor also offers the following features:

1. The user may automatically increment or decrement memory addresses which are being examined or modified.
2. The user may automatically increment or decrement through the registers which are being examined or modified.
3. The user is provided with a visual indication of the current mode in which the panel monitor is operating.
4. The user is provided with audio feedback upon valid and invalid command and data entry.
5. The H8 front panel utilizes an octal display rather than the more difficult to read binary display.
6. The front panel key switches and display are available for your programs.
7. The front display is operated on a continuously updated basis and, therefore, is active even during the execution of a user program. This feature permits the user to monitor either registers or memory location while his program is operating.

PAM-8 provides the fundamental tape routines by which the user loads all other programs, including the Heath supplied software and user-written software into the computer.



CONSOLE DEBUGGER (BUG-8)

BUG-8 allows you to perform very sophisticated operations from a console terminal with a full active keyboard and display. BUG-8 resides in H8 memory, using approximately 3,000 bytes of storage. You can use BUG-8 to write, load, execute, and debug machine language programs in the H8 computer in octal, decimal, or ASCII format. This package also has many of the features included in PAM-8.

With the Heath Console Debugger, you can:

1. Examine the contents of memory locations.
2. Alter the contents of memory locations.
3. Examine the contents of the CPU registers.
4. Alter the contents of the CPU registers.
5. Start program execution.
6. Execute a program in a single step form.
7. Set break points with multiple hit capability.
8. Clear break points.
9. Load programs from magnetic tape or paper tape.
10. Dump programs onto magnetic tape or paper tape.

BUG-8 is an advanced monitor, permitting you to prepare extensive software in machine code format that can be readily debugged and then recorded on a mass storage unit for future use.

HEATH TEXT EDITOR (TED-8)

The Heath TED-8 Text Editor is a general purpose, line-oriented text editor that is used primarily to prepare source code that can be assembled by the Heath Assembly Language (HASL-8). But while this is its primary purpose, it is also useful for such things as letter writing, preparation of club newspapers, and manuscript editing.

This software product requires an H8 system with 8192 bytes of memory, an ASCII keyboard for text entry, and an ASCII display for text display. If large files are to be used or files are to be saved, a separate input/output tape unit is recommended.



With the Heath Text Editor, you can:

1. Read text from a pre-existing text file.
2. Create text for a new file.
3. Output text to a named tape file.
4. Insert new text after a given line.
5. Search the text for a given character string.
6. Delete a given line or lines.
7. Print a particular line or lines.
8. Replace a given line.
9. Edit a given string; that is, replace a particular string with another string.

All the above functions are supported by a number of special features, some of which are only available on the Heath Text Editor. Some of these features are:

1. A wide scope of range expressions, including:
 - A. First line.
 - B. Last line.
 - C. Single line.
 - D. Line to line.
2. Count and string versions of range expressions, which permit you to edit lines, plus or minus a certain number of lines from a given line, or to edit all lines containing a certain string.
3. You have the option of selecting one of three optional modes. Optional mode A prints the line after operating on it, optional mode B prints the line before operating on it, and optional mode BA prints the line before and after operating on it.
4. The use of a Qualifier String (Qualifier Strings permit operating only on the lines containing designated strings).
5. Tab. This command lets you set tab stops for entering text. The editor is constructed so that tabs do not occupy extensive user storage.
6. A Use statement, which provides a line count and memory usage information.



7. File Labeling Procedures to create new file names in either the input or output mode.

Under the H8 text editor, source code is prepared for the Heath Assembly Language (HASL-8). Once the source code has been prepared, it is written to a cassette tape or paper tape output file. Once this has been done, the user proceeds to the assembler.

HEATH ASSEMBLY LANGUAGE (HASL-8)

Heath Assembly Language runs on a Heath H8 Computer using about 8192 bytes of memory. This program assembles source code and produces object code. HASL-8 utilizes all the standard 8080 mnemonics, extended mnemonics, and numerous psuedo instructions.

Some of the special features of HASL-8 are that it:

1. Recognizes five operators: plus, minus, *, /, unary-.
2. Recognizes four token operand expressions:
 - A. Integers.
 - B. Symbols.
 - C. Character strings.
 - D. The origin symbol.

HASL-8 is a two pass assembler. Before the user starts assembly, it asks if a binary output is to be generated. On the second pass, it produces the binary if directed to do so, as well as the appropriate listing. The binary object code may be placed on a specified output device or may be placed directly in memory.

HASL-8 features the same terminal controls as do other Heath programs, including a suspend output mode and a discard output mode.



BENTON HARBOR BASIC

BENTON HARBOR BASIC is a modified version of Dartmouth BASIC, an easy-to-learn-and-use conversational language.

The BENTON HARBOR BASIC system is interpretive. That is, it executes each statement as it comes to it. BENTON HARBOR BASIC utilizes an H8 computer with 8K of memory, and appropriate terminal and paper tape or magnetic tape handling capability.

Extended BENTON HARBOR BASIC requires 12K of memory and offers strings. Some of the features of BENTON HARBOR BASIC are:

1. Three different data types:
 - A. Numeric data, which has over six digits of accuracy and lies in the range of 10^{-39} to 10^{+38} . Numeric data may be either fixed or floating point.
 - B. Strings, which can be from 0 to 255 characters.
 - C. Boolean values, which permit logical operations.
2. Multidimensioned variables.
3. BASIC supports fifteen operators, which are:
 - A. -(unary) NOT
 - B. \uparrow Exponentiation
 - C. * /
 - D. + -
 - E. <, <=, =, <>, >=, and >
 - F. OR
 - G. AND
4. Free Format Programs.
5. Multiple statements per line.
6. Enhanced expression and conditional statement facilities.



BASIC features both command and program modes, where statements may be executed immediately after the line is written or numbered lines may be used so the program will not be executed until a RUN statement is executed.

BASIC also features command completion. In the command mode, BASIC checks inputted characters and, as soon as there are sufficient characters to establish a unique command, the command is completed. This feature saves considerable typing time and reduces errors.

NOTE: In order to fully use the Heath Software package, you must not only review the special features of Heath programs, but you must also know how to use monitors, debuggers, Text Editors, Assemblers, and BASIC. Once you have learned to use such programs, this Software Reference Manual will be an invaluable quick reference on how to carry out specific functions within the H8 software packages.

TAPE FILES

This section describes the tape format used in the Heath H8 Computer System. Tape formats are identical, regardless of the media used. The following terms are used to define the Heath H8 Tape format.

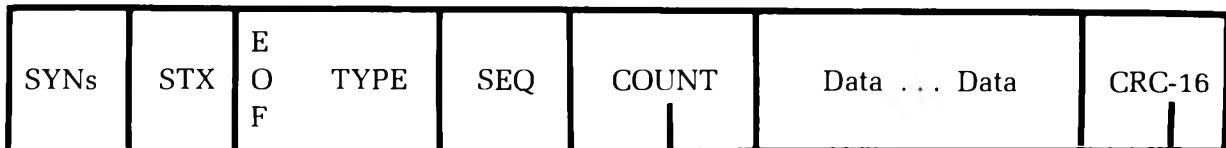
FILE A logically complete set of data. For example, a memory dump causes the FILE to be written on the tape. Although several files may be written onto one tape, the files are each totally independant of any other information written on that tape. A file consists of one or more records.

RECORD A record is a discrete block of data written to the tape transport. Each record must be read all at one time. It is not possible to read part of the record, pause, and then read the rest. Each record contains a CRC-16 Check. Each file has a first and last record. They may be the same record in a one-record file. The records in the file are numbered so a missing record can be detected.



System Record Structure

As discussed on Page 0-12, all H8 files consist of one or more records. All of the records have the same format.



- SYN From 20 to 40 ASCII Synchronizing Idle (026) characters.
- STX An ASCII STX character. This character, preceded by at least 10 SYN characters, indicates the start of a record. The SYN characters and the STX character are not included in the CRC. Note that a gap may be required between records to allow the tape transport to start and stop.
- EOF End of file. This flag is the high-order bit in the 'TYPE' byte. If set, it indicates that this is the last record in the file. The record is otherwise normal, and may contain data.
- TYPE This 7-bit field (the 8th bit is 'EOF') indicates the type of the record. All records in a file have the same type. The data field's format is type dependent. See below for a description of file types.
- SEQ This field is an 8-bit sequence counter, used to detect missing records. If a label record is present in the file, it is record #0. The first data record is #1. If the file contains no label record, the first record is record #1. Note that the record following record #255 is record #0, but is not a label record.
- COUNT This two-byte field contains a count of the number of bytes in the Data field. The high-order byte of the count appears first. Note that the count may be zero, indicating that there is no data field.
- DATA This field contains the data. Its format is dependent upon the record type. Its length is set in 'count'.



CRC-16 This is a polynomial remainder check, computed byte-wise upon the entire record (starting with the EOF/TYPE byte) from $(X + 1) * (X^{15} + X + 1)$. This checksum provides nearly flawless error detection.

<u>ERROR</u>	<u>DETECTION RATE</u>
Single bit error	100%
Double bit errors	100%
An odd number of bits in error	100%
An error burst < 17 bits long	100%
An error burst ≥ 17 bits	99.997%

Label Record Format

Some file types require a label record to be present, and some require that no label record be present. A label record is detected by its record number of 0. Except for the contents of the data field, a label record has the same format as the other records in the file. The data field consists of a string of 7-bit ASCII characters which comprise the file's label. The 8th bit should be 0 for all characters.

System Data Formats

The following section describes the data formats associated with the various file types. There are currently three file types:

1. Memory Image.
2. BASIC programs.
3. Compressed Text.

MEMORY IMAGE (Type = 001)

The file type 'memory image' is used when you dump or load programs from H8 memory. This file type has no label record, the first record in the file is #1. The file may consist of one or more records. The format of the data field is:

ENTRY	ADDR	Program bytes
-------	------	---------------

Where ENTRY = the program's entry point address, and ADDR = the address to start loading this group of program bytes. If there are multiple records in this data file, the 'entry' portion of each record should be identical.



NOTE: The COUNT field in the record header does not include the 4 bytes for ENTRY and ADDR. Thus, an empty record of this type has a zero COUNT field, but still contains the ENTRY and ADDR in the data field. Note that the high-order byte comes first for the ENTRY and ADDR fields.

BASIC PROGRAM (Type = 002)

This file type is used by BASIC when you load and dump programs. The file always has a label record (#0), and always has only one data record, #1. The data field contains the BASIC program in a special internal format. This file type can not be processed by the text editor.

COMPRESSED TEXT (Type = 003)

This file type is used by TED-8 and HASL-8 for source statements. It always has a label record (record #0) and has one or more data records. The data field in each record should not exceed 512 bytes. Lines should not be split between records. Each line is compressed according to the following format:

1. All characters are 7-bit ASCII, with the parity bit zero.
2. The carriage return and line-feed characters are not used. The end of line is indicated by a 000 byte.
3. Strings of spaces are represented by the value $200_8 + N$, where 'N' is the number of spaces in the series. Thus, a single blank is encoded as 201_8 ; ten blanks are encoded as 212_8 .
4. The maximum line length is 127 characters.

Reading the Displays

When the H8 computer is reading or writing data on a tape transport, the front panel displays are continually displaying data about the tape operation. Data about tape operation is displayed into two areas:

ADDRESS LEDs — Display the number of bytes left in the record when the transport is reading or writing data. The Address LEDs do not display any information during the inter-record gap. The address LEDs display the actual address being loaded when a memory image load or dump is executed. During a memory image operation, the Data LEDs display the data being entered into or read from memory.

DATA LEDs — Display the type of data and the record number. The right-hand-most data LED displays the type of data being read or written. This information is displayed as:

<u>DISPLAY</u>	<u>DATA/TYPE</u>
1.	Memory Image
2.	A BASIC program.
3.	Compressed text.

The two left hand LEDs display the octal record count within the particular file. As noted earlier, a file may contain one or more records. When 17_8 records are exceeded, the record count in the two left hand data LEDs starts over at 00. When the last record is read, the extreme left hand data LED displays a two or a three if the record count is between 10_8 and 17_8 . Using this information, you can readily observe the type of data being handled by the H8. Figure 0-1 shows how these displays are used.

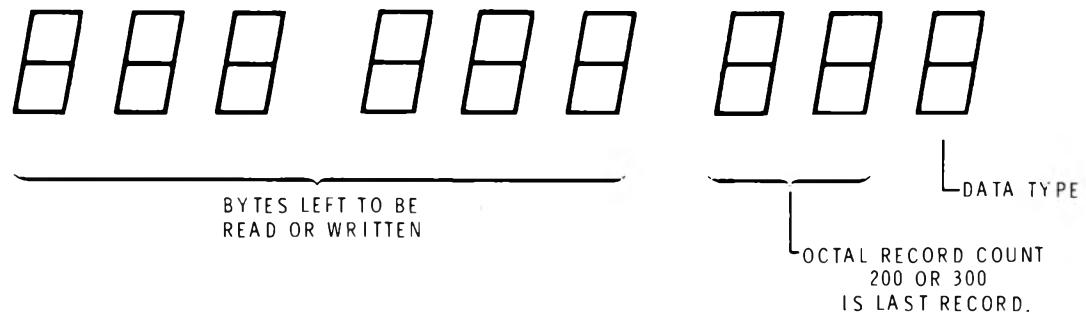


Figure 0-1



For example:

2 / 3

indicates the first and last data record of compressed text. NOTE: This could be the 1st or the 17₈th data record.

0 / /

indicates first data record of a memory image file.

0 / 0 3

indicates a label record of a compressed text file. Note, the label record is record number 0.

2 3 3

indicates the third and last record of a compressed text file.

3 / /

indicates the last record of a memory image file.



USING THE MAGNETIC TAPE SYSTEM

The Heath H8-5 serial card supports two different configurations of magnetic tape recorders. You may use a single tape recorder or a dual tape recorder. The most versatile system operation is obtained by using two tape recorders with independent control. However, you can achieve a perfectly workable and somewhat less expensive system with the single recorder. Dual recorders are preferable when you are assembling long programs, as it is necessary to read a few records of the source program and then assemble this source material, generating the appropriate binary output before reading additional source records. If you use a single recorder, you must change cassettes frequently.

Recorder Operating Hints

Observe the following guidelines carefully to get the maximum operating efficiency from your H8 Cassette Recorder system.

1. Use only Heath approved cassette tape recorders. Although there are a variety of very good tape recorders on the market, only those models tested and approved by the Heath Company will assure successful operation.
2. Use a high-quality cassette recording tape. Once again, the Heath approved tape is required to assure success.
3. Be sure the tape is off the leader before recording a file. Frequently cassettes have excessively long leaders. Therefore, the initial portion of a file placed on tape may lose the synchronizing characters, and be lost.
4. Keep the tape and the recorder clean. Dirty tapes tend to cause drop outs which cause tape errors.
5. Label your tapes. There is nothing more frustrating than a good program written onto a tape that is unlabeled and therefore not recoverable later.

USING THE PAPER TAPE SYSTEM

The H10 Paper Tape Reader/Punch is two independent devices. It is both a paper tape punch and a paper tape reader. When it is used with the H8-2 Parallel I/O Interface, operation of the H10 becomes identical to operation with dual cassette recorders.



Paper Tape Operating Hints

The following hints will help insure you of maximum effectiveness with your paper tape operating system.

1. Use oiled paper tape. Oiled paper tape helps keep the punch system operating smoothly and reduces punch wear.
2. Keep the punch well adjusted. After the first few hours of operation, carefully observe the punch mechanism to be sure it is still properly aligned.
3. Label all your tapes. Just like magnetic tapes, unlabeled paper tapes are virtually useless.
4. Keep the chad cleaned up. Excessive loose chad tends to cause reliability problems.

PRODUCT INSTALLATION

Use the following procedure as a guideline when you install your magnetic or paper tape operating system. Remember, a major part of your H8 System purchase is the software. The installation of the software should be treated with as much care and thought as the installation of the hardware products. Without the software, the hardware is of little or no value.

Creating a Configured Tape

The Heath H8 software is supplied on distribution tapes. These tapes enable you to configure the software for your own particular needs. Use the following procedure to create a configured tape from the software distribution tape.

- A. Load tape in reader.
- B. Ready tape transport.
- C. Press LOAD on H8 front panel.
- D. Wait for a single beep indicating successful load.
- E. Press GO on H8 front panel.*

*NOTE: Any software patches received with the product should be entered before you execute their step. Once entered, they become a permanent part of the configured tape. (See Page 0-22.)



G. Configure the software product as desired, answering each of the following questions. Prompt each question by typing its first character on the console terminal keyboard. Simply type a return or do not prompt the question if you wish to leave them as distributed. The questions are:

AUTO NEWLINE (Y/N) ?

A yes (Y) response to this question directs the product to generate a new line each time the print head (or cursor) moves out of the last column of the console terminal. This function is distributed preset to Y.

BKSP = 00008 /

The backspace character is normally a control H (00008 decimal). When used with the video terminal or other backspacing devices, the control H generates a true backspace. The backspace character may be changed to other ASCII printing or nonprinting characters (See Appendix D) such as a backslash, if a non-backspacing terminal is used. This new character will be considered a true backspace by the software.

CONSOLE LENGTH = 00080 /

The console length is initially set at 80 (decimal) characters, which is the normal width of a video terminal. This may be changed to other common values such as 132 characters for a wide printing terminal, or to 72 characters for a teleprinter.

NOTE: The maximum number of characters per line is a function of each software product. See the individual sections to determine the maximum permissible characters per line.

HIGH MEMORY = XXXXX /

When the software product is initially started, the limit of available high memory is determined. All products start at 040 100 (offset octal). If you wish not to use a certain portion of high memory, a new high memory limit (decimal count) should be typed in. If the upper memory limit is set too low, the new limit will be refused (the terminal bell will sound).

LOWER CASE (Y/N) ?

A Yes (Y) response to LOWER CASE configures the software product to input lower case letters and output lower case letters. An N (no) in response to the question configures the product to work with upper case only terminals. This function is distributed preset to N.



PAD = 4/

The pad characters (nulls) are inserted following a carriage return. The pad characters are sent at this time to allow the print head time to return to the left-hand margin. For video terminals, and most teleprinters, the number of pad characters may be changed to zero. If you do not know how many pad characters are required for your terminal, initially try zero. If you appear to be overtyping (or missing characters) at the beginning of lines, increase the pad count until the overtyping stops. You may enter up to a maximum of 9 pad characters.

RUB OUT = 00127/

The rub out character is set as 127 (decimal). If you desire to use a special rub out character, you may change it by entering in a new decimal number identifying a different ASCII character. See Appendix D.

SAVE?

A yes (Y) in response to this question directs the software product to generate a memory image of the configured product. This memory image of the configured product should be the tape you use regularly to load your program. This will avoid your having to configure the product on a regular basis. Before executing the save command, be sure the tape transport at the dump output is ready.

To use the product directly from the distribution tape, type the return key at any time rather than typing a key which prompts a question.

NOTE: It is very important that you immediately configure products as you will use them, and then place your original software distribution tape in an appropriate place for safe keeping. Use the above procedure any time you wish to configure the product.

Loading From a Configured Tape

Loading from a configured tape is a very simple procedure. It is the recommended way to normally load the software. The procedure is:

1. Load tape in the tape transport.
2. Ready the tape transport.
3. Press LOAD on the H8 front panel.
4. Wait for a single beep, indicating a successful load.
5. Press GO key on the H8 front panel.
6. The console terminal will respond with the product description and its prompt character. The product is now ready to use in a preconfigured form.



Copying an Existing Memory Tape

Use the following procedure to copy a memory image tape. Be sure to use this procedure. Memory image tapes should not be copied on an audio-to-audio basis. An audio-to-audio copy may not work. To copy a tape:

1. Load the source tape in the tape transport.
2. Ready the tape transport.
3. Press LOAD on the H8 front panel.
4. Wait for a single beep, indicating a successful load.
5. Load a blank tape into the dump tape transport.
6. Ready the dump tape transport.
7. Press DUMP on the H8 front panel.
8. Wait for a single beep, indicating a completed dump.
9. Repeat steps 7 and 8 to produce a second dump, creating a double copy.

The product is now copied and ready to be used. **IMPORTANT:** Make at least one double copy of the distribution tapes you received with the H8 as a protection against accidental tape damage. Once the tape is copied, if magnetic tape is being used, the read-only plugs should be knocked out of the back of the cassette.

Installing a Patch

To implement a patch supplied with the distribution software tapes or those in Appendix A, load the distribution tape following steps A - P in "Creating a Configured Tape" (Page 0-19). Alter the memory contents at the locations shown in the desired patch(s), inserting the new data given in the patch. Refer to Page 1-10 "Displaying and Altering Memory Locations" for the appropriate procedure to modify a memory location. For example, to use Option Patch #1 on BENTON HARBOR BASIC:

1. Load the distribution tape through step D.
2. Change the contents of location 041 010 to 316 and the contents of location 064 077 to 001. This completes step D.
3. Finish the configuration continuing with step E. BENTON HARBOR BASIC is now patched to 5.01.00.I and will supply two stop bits on each transmitted ASCII character.



Using an ASR Console

The following procedure allows you to use an ASR console as the main load/dump port, as well as the console terminal with an H8 system. An example of such an ASR (automatic send/receive) console would be the Teletype Corporation Model 33 Teleprinter. Perform the initial load by first setting the port interchange switch to the port interchange position on your H8-5 Serial I/O card. The tapes are then read in, in accordance with the procedure outlined under "Creating a Configured Tape." Once the tapes are read in, PAM-8 should be used to patch the software to the ASR console terminal configuration.

The appropriate patches for the ASR console terminal configurations of the software products are found in Appendix A of this section. Once the patches are accomplished, the GO key may be pressed and the normal configuration procedure takes place. Leave the Port Interchange switch in the Port Interchange position as long as the tape handler on the console terminal is used as the main load dump terminal.

Using a 110-Baud Console Terminal

If you use a 110-baud console terminal such as a teleprinter, one extra stop bit must be added to the ASCII characters sent by the H8. This change is made at configuration time and should be done any time you use a 110-baud console, regardless of whether or not you use the terminal as an ASR console (such as a teleprinter with a paper tape reader/punch) or simply as the console terminal. The patch is listed in Appendix A.

Console Interface

Appendix B contains a listing of the H8 Console Driver, a software module included in all H8 software packages which utilize a console terminal. A console driver is a general-purpose software package, providing you with such special characters as Control A, Control B, Control C, Control D, Control O, Control P, Control S, and Control Q.



The characters Control-A, -B, -C, and -D are available for use within the program. For example, most Heath programs use Control-C as a general purpose cancel. The other characters are permanently assigned in the console driver, and therefore in all Heath software products using the console driver. These characters are assigned as follows:

CONTROL-O

Control-O toggles the output discard flag. When the output discard flag is set, output to the console terminal is stopped, but program execution continues. Typing Control-O once sets the discard flag. Typing the Control-O again resets the output flag, permitting output to the console terminal to resume.

CONTROL-P

Control-P resets the output discard flag. Typing Control-P insures the output discard flag is not set. NOTE: Control-O toggles the flag, but Control-P only resets it.

CONTROL-S

Control-S sets the suspend output flag.

CONTROL-Q

Control-Q resets the suspend output flag.

The above control characters are not echoed to the console terminal when they are typed as is a normal character. NOTE: Many Heath Software products use Control-H, Control-I and Rubout. These characters are not used by the console driver but are passed directly through to the program for individual processing. They are also echoed to the console terminal.

CONSOLE DRIVER

The console driver also provides all capabilities for communicating with the console terminal and a tape transport at the load/ dump ports. If you, the user, develop any of your own software packages, we recommend that you incorporate this console driver rather than attempting to develop your own console driver.

The use of the control characters is explained on Page 3 of the console driver listing and in the individual software product reference sections.

The console driver also provides two front panel entry points. These are listed on Page 5 of the console driver listing. They are:

PROGRAM COUNTER*ENTRY TYPE

040 100	Program Reset entry point. All text buffers, etc., are effectively cleared and the product is restarted. This is known as a "cold" or "hard" start.
040 103	Program restart entry point. Product is restarted with text buffers, etc., intact. This is known as a "warm" or "soft" start.

*NOTE: Set the value of the program counter using the front panel monitor. Then press GO.

Reporting Software Problems

Every effort has been made to keep the Heath H8 Software products free of defects. Should you suspect that a Heath H8 system software product may be defective, review the following procedure before contacting Heath Company.

1. Attempt to reload all software to be sure it has not been damaged.
2. Reconfigure the product from the distribution tape in an attempt to duplicate the problem.
3. If the problem persists, document the apparent product problem and the software version code. NOTE: It is extremely important that you know your exact software version code when you contact the Heath Company about your software product.

For example:

TED-8 # 3.01.00 XXXX
Product name _____
Product number _____
Revision level _____
Patch level _____
Installed Options _____

NOTE: Heath Company can not consult on user developed programs or modified versions of the Heath Software products.



APPENDIX A

This appendix is a listing of the patches required to use an ASR console terminal as both the load/dump port and as the console terminal (option patch #2). A patch to convert the output to two-stop bits is also included (option patch #1).

BUG # 8
02.01.***.

OPTION PATCH #2

ASR CONSOLE

USE: This patch is used for systems which have both their console terminal and load/dump device attached to the same port, such as a Teletype model 33 ASR. For those systems, these patches are required. Note that the port interchange switch must be set to interchange.

NOTES: These patches remove the VERIFY command from BUG-8. It will no longer be valid. Also, CTRL-Q is no longer effective during tape operations. The proper procedure for aborting a load or dump is to return to the front panel monitor (via the front panel keyboard), set PC to the warm start address: 040103, and press 'GO'.

040107 370
040112 370
040115 371
040120 371
040123 371
040126 371
041004 072
041014 072
044046 072
046067 315 034 041 257
046231 251 046
046252 036 041 303 360 052
051164 311
051171 057 053
053047 055 052 315 036 041 303 305 050 315 366 051 303 036 041
053321 003
056360 001

TED - 8
03.01.xx.

OPTION PATCH #2

ASR CONSOLE

USE: This patch is used for systems which have both their console terminal and load/dump device attached to the same port, such as a Teletype model 33 ASR. For those systems, these patches are required. Note that the port interchange switch must be set to interchange.

NOTES: These patches remove the VERIFY command from TED-8. It will no longer be valid. CNTRL-C is no longer effective during tape operations. The proper procedure for aborting a load or a dump is to return to FAM-8 (via RTM on the FAM-8 keyboard), set Fc to the warm start address: 040103, and press 'GO'.

040107 370
040112 370
040115 371
040120 371
040123 371
040126 371
041004 072
041014 072
041170 072
042036 072
046267 146 047
046274 324
046324 315 036 041
047146 356 200 310 303 324 046
047156 357 001 200 052 226 057 311
053064 315 156 047
055074 311
055101 336 056
056326 363 055 315 036 041 303 044 054 315, 274 055 303 036 041
057143 003
064100 001



HASL = 8
04.01.***.

OPTION PATCH #2

ASR CONSOLE

USE: This patch is used for systems which have both their console terminal and load/dump device attached to the same port, such as a Teletype model 33 ASR. For those systems, these patches are required. Note that the port interchange switch must be set to interchange.

NOTES: None.

040107 370
040112 370
040115 371
040120 371
040123 371
040126 371
041004 072
041014 072
057022 064 061
061057 104 060 303 036 041 315 015 060
061067 303 036 041
072316 001

BENTON HARBOR BASIC
05.01.xx.

OPTION PATCH #2

ASK CONSOLE

USE: This patch is used for systems which have both their console terminal and load/dump device attached to the same port, such as a Teletype model 33 ASR. For those systems, these patches are required.
Note that the port interchange switch must be set to interchange.

NOTES: These patches remove the VERIFY command from B.H. BASIC. It will no longer be valid. Also, CTL-C is no longer effective during tape operations. The proper procedure for aborting a load or dump is to return to the front panel monitor (via the front panel keyboard), set Fc to the warm start address: 040103, and press 'GO'.

040107 370
040112 370
040115 371
040120 371
040123 371
040126 371
041004 072
041014 072
044356 303 252 046
046252 265 341 310 315 036 041 303 361
046262 044 302 160 067 341 303 340 070
057071 003
057171 036 041
067166 303 263 046
070034 311
070060 106 071
071076 017 003 315 036 041 303 020 067
071106 315 325 002 303 036 041
075202 001

EXTENDED BENTON HARBOR BASIC

10.01.XX.

OPTION PATCH #2

ASR CONSOLE

USE: This patch is used for systems which have both their console terminal and load/dump device attached to the same Port, such as a Teletype model 33 ASR. For those systems, these patches are required. Note that the Port interchange switch must be set to interchange.

NOTES: These patches remove the VERIFY command from BASIC. It will no longer be valid. Also, CTL-C is no longer effective during tape operations. The proper procedure for aborting a load or dump is to return to the front panel monitor (via the front panel keyboard), set Fc to the warm start address: 040103, and press 'GO'.

040107 370
040112 370
040115 371
040120 371
040123 371
040126 371
041004 072
041014 072
046307 303 316 050
050316 265 341 310 315 036 041 303 312
050326 048 302 151 100 341 303 011 100
064226 003
065110 036 041
100157 303 270 050
101116 311
101142 313 102
102303 024 102 315 036 041 303 011 100
102313 315 335 101 303 036 041
106173 001

BUG - 8
02.01.xx.

OPTION PATCH #1

2 STOP BITS

USE: This patch is inserted for systems which use a terminal device requiring 2 stop bits. This should not be used for devices which can run with only one stop bit.

NOTES: None.

041010 316
056357 001

TEII - 8
03.01.xx.

OPTION PATCH #1

2 STOP BITS

USE: This patch is inserted for systems which use a terminal device requiring 2 stop bits. This should not be used for devices which can run with only one stop bit.

NOTES: None.

041010 316
064077 001



HASL - B
04.01.xx.

OPTION PATCH #1

2 STOP BITS

USE: This patch is inserted for systems which use a terminal device requiring 2 stop bits. This should not be used for devices which can run with only one stop bit.

NOTES: None.

041010 316
072315 001

BENTON HARBOR BASIC
05.01.xx.

OPTION PATCH #1

2 STOP BITS

USE: This patch is inserted for systems which use a terminal device requiring 2 stop bits. This should not be used for devices which can run with only one stop bit.

NOTES: None.

041010 316
075201 001



EXTENDED BENTON HARBOR BASIC
10.01.xx.

OPTION PATCH #1

2 STOP BITS

USE: This patch is inserted for systems which use a terminal device requiring 2 stop bits. This should not be used for devices which can run with only one stop bit.

NOTES: None.

041010 316
106173 001



APPENDIX B

Console Driver Listing

HEATH H8 CONSOLE DRIVER
INTERRUPT-TIME CONSOLE DRIVER.

HEATH X8ASM V1.1...04/21/77
17:50:05 01-APR-77 PAGE 1

HEATHKIT

```
2 *** HEATH H8 SOFTWARE CONSOLE DRIVER.  
3 *  
4 * 4GL, 01/01/77, FOR *HEATH* COMPANY.  
5 *  
6 * COPYRIGHT 1977 BY HEATH COMPANY,  
7 * BENTON HARBOR, MI.  
8 *  
9 *  
10 *** THE FOLLOWING CONTAINS THE TEXT FOR THE HEATH H8  
11 * CONSOLE DRIVER. THESE EXACT ROUTINES ARE USED IN  
12 * ALL SOFTWARE PRODUCTS.  
13 *  
14 * ALL PROGRAMS WISHING TO COMMUNICATE WITH THE CONSOLE  
15 * SHOULD USE THESE ROUTINES.  
16 *  
17 040.100 18 ORG 40100A START OF USER RAM.  
19 *  
20 *  
21 ** ASSEMBLY CONSTANTS.  
22 *  
000.007 23 BELL EQU 0070 ASCII BELL  
040.037 24 .UIVEC EQU 040037A H8MTR USER INTERRUPT VECTORS  
111.111 25 START EQU 111111A DUMMY START LABEL  
222.222 26 RESTART EQU 222222A DUMMY RESTART LABEL  
333.333 27 CONFIG EQU 333333A DUMMY CONFIGURE LABEL  
28 *  
040.100 29 XTEXT U8251
```

HEATH HB CONSOLE DRIVER (ISSUE # XX.00.XX.)
8251 USART BIT DEFINITIONS.

HEATH X8ASM V1.0 02/18/77
12:15:16 01-APR-77 PAGE 2

32X ** 8251 USART BIT DEFINITIONS.

33X *

34X

35X ** MODE INSTRUCTION CONTROL BITS.

36X

000.100	37X UMI.1B	EQU	01000000B	1 STOP BIT
000.200	38X UMI.HB	EQU	10000000B	1 1/2 STOP BITS
000.300	39X UMI.2B	EQU	11000000B	2 STOP BITS
000.040	40X UMI.FE	EQU	00100000B	EVEN PARITY
000.020	41X UMI.FA	EQU	00010000B	USE PARITY
000.000	42X UMI.L5	EQU	00000000B	5 BIT CHARACTERS
000.004	43X UMI.L6	EQU	00000100B	6 BIT CHARACTERS
000.010	44X UMI.L7	EQU	00001000B	7 BIT CHARACTERS
000.014	45X UMI.L8	EQU	00001100B	8 BIT CHARACTERS
000.001	46X UMI.IX	EQU	00000001B	CLOCK X 1
000.002	47X UMI.16X	EQU	00000010B	CLOCK X 16
000.003	48X UMI.64X	EQU	00000011B	CLOCK X 64

49X

50X ** COMMAND INSTRUCTION BITS.

51X

000.100	52X UCI.IR	EQU	01000000B	INTERNAL RESET
000.040	53X UCI.RD	EQU	00100000B	READER-ON CONTROL FLAG
000.020	54X UCI.ER	EQU	00010000B	ERROR RESET
000.004	55X UCI.RE	EQU	00001000B	RECEIVE ENABLE
000.002	56X UCI.IE	EQU	00000100B	ENABLE INTERRUPTS FLAG
000.001	57X UCI.TE	EQU	00000010B	TRANSMIT ENABLE

58X

59X ** STATUS READ COMMAND BITS.

60X

000.040	61X USR.FE	EQU	00100000B	FRAMING ERROR
000.020	62X USR.OE	EQU	00010000B	OVERRUN ERROR
000.010	63X USR.FE	EQU	00001000B	FARITY ERROR
000.004	64X USR.TXE	EQU	00000100B	TRANSMITTER EMPTY
000.002	65X USR.RXF	EQU	00000010B	RECEIVER READY
000.001	66X USR.TXR	EQU	00000001B	TRANSMITTER READY
040.100	67? XTEXT CONSIX			



70X *** CONSL - SYSTEM CONSOLE AND I/O DRIVER.
71X *
72X * CONSL IS A GENERAL PURPOSE CONSOLE DRIVER. IT IS A STANDARD
73X * PRODUCT USED IN ALL HEATH H8 SOFTWARE (WHICH COMMUNICATES WITH
74X * A CONSOLE DEVICE).
75X *
76X * CONSL CONTAINS:
77X *
78X * 1) FORT ROUTINES. THESE ARE PLACED IN CONSOLE SO THEY HAVE THE SAME
79X * LOCATION IN ALL PRODUCTS. FORT ADDRESSES MAY BE CHANGED BY PATCHING
80X * THESE ROUTINES.
81X *
82X * 2) THE CONSOLE DRIVER PACKAGE. THIS PACKAGE CONSISTS OF THREE ROUTINES:
83X * \$RCHAR - READ A SINGLE CHARACTER
84X * \$WCHAR - WRITE A SINGLE CHARACTER
85X * \$PRSC - PRESET CONSOLE AND TAPE UARTS.
86X *
87X * THE CONSOLE PACKAGE PROVIDES SOPHISTICATED SUPPORT FOR
88X * ITS CALLERS:
89X * INTERRUPT PROCESSING FOR INPUT CHARACTERS
90X * 28 CHARACTER TYPE-AHEAD CAPABILITY
91X * SPECIAL CONTROL CHARACTER PROCESSING.
92X *
93X * NOTE THAT IF THE CONSOLE PACKAGE IS USED BY ANY RUNNING
94X * ROUTINE, ALL ROUTINES RUNNING WITH IT MUST USE IT ALSO.
95X * THIS IS BECAUSE *CONSL* WILL PROCESS INPUT CHARACTERS AT
96X * INTERRUPT TIME, BEATING OUT ANY TASK-TIME ROUTINE WHICH
97X * ATTEMPTS TO READ CHARACTERS.

99X ** SPECIAL CHARACTER PROCESSING.
100X *
101X * *CONSL* SUPPORTS 8 SPECIAL CHARACTERS:
102X *
103X * CTL-A USER DEFINED CONTROL FLAGS. THESE CAN BE CHECKED
104X * CTL-B AT TASK TIME, OR THE USER PROGRAM CAN SET UP AN
105X * CTL-C INTERRUPT VECTOR WHICH IS ENTERED, AT INTERRUPT
106X * CTL-D TIME, WHEN ANY OF THESE CHARACTERS ARE ENTERED.
107X *
108X * CTL-O TOGGLE DISCARDING OUTPUT CHARACTERS
109X * CTL-F RESUME OUTPUTTING CHARACTERS
110X * CTL-S SUSPEND OUTPUT
111X * CTL-Q RESUME OUTPUT

HEATH H8 CONSOLE DRIVER (ISSUE # XX.00.XX.)
SYSTEM I/O DRIVER

HEATH X8ASM V1.0 02/18/77
12:15:20 01-APR-77 PAGE 4

113X ** CONSOLE DRIVER ASSEMBLY CONSTANTS.

114X *

115X

116X

117X ** I/O PORT ADDRESSES.

118X

000.372	119X IP.CDF	EQU	3720	CONSOLE DATA IN PORT
000.372	120X DF.CDF	EQU	3720	CONSOLE DATA OUT PORT
000.373	121X IP.CIS	EQU	3730	CONSOLE INPUT STATUS IN PORT
000.373	122X DF.CTS	EQU	3730	CONSOLE INPUT STATUS OUT PORT
000.373	123X IP.COS	EQU	3730	CONSOLE OUTPUT STATUS IN PORT
000.373	124X DF.COS	EQU	3730	CONSOLE OUTPUT STATUS OUT PORT
	125X			
000.370	126X IP.TDP	EQU	3700	TAPE DATA IN PORT
000.370	127X DF.TDP	EQU	3700	TAPE DATA OUT PORT
000.371	128X IP.TSP	EQU	3710	TAPE STATUS IN PORT
000.371	129X DF.TSF	EQU	3710	TAPE STATUS OUT PORT

131X ** \$CSLCTL (CONSOLE CONTROL FLAG) BITS.

132X *

133X * THESE BITS ARE SET IN \$CSLCTL WHEN THE APPROPRIATE CONTROL

CHARACTERS ARE STRUCK.

135X *

136X * THESE CAN BE EXAMINED AT TASK-TIME, OR WHEN THE USERS CONTROL

CHARACTER ROUTINE IS ENTERED (AT INTERRUPT TIME)

137X *

138X

139X

000.001	140X CC.HLD	EQU	01	SUSPEND OUTPUT
000.002	141X CC.DMP	EQU	02	DISCARD OUTPUT
000.010	142X CC.CTLA	EQU	0100	CTL-A
000.020	143X CC.CTLB	EQU	0200	CTL-B
000.040	144X CC.CTLC	EQU	0400	CTL-C
000.100	145X CC.CTLD	EQU	1000	CTL-D



HEATH H8 CONSOLE DRIVER
SYSTEM I/O DRIVER

HEATH X8ASM V1.1 06/21/77
17:51:24 01-AFR-77 PAGE 5



147X ** PROGRAM ENTRY POINTS.

148X

040.100 303 333 333 149X ENTRY JMF CONFIG PROGRAM RESET ENTRY POINT
040.103 303 222 222 150X JMF RESTART PROGRAM RESTART ENTRY POINT

152X ** PORT ROUTINES.

153X *

154X * ALL PROGRAMS MUST USE THESE ROUTINES, TO ALLOW PORT ADDRESS
155X * CHANGEARABILITY.

156X

157X

040.106 333 372 158X \$CDIN IN IP.CDF CONSOLE DATA IN
040.110 311 159X \$RET RET

160X

040.111 323 372 161X \$CDOUT OUT OF.CDF CONSOLE DATA OUT
040.113 311 162X RET

163X

040.114 333 373 164X \$CISI IN IP.CIS CONSOLE INPUT STATUS IN
040.116 311 165X RET

166X

040.117 323 373 167X \$CISO OUT OF.CIS CONSOLE INPUT STATUS OUT
040.121 311 168X RET

169X

040.122 333 373 170X \$CUSI IN IP.COS CONSOLE OUTPUT STATUS IN
040.124 311 171X RET

172X

040.125 323 373 173X \$COSO OUT OF.COS CONSOLE OUTPUT STATUS OUT
040.127 311 174X RET

175X

040.130 333 370 176X \$TIIN IN IP.TIF TAPE DATA IN
040.132 311 177X RET

178X

040.133 323 370 179X \$TIOUT OUT OF.TIF TAPE DATA OUT
040.135 311 180X RET

181X

040.136 333 371 182X \$TSIN IN IP.TSF TAPE STATUS IN
040.140 311 183X RET

184X

040.141 323 371 185X \$TSOUT OUT OF.TSF TAPE STATUS OUT
040.143 311 186X RET

188X ** REMOTE ENTRY POINTS FOR CONSOLE DRIVER.

189X *

190X

040.144 303 255 040 191X \$RCHAR JMF \$RCHAR READ ONE CHARACTER

192X

040.147 303 332 040 193X \$WCHAR JMF \$WCHAR WRITE ONE CHARACTER

194X

040.152 303 367 040 195X \$FRSCL JMF \$FRSCL PRESET CONSOLE UART

HEATH H8 CONSOLE DRIVER
SYSTEM I/O DRIVER

HEATH X8ASM V1.1 06/21/77
17:51:48 01-APR-77 PAGE 6

197X ** DATA AND BUFFERS.

198X *

199X

200X

040.155 000	201X \$INBUF DB 0	INPUT BUFFER COUNT
040.156	202X DS 30	TYPE AHEAD BUFFER. FIRST BYTE = NEXT CHARACTER
	203X	
000.035	204X \$INBUFL EQU *-\$INBUF-2	MAX LENGTH OF BUFFER

206X ** CONTROL CHARACTER TABLE.

207X *

208X * DB CHAR,MASK,VALUE

209X *

210X * IF CHAR, \$CSLCTL = (\$CSLCTL ,AND, MASK) ,OR, VALUE

211X

040.214 000.377 000	212X \$CSIR DB 000,377Q,0000	CTL-B (00 - NULL)
040.217 001 167 210	213X DB 01Q,167Q,210Q	CTL-A
040.222 002 157 220	214X DB 02Q,157Q,220Q	CTL-B
040.225 003 137 240	215X DB 03Q,137Q,240Q	CTL-C
040.230 004 077 300	216X DB 04Q,077Q,300Q	CTL-D
040.233 017 177 002	217X DB 17Q,177Q,002Q	CTL-O
040.236 020 175 000	218X DB 20Q,175Q,000Q	CTL-F
040.241 021 176 000	219X DB 21Q,176Q,000Q	CTL-Q
040.244 023 176 001	220X DB 23Q,176Q,001Q	CTL-S
040.247 177	221X DB 17QQ	END OF TABLE

223X ** \$CSIC - ADDRESS OF INTERRUPT TIME CONTROL CHARACTER PROCESSOR.

224X *

225X * \$CSIC CONTAINS THE ADDRESS OF THE ROUTINE CONSL ENTERS WHENEVER A

226X * CONTROL CHARACTER (CTL-A THROUGH CTL-D) IS STRUCK. THE PROPER BITS

227X * ARE SET/CLEARED IN \$CSLCTL, AND THE ROUTINE IS ENTERED AT INTERRUPT

228X * TIME. AFTER THE USER ROUTINE HAS COMPLETED PROCESSING, IT SHOULD CLEAR

229X * THE BITS FROM \$CSLCTL, AND RETURN TO THE SYMBOL '\$RET'. IF INTERRUPT

230X * PROCESSING IS NOT REQUIRED, LEAVE THE \$RET ADDRESS IN \$CSIC.

231X

040.250 110.040	232X \$CSIC DW \$RET	ADDRESS OF USER ROUTINE FOR CTL-A THROUGH CTL-D
-----------------	----------------------	---

233X

040.252 000	234X \$CSLCTL DB 0	CONSOLE CONTROL BYTE
-------------	--------------------	----------------------

040.253 000	235X COLNO DB 0	CONSOLE CURSOR POSITION
-------------	-----------------	-------------------------

040.254 120	236X \$CSLLEN DB 80	CONSOLE WIDTH
-------------	---------------------	---------------

237X

000.040	238X SET \$INBUF/1000A	
---------	------------------------	--

000.000	239X ERRNZ */1000A-	ALL DATA MUST BE IN SAME PAGE
---------	---------------------	-------------------------------



	243X **	\$RCHAR - READ SINGLE CHARACTER.
	244X *	
	245X *	\$RCHAR IS CALLED TO READ A CONSOLE CHARACTER.
	246X *	
	247X *	ENTRY NONE
	248X *	EXIT (A) = CHAR (PARITY BIT CLEARED)
	249X *	USES A,F
	250X	
	251X	
040.255	345	252X \$RCHAR. PUSH H SAVE (HL)
040.256	041 155 040	253X LXI H,\$INBUF (HL) = ADDRESS OF CHARACTER PUNTER
040.261	176	254X \$RCHAR1 MOV A,M (A) = COUNTER
040.262	247	255X ANA A
040.263	312 261 040	256X JZ \$RCHAR1 WAIT FOR INTERRUPT TO READ CHARACTER
	257X	
040.266	363	258X DI INTERLOCK SEQUENCE
040.267	325	259X PUSH D
040.270	065	260X DCR M DECREMENT COUNT
040.271	126	261X MOV D,M (D) = COUNT-1
040.272	043	262X INX H
040.273	176	263X MOV A,M (A) = READ CHARACTER
040.274	376 173	264X CPI 173Q SEE IF LOWER CASE
040.276	322 310 040	265X JNC \$RC1.5
040.301	376 141	266X CPI 141Q
040.303	332 310 040	267X JC \$RC1.5 IS NOT LOWER CASE
040.306	326 040	268X SUI 40Q MAKE UPPER
040.307		269X \$RCHARA EQU *-1 ZEROED FOR LOWER CASE
040.310	365	270X \$RC1.5 PUSH FSW SAVE IT
	271X	
040.311	025	272X \$RCHAR2 DCR D MOVE OTHERS DOWN IN QUEUE
040.312	372 325 040	273X JM \$RCHAR3 NO MORE
040.315	043	274X INX H
040.316	176	275X MOV A,M
040.317	053	276X DCX H
040.320	167	277X MOV M,A
040.321	043	278X INX H
040.322	303 311 040	279X JMP \$RCHAR2 RESTORE INTERRUPTS
	280X	
040.325	373	281X \$RCHAR3 EI
040.326	381	282X POP FSW
040.327	321	283X POP D
040.330	341	284X POP H
040.331	311	285X RET EXIT: (A) = CHAR

HEATH H8 CONSOLE DRIVER
TASK-TIME CONSOLE DRIVERS.

HEATH X8ASM V1.1 06/21/77
17:52:40 01-AFR-77 PAGE 8

```

287X **      $WCHAR - WRITE SINGLE CHARACTER.
288X *
289X *      $WCHAR IS CALLED TO OUTPUT A SINGLE CHARACTER.
290X *
291X *      ENTRY (A) = CHARACTER.
292X *      EXIT  NONE
293X *      USES   NONE.
294X
295X
040.332 365    296X $WCHAR. PUSH  PSW      SAVE CHARACTER
040.333 .072 .252 .040  297X $WCHAR1 LDA  $CSLCTL
000.000        298X ERRNZ CC.HLI-1
040.336 037    299X RAR
040.337 332 333 040 300X JC  $WCHAR1    AM TO WAIT
040.342 037    301X RAR
000.000        302X ERRNZ CC.IMP-2
040.343 .322 .354 .040 303X JNC  $WCHAR2    AM TO PRINT
040.346 361    304X POF  PSW      CTL-U IN EFFECT, DISCARD CHAR
040.347 .311    305X RET
306X
307X *      OVERFLOW TYPE-AHEAD BUFFER, ECHO BELL.
308X
040.350 .065    309X $CSI2 ICR  M      REMOVE LAST CHARACTER INPUT TO MAKE ROOM
040.351 .076 .007  310X MVI  A,BELL
040.353 365    311X PUSH  PSW      STORE CHARACTER
312X
313X *      TYPE CHARACTER.
314X
040.354 .315 .122 .040 315X $WCHAR2 CALL  $COSI
000.000        316X ERRNZ USR.TXR-1
040.357 037    317X RAR
040.360 322 354 040 318X JNC  $WCHAR2    WAIT FOR ROOM
040.363 361    319X POF  PSW
040.364 303 111 040 320X JMP  $CIOUT    OUTPUT AND EXIT

```

```

322X **      $FRSCL - RESET CONSOLE DRIVERS.
323X *
324X *      $FRSCL IS CALLED TO RESET TERMINAL OPERATIONS.
325X *      IT ASSUMES THAT THE SAME PORT IS BEING USED FOR CONSOLE INPUT AS FOR
326X *      CONSOLE OUTPUT, BECAUSE THE CONFIGURATION IS DONE THROUGH
327X *      THE INPUT PORT ROUTINES.
328X *
329X *      ENTRY  NONE
330X *      EXIT   NONE
331X *      USES   A,F
332X
333X
040.367 .076 .201  334X $FRSCL. MVI  A,2010
040.371 315 117 040 335X CALL  $CISO      GUARANTEE OUT OF MODE-SET
040.374 .315 .191 .040 336X CALL  $TSOUT
040.377 .076 100   337X MVI  A,UCI.IR
041.001 .315 .117 .040 338X CALL  $CISO
041.004 315 141 040 339X CALL  $TSOUT      FORCE INTO MODE-SET

```



HEATH HG CONSOLE DRIVER
TASK-TIME CONSOLE DRIVERS.

HEATH X8ASM V1.1 06/21/77
17:53:05 01-APR-77 PAGE 9

041.007	076 116	340X	MVI	A,UMI,1B+UMI,L8+UMI,16X
041.011	315 117 040	341X	CALL	\$CISO
041.014	315 141 040	342X	CALL	\$TSOUT
041.017	257	343X	XRA	A
041.020	062 155 040	344X	STA	\$INBUF
041.023	076 303	345X	MVI	A,303Q SET UP INTERRUPT VECTOR
041.025	062 045 040	346X	STA	.UIVEC+6
041.030	041 043 041	347X	LXI	H,\$CSINT
041.033	042 046 040	348X	SHLD	.UIVEC+7
041.036	076 027	349X	MVI	A,UCI,ER+UCI.RE+UCI.IE+UCI.TE
041.040	303 117 040	350X	JMP	\$CISO ENABLE CONSOLE OUTPUT



HEATH H8 CONSOLE DRIVER
INTERRUPT-TIME CONSOLE DRIVER.

HEATH X8ASM V1.1 06/21/77
17:53:10 01-AFR-77 PAGE 10

```

354X ** $CSINT - CONSOLE INTERRUPT PROCESSOR.
355X *
356X * CONSOLE READ INTERRUPTS ENTER HERE FROM PAM-8.
357X *
358X * ENTRY NONE.
359X * EXIT 'EI' AND RET.
360X * USES NONE.
361X
362X
041.043 345 363X $CSINT PUSH H
041.044 365 364X PUSH FSW PRESERVE REGISTERS
041.045 315 114 040 365X CALL $CISI
041.050 346 002 366X ANI USR.RXR
041.052 304 061 041 367X CNZ $CSI1 IF HAVE DATA FROM INPUT
041.055 361 368X $CSIX POP PSW
041.056 341 369X POP H
041.057 373 370X EI
041.060 311 371X RET
372X
373X ** HAVE DATA INPUT INTERRUPT. SEE IF ROOM IN THE QUEUE.
374X
041.061 041 155 040 375X $CSI1 LXI H,$INBUF
041.064 076 035 376X MVI A,$INBUF
041.066 226 377X CMF M
041.067 334 350 040 378X CC $CSI2 NO ROOM.
379X
380X * ADD CHARACTER TO QUEUE.
381X
041.072 064 382X $CSI3 INR M
041.073 176 383X MOV A,M
041.074 205 384X ADD L
041.075 157 385X MOV L,A (HL) = ADDRESS OF CHARACTER
041.076 315 106 040 386X CALL $CIIN INPUT CHARACTER
041.101 346 177 387X ANI 177Q TRIM IT
041.103 167 388X MOV M,A STORE IT
389X
390X * CHECK FOR SPECIAL CONTROL SEQUENCES.
391X
392X * CTL-A TO CTL-D SPECIAL USER INTERRUPT PROCESSING
393X * CTL-O SET DISCARD MODE
394X * CTL-P CLEAR DISCARD MODE
395X * CTL-Q CLEAR HOLD MODE
396X * CTL-S SET HOLD MODE
397X
041.104 376 040 398X CPI
041.106 320 399X RNC NOT CONTROL CHARACTER
400X
401X * HAVE CONTROL CHARACTER.
402X
041.107 056 212 403X MVI L,$CSI8-2
041.111 043 404X $CSI5 INX H POINT AT NEXT ELEMENT IN TABLE
041.112 043 405X INX H COMPARE TO CHARACTER
041.113 226 406X CMP M IS NOT IN LIST
041.114 330 407X RC
041.115 043 408X INX H
041.116 302 111 041 409X JNE $CSI5 IS NOT THIS ONE

```

HEATH K8 CONSOLE DRIVER
INTERRUPT TIME CONSOLE DRIVER

HEATH X8ASM V1.1 06/21/77
17:53:40 01-APR-77 PAGE 11



041.121 072 252 040	410X	LIA	\$CSLCTL	
041.124 240	411X	ANA	M	CLEAR \$CSLCTL BITS
041.125 043	412X	INX	H	
041.126 256	413X	XRA	M	SET \$CTLCTL BITS
041.127 062 252 040	414X	STA	\$CTLCTL	
041.132 365	415X	PUSH	PSW	SAVE BITS
041.133 056 155	416X	MVI	L,\$\$INBUF	
041.135 065	417X	DCR	M	DECREMENT CHARACTER FROM BUFFER
041.136 361	418X	FOP	PSW	
041.137 360	419X	RF		RETURN IF NOT CTL-A THROUGH CTL-D
	420X			
	421X *			HAVE SPECIAL CONTROL, CALL USER.
	422X			
041.140 052 250 040	423X	LHLD	\$CSIC	
041.143 351	424X	PCHI		CALL USER ROUTINE

041.144
ASSEMBLY COMPLETE
426 STATEMENTS
0 ERRORS DETECTED
22612 BYTES FREE

HEATH H8 CONSOLE DRIVER
CROSS REFERENCE TABLE

XREF V1.1
PAGE 12

\$CDIN	040106	158L	386
\$CDOUT	040111	161L	320
\$CISI	040114	164L	365
\$CISO	040117	167L	335 338 341 350
\$COSI	040122	170L	315
\$COSO	040125	173L	
\$CSI1	041061	367	375L
\$CSI2	040350	309L	378
\$CSI3	041072	382L	
\$CSIS	041111	404L	409
\$CSIK	040214	212L	403
\$CSIC	040250	232L	423
\$CSINT	041043	347	363L
\$CSIX	041055	368L	
\$CSLCTL	040252	234L	297 410 414
\$CSLLEN	040254	236L	
\$INBUF	040155	201L	204 238 253 344 375 418
\$INRUF	000035	204E	376
\$PRSC1	040152	195L	
\$PRSC1.	040367	195	334L
\$RC1.5	040310	265	267 270L
\$RCHAR	040144	191L	
\$RCHAR.	040255	191	252L
\$RCHAR1	040261	254L	256
\$RCHAR2	040311	272L	279
\$RCHAR3	040325	273	281L
\$RCHARA	040307	269E	
\$RET	040110	159L	232
\$TIN	040130	176L	
\$TIOUT	040133	179L	
\$TSIN	040136	182L	
\$TSOUT	040141	185L	336 339 342
\$WCHAR	040147	193L	
\$WCHAR.	040332	193	296L
\$WCHAR1	040333	297L	300
\$WCHAR2	040354	303	315L 318
.	000040	238S	239
.UIVEC	040037	24E	346 348
BELL	000007	23E	310
CC.CTLA	000010	142E	
CC.CTLB	000020	143E	
CC.CTLC	000040	144E	
CC.CTLD	000100	145E	
CC.DMF	000002	141E	302
CC.HLB	000001	140E	298
COLNO	040253	235L	
CONFIG	333333	27E	149
ENTRY	040100	149L	
IP.CDF	000372	119E	158
IP.CIS	000373	121E	164
IP.CUS	000373	123E	170
IP.TIF	000370	126E	176
IP.TSF	000371	128E	182
IP.CDF	000372	129E	161
IP.CIS	000373	122E	167
IP.CUS	000373	124E	173
IP.TIF	000370	127E	179
IP.TSF	000371	129E	185

HEATH HB CONSOLE DRIVER
CROSS REFERENCE TABLE

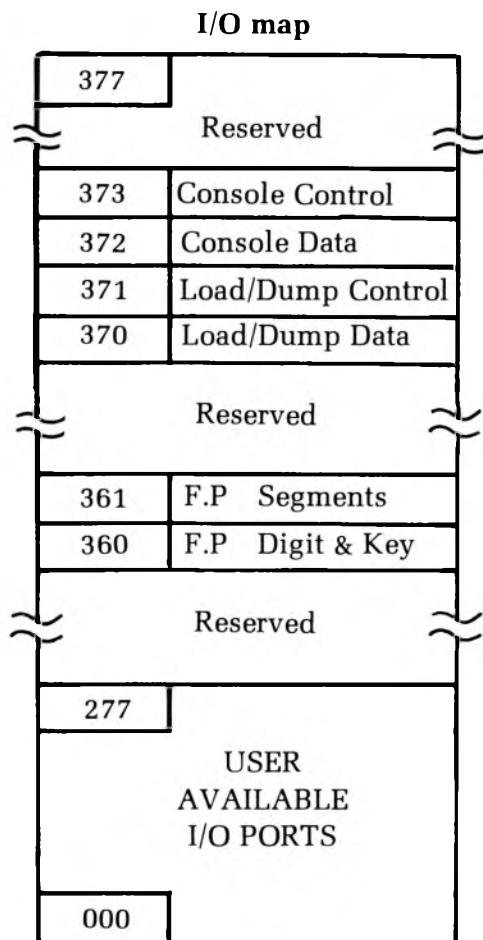
XREF V1.1
PAGE 13

RESTART	222222	26E	150
START	111111	25E	
UCI.EC	000020	54E	349
UCI.IE	000002	56E	349
UCI.IR	000100	52E	337
UCI.RE	000004	55E	349
UCI.RU	000040	53E	
UCI.TE	000001	57E	349
UMI.16X	000002	47E	340
UMI.1B	000100	37E	340
UMI.1X	000001	46E	
UMI.2R	000300	39E	
UMI.64X	000003	48E	
UMI.HB	000200	38E	
UMI.L5	000000	42E	
UMI.L6	000004	43E	
UMI.L7	000010	44E	
UMI.L8	000014	45E	340
UMI.PA	000020	41E	
UMI.PE	000040	40E	
USR.FE	000040	61E	
USR.DE	000020	62E	
USR.PE	000010	63E	
USR.RXR	000002	65E	366
USR.TXE	000004	64E	
USR.TXR	000001	66E	316

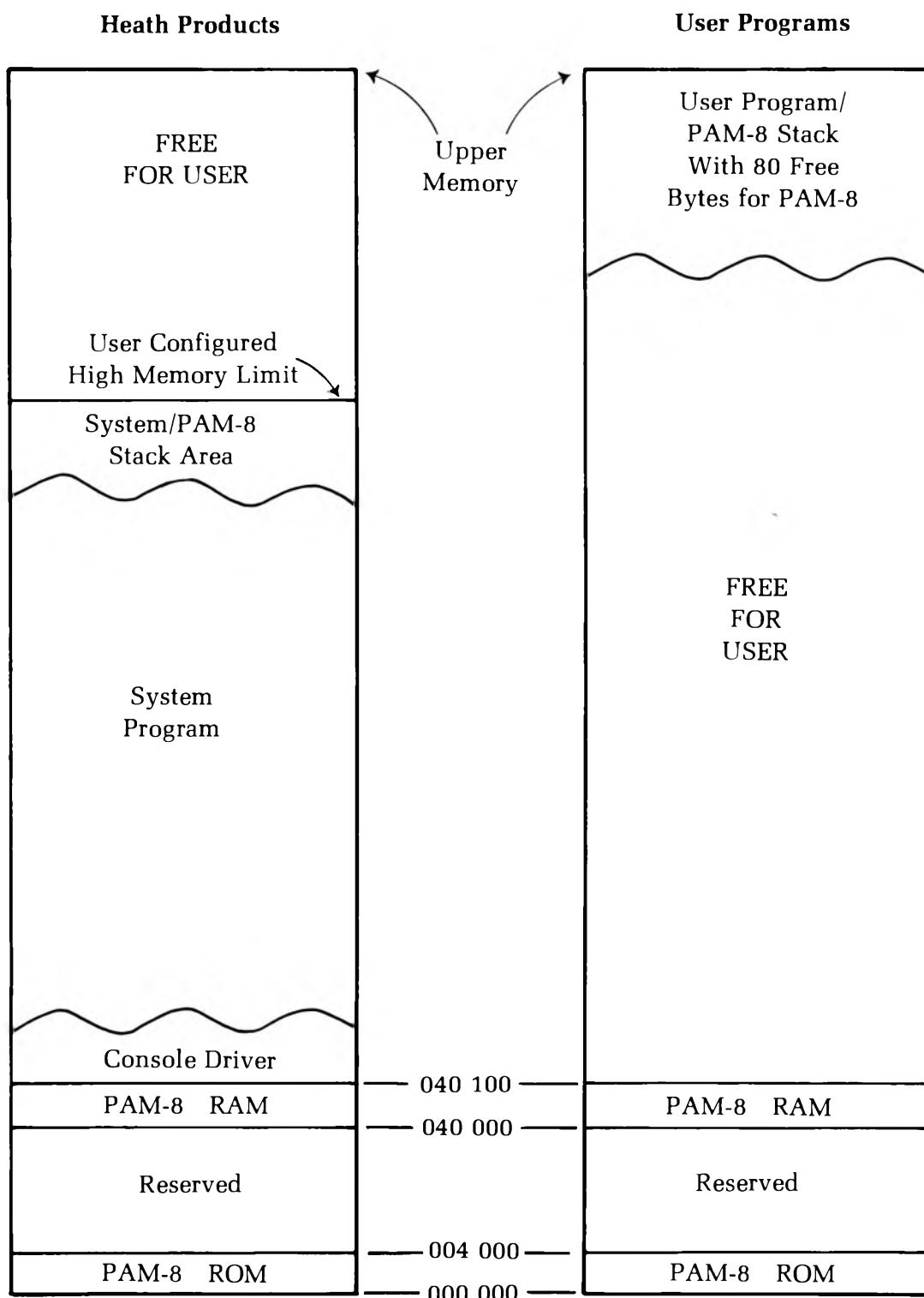
31702 BYTES FREE

APPENDIX C

This appendix contains I/O and Memory Maps for the H8 software products.



MEMORY MAP





APPENDIX D

ASCII Characters

<u>7-BIT OCTAL CODE</u>	<u>DECIMAL CODE</u>	<u>CHARACTER</u>	<u>DESCRIPTION</u>
000	0	NUL	NULL, TAPE FEED, CONTROL SHIFT P.
001	1	SOH	START OF HEADING; ALSO SOM, START OF MESSAGE, CONTROL A,
002	2	STX	START OF TEXT; ALSO EOA, END OF ADDRESS, CONTROL B,
003	3	ETX	END OF TEXT: ALSO EOM, END OF MESSAGE CONTROL C,
004	4	EOT	END OF TRANSMISSION (END): CONTROL D.
005	5	ENQ	ENQUIRY; ALSO WRU, CONTROL E,
006	6	ACK	ACKNOWLEDGE. ALSO RU, CONTROL F.
007	7	BEL	RINGS THE BELL. CONTROL G.
010	8	BS	BACKSPACE: ALSO FEO, FORMAT EFFECTOR BACKSPACE SOME MACHINES, CONTROL H.
011	9	HT	HORIZONTAL TAB. CONTROL I
012	10	LF	LINE FEED (NEW LINE): ADVANCES PAPER TO NEXT LINE, DUPLICATED BY CONTROL J.
013	11	VT	VERTICAL TAB (VTAB). CONTROL K.
014	12	FF	FORM FEED TO TOP OF NEXT PAGE (PAGE). CONTROL L.
015	13	CR	CARRIAGE RETURN TO BEGINNING OF LINE. DUPLICATED BY CONTROL M.
016	14	SO	SHIFT OUT: CHANGES RIBBON COLOR TO RED. CONTROL N.
017	15	SI	SHIFT IN: CHANGES RIBBON COLOR TO BLACK. CONTROL O.
020	16	DLE	DATA LINK ESCAPE. CONTROL P (DCO).
021	17	DC1	DEVICE CONTROL 1, TURNS TRANSMITTER (READER) ON, CONTROL Q (XON).
022	18	DC2	DEVICE CONTROL 2, TURNS PUNCH OR AUXILIARY ON, CONTROL R (TAPE, AUX ON).
023	19	DC3	DEVICE CONTROL 3, TURNS TRANSMITTER (READER) OFF, CONTROL S (XOFF).
024	20	DC4	DEVICE CONTROL 4, TURNS PUNCH OR AUXILIARY OFF. CONTROL T (TAPE, AUX OFF).
025	21	NAK	NEGATIVE ACKNOWLEDGE: ALSO ERR. ERROR. CONTROL U.
026	22	SYN	SYNCHRONOUS IDLE (SYNC). CONTROL V.
027	34	ETB	END OF TRANSMISSION BLOCK: ALSO LEM. LOGICAL END OF MEDIUM. CONTROL W.

7-BIT

OCTAL	DECIMAL		
<u>CODE</u>	<u>CODE</u>	<u>CHARACTER</u>	<u>DESCRIPTION</u>
030	24	CAN	CANCEL (CANCL). CONTROL X.
031	25	EM	END OF MEDIUM. CONTROL Y.
032	26	SUB	SUBSTITUTE. CONTROL Z.
033	27	ESC	ESCAPE. PREFIX.
034	28	FS	FILE SEPARATOR. CONTROL SHIFT L.
035	29	GS	GROUP SEPARATOR. CONTROL SHIFT M.
036	30	RS	RECORD SEPARATOR. CONTROL SHIFT N.
037	31	US	UNIT SEPARATOR. CONTROL SHIFT O.
040	32	SP	SPACE.
041	33	!	
042	34	"	
043	35	#	
044	36	\$	
045	37	%	
046	38	&	
047	39	'	ACUTE ACCENT OR APOSTROPHE.
050	40	(
051	41)	
052	42	*	
053	43	+	
054	44	,	
055	45	-	
056	46	.	
057	47	/	
060	48	0	
061	49	1	
062	50	2	
063	51	3	
064	52	4	
065	53	5	
066	54	6	
067	55	7	
070	56	8	
071	57	9	
072	58	:	
073	59	;	
074	60	<	
075	61	=	
076	62	>	
077	63	?	
100	64	@	
101	65	A	
102	66	B	
103	67	C	
104	68	D	
105	69	E	
106	70	F	
107	71	G	



<u>7-BIT OCTAL CODE</u>	<u>DECIMAL CODE</u>	<u>CHARACTER</u>	<u>DESCRIPTION</u>
110	72	H	
111	73	I	
112	74	J	
113	75	K	
114	76	L	
115	77	M	
116	78	N	
117	79	O	
120	80	P	
121	81	Q	
122	82	R	
123	83	S	
124	84	T	
125	85	U	
126	86	V	
127	87	W	
130	88	X	
131	89	Y	
132	90	Z	
133	91	[SHIFT K
134	92]	SHIFT L
135	93	↑	SHIFT M
136	94	←	SHIFT N
137	95		
140	96		ACCENT GRAVE.
141	97	a	
142	98	b	
143	99	c	
144	100	d	
145	101	e	
146	102	f	
147	103	g	
150	104	h	
151	105	i	
152	106	j	
153	107	k	
154	108	l	
155	109	m	
156	110	n	
157	111	o	
160	112	p	
161	113	q	
162	114	r	
163	115	s	
164	116	t	
165	117	u	
166	118	v	
167	119	w	



<u>7-BIT OCTAL CODE</u>	<u>DECIMAL CODE</u>	<u>CHARACTER</u>	<u>DESCRIPTION</u>
170	120	x	
171	121	y	
172	122	z	
173	123		
174	124		
175	125		THIS CODE GENERATED BY ALT MODE.
176	126		THIS CODE GENERATED BY ESC KEY (IF PRESENT)
177	127	DEL	DELETE, RUB OUT.



Appendix E

Decimal To Octal Tables

for 0 to 255_{10}

<u>DECIMAL OCTAL</u>	<u>DECIMAL OCTAL</u>	<u>DECIMAL OCTAL</u>
0 0	37 45	74 112
1 1	38 46	75 113
2 2	39 47	76 114
3 3	40 50	77 115
4 4	41 51	78 116
5 5	42 52	79 117
6 6	43 53	80 120
7 7	44 54	81 121
8 10	45 55	82 122
9 11	46 56	83 123
10 12	47 57	84 124
11 13	48 60	85 125
12 14	49 61	86 126
13 15	50 62	87 127
14 16	51 63	88 130
15 17	52 64	89 131
16 20	53 65	90 132
17 21	54 66	91 133
18 22	55 67	92 134
19 23	56 70	93 135
20 24	57 71	94 136
21 25	58 72	95 137
22 26	59 73	96 140
23 27	60 74	97 141
24 30	61 75	98 142
25 31	62 76	99 143
26 32	63 77	100 144
27 33	64 100	101 145
28 34	65 101	102 146
29 35	66 102	103 147
30 36	67 103	104 150
31 37	68 104	105 151
32 40	69 105	106 152
33 41	70 106	107 153
34 42	71 107	108 154
35 43	72 110	109 155
36 44	73 111	110 156

<u>DECIMAL</u>	<u>OCTAL</u>	<u>DECIMAL</u>	<u>OCTAL</u>	<u>DECIMAL</u>	<u>OCTAL</u>
111	157	160	240	209	321
112	160	161	241	210	322
113	161	162	242	211	323
114	162	163	243	212	324
115	163	164	244	213	325
116	164	165	245	214	326
117	165	166	246	215	327
118	166	167	247	216	330
119	167	168	250	217	331
120	170	169	251	218	332
121	171	170	252	219	333
122	172	171	253	220	334
123	173	172	254	221	335
124	174	173	255	222	336
125	175	174	256	223	337
126	176	175	257	224	340
127	177	176	260	225	341
128	200	177	261	226	342
129	201	178	262	227	343
130	202	179	263	228	344
131	203	180	264	229	345
132	204	181	265	230	346
133	205	182	266	231	347
134	206	183	267	232	350
135	207	184	270	233	351
136	210	185	271	234	352
137	211	186	272	235	353
138	212	187	273	236	354
139	213	188	274	237	355
140	214	189	275	238	356
141	215	190	276	239	357
142	216	191	277	240	360
143	217	192	300	241	361
144	220	193	301	242	362
145	221	194	302	243	363
146	222	195	303	244	364
147	223	196	304	245	365
148	224	197	305	246	366
149	225	198	306	247	367
150	226	199	307	248	370
151	227	200	310	249	371
152	230	201	311	250	372
153	231	202	312	251	373
154	232	203	313	252	374
155	233	204	314	253	375
156	234	205	315	254	376
157	235	206	316	255	377
158	236	207	317		
159	237	208	320		

APPENDIX F

Memory Table

Offset Octal and Decimal Boundaries

<u>Hi Byte</u>	<u>Lo Byte</u>	<u>Decimal Boundary</u>
A15.....A8	A7.....A0	
0 0 4	0 0 0	1024
0 2 0	0 0 0	4096
0 4 0	0 0 0	8192
0 6 0	0 0 0	12288
1 0 0	0 0 0	16384
1 2 0	0 0 0	20480
1 4 0	0 0 0	24576
1 6 0	0 0 0	28672
2 0 0	0 0 0	32768
2 2 0	0 0 0	36864
2 4 0	0 0 0	40960
2 6 0	0 0 0	45056
3 0 0	0 0 0	49152
3 2 0	0 0 0	53248
3 4 0	0 0 0	57344
3 6 0	0 0 0	61440
3 7 7	3 7 7	65535*

For example, if you have 12K bytes in an H8, the lower boundary is at 8192, or 040 000 offset octal. The upper boundary is at 8K + 12K = 20K (20480), or 120 000 Octal.

*NOTE: 65,535 is the last location in a memory addressed by 16 bits.



INDEX

- ASR Console, 0-22
Basic Program Format, 0-15
Checksum, 0-12
Compressed, Text Format, 0-15
Configured Tape, 0-19
Console Debugger (BUG-8), 0-8
Console Driver, 0-23
Copying Tapes, 0-22
CRC-16, 0-12
Control Characters, 0-23
Data Formats, 0-14
Distribution Tapes, 0-19
Displays, 0-15
Entry Point, 0-23
File, 0-12
Front Panel Displays, 0-15
Front Panel Entry Points, 0-23
Heath Assembly Language (HASL-8), 0-10
Heath Text Editor (TED-8), 0-8
Installation, 0-19 ff.,
Label Record, 0-14
Loading, 0-21
Magnetic Tape, 0-18
Memory Image Format, 0-14
110-Baud Console, 0-22
Patch Installation 0-22
Pad Characters, 0-20
Panel Monitor (PAM-8), 0-7
Paper Tape, 0-18
ROM, 0-7
Record, 0-12
Record Structure, 0-13
Software Installation, 0-19 ff.,
Software Distribution Tape, 0-19
Software Problems, 0-24
Software Version Code, 0-24
Tape Files, 0-12
Tape Format, 0-12

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
REQUIRED PATCHES FOR

HEATH H8 BUG - 8

Upgrading 02.01.00.
To 02.01.01.

**THESE PATCHES MUST BE INSTALLED IN THE
SEQUENCE EVERY TIME A LOAD IS MADE FROM
THE CONFIGURATION TABLE**

NOTES: None.

USE: These patches cause BUG-8 to remove any outstanding
breakpoints from the program when it is entered at the
warm-start address (040 103).
The breakpoint table is cleared WITHOUT removing any
patches from the program when BUG-8 is entered at the
cold start address (040 100).

043377 061
047040 126
047042 012
050011 031 054
050017 031 054
054023 315 100 045 303 336 043 346 177
054033 303 225 052
054307 023 054
055222 337
057100 061

.....
.....
.....
.....
REQUIRED PATCHES FOR

.....
.....
HEATH...M8...TDP - B.....

.....
.....
**Upgrading 03.01.00.
To 03.01.01.**.....

.....
**THESE PATCHES MUST BE INSTALLED IN THE
PRODUCT_EVERY_TIME_A_LOAD_IS_MADE_EROM
THE_CONFIGURATION_IABE**.....

.....
NOTES: None.

.....
USE: The DELETE command will not leave a 'previous command range'.
That is, a command immediately following a DELETE command
must specify a range via '\$', '/', or '' (blank).
.....

.....
.....
**043356 041
044131 315 240 053 312 175 044 345
044140 052 232 057 353 315 032 061 172
044150 234 341 365
046234 041
050067 041
061032 052 230 057 173 225 311 000
061314 061
064171 061**.....
.....

.....
.....
.....
.....
.....
.....
.....

REQUIRED PATCHES FOR

BENTON HARBOR BASIC

Upgrading 05.01.00.

To 05.01.01.

.....
.....
THESE PATCHES MUST BE INSTALLED IN THE
PRODUCT EVERY TIME A LOAD IS MADE FROM
THE CONFIGURATION TAPE

.....
.....
NOTES: None.

USE: None.

.....
.....
051166 157

054356 126 072

055245 303 115 072 107

072115 322 251 055 361 365 200 303

072124 250 055 361 315 020 067 303 321 054

073001 061

073062 167

076001 061

REQUIRED PATCHES FOR

HEATH H8 HASL - 8

Upgrading 04.01.00.
To 04.01.01.

THESE_EACHES_MUST_BE_INSTALLED_IN_THE
PRODUCT_EVERY_TIME_A_LOAD_IS MADE FROM
.....THE_CONFIGURATION_YAML.....

NOTES: None

USE: None

041221	041
041366	315 072 067
051114	127
053133	303 101 067
053227	112 067
054010	127
055104	064
055112	061
055150	127
055213	325
055231	315 120 067 000
067072	062 122 063 062 126 063 311 302
067102	043 053 315 042 056 303 136 053
067112	315 144 040 303 042 056 062 050
067122	047 376 325 311
073002	064
073010	061

Section 1

PANEL MONITOR

PAM-8



TABLE OF CONTENTS

INTRODUCTION	1-4
THEORY OF OPERATION	
Power Up and Master Clear	1-5
Clock Interrupts	1-5
PAM-8 Modes/Using RST and RTM	1-6
H8 Displays	1-7
H8 Keypad	1-9
DISPLAYING AND ALTERING MEMORY LOCATIONS	
Specifying a Memory Address	1-10
Altering a Memory Location	1-12
Stepping Through Memory	1-13
DISPLAYING AND ALTERING REGISTERS	
Specifying a Register for Display	1-14
Altering the Contents of a Selected Register	1-15
Stepping Through the Registers	1-15
PROGRAM EXECUTION CONTROL	
Initiating Program Execution	1-16
Breakpointing	1-16
Single Instruction Operation	1-17
Interrupting a Program During Execution	1-17



LOAD/DUMP ROUTINES	1-17
Loading From Tape	1-18
Dumping to Tape	1-18
Copying a Tape	1-20
Tape Errors	1-20
 I/O FACILITIES	
Inputting From a Port	1-21
Outputting to a Port	1-21
Addressing Port Pairs	1-21
 ADVANCED CONTROL	
16-Bit Tick Counter (TICCNT)	1-22
Using the Keypad	1-22
Display Usage	1-23
Using Interrupts	1-24
 APPENDIX A	1-25
 INDEX	1-65



INTRODUCTION

This Manual describes the functions and operations of the Heath H8 Panel Monitor Program, PAM-8, which resides permanently in a ROM on the H8 CPU board. PAM-8 provides a sophisticated front panel display and keyboard emulation as well as handling master clear and interrupt operations. Some of the major features of PAM-8 are:

- Memory contents display and alteration.
- Register contents display and alteration.
- Program execution control (both breakpoint and single instruction operation).
- Self-contained bootstraps for program loading and dumping.
- Port input and output routines.

In addition to the above features, PAM-8 can be instructed (by means of a flag byte contained in H8 RAM) to bypass some or all of its normal functions so the sophisticated user can augment or totally replace them.

Communication with the Panel Monitor is accomplished through three devices: the keypad, the 7-segment displays, and the audio alert. The user enters commands and values through the 16-key keypad, and PAM-8 responds visually through the front panel displays. In addition to the front panel displays, PAM-8 provides the keypad entry and function feedback to the built-in speaker. Appropriate signals (short, medium, and long beeps) indicate that commands and data are accepted or rejected.



PAM-8 Modes/Using RST and RTM

PAM-8 is always in either the monitor mode or the user mode. In the monitor mode no user program is executing, PAM-8 loops reading the keypad and refreshing the displays. All commands entered via the keypad are valid; however, the RTM command is meaningless.

When your program is being executed, PAM-8 is in the user mode and the MON LED on the front panel is extinguished. Only two keyboard commands are valid in this mode: RST (master clear) and RTM (Return To Monitor). NOTE: Both of these commands are dual key commands. No single key command is recognized, so a user program may have free use of the entire keypad.

You can return PAM-8 to the monitor mode by using the RTM command (simultaneously press the ϕ and the # keys). This command stops program execution at the end of the current instruction, stores the current value of each register, and returns PAM-8 to the monitor mode. You can then continue your program by pressing the GO key. The RST command (simultaneously press the 0 and the / keys) performs the master clear operation described earlier and does not save any register values.

Normally, when a user program is running, PAM-8 is also running. Thus, if PAM-8 is displaying the contents of the HL register pair and the user program is started, it continues to display the contents of this register pair as the program is run. If the user program changes the contents of the HL pair, the change is immediately reflected in the front panel displays. In a similar manner, if a memory location is displayed when a user program is started, it is displayed during the time the user program is run. If the user program changes the contents of the displayed memory location, the front panel display changes.

Since PAM-8 does not recognize keypad commands in the user mode, the RTM command must be used before the memory location or register being displayed is changed to a new location or a different register. Once you select the new location or different register, you can resume program execution by pressing GO.

NOTE: PAM-8 requires about 10% of the H8 CPU's resources to process the display interrupts. Programs which are compute-bound may be slowed down by simultaneous operation of PAM-8. In this situation, you may wish to turn off the clock interrupts to improve execution time. See "Using Interrupts" on Page 1-24.



PAM-8 Modes/Using RST and RTM

PAM-8 is always in either the monitor mode or the user mode. In the monitor mode no user program is executing, PAM-8 loops reading the keypad and refreshing the displays. All commands entered via the keypad are valid; however, the RTM command is meaningless.

When your program is being executed, PAM-8 is in the user mode and the MON LED on the front panel is extinguished. Only two keyboard commands are valid in this mode: RST (master clear) and RTM (Return To Monitor). NOTE: Both of these commands are dual key commands. No single key command is recognized, so a user program may have free use of the entire keypad.

You can return PAM-8 to the monitor mode by using the RTM command (simultaneously press the ϕ and the # keys). This command stops program execution at the end of the current instruction, stores the current value of each register, and returns PAM-8 to the monitor mode. You can then continue your program by pressing the GO key. The RST command (simultaneously press the 0 and the / keys) performs the master clear operation described earlier and does not save any register values.

Normally, when a user program is running, PAM-8 is also running. Thus, if PAM-8 is displaying the contents of the HL register pair and the user program is started, it continues to display the contents of this register pair as the program is run. If the user program changes the contents of the HL pair, the change is immediately reflected in the front panel displays. In a similar manner, if a memory location is displayed when a user program is started, it is displayed during the time the user program is run. If the user program changes the contents of the displayed memory location, the front panel display changes.

Since PAM-8 does not recognize keypad commands in the user mode, the RTM command must be used before the memory location or register being displayed is changed to a new location or a different register. Once you select the new location or different register, you can resume program execution by pressing GO.

NOTE: PAM-8 requires about 10% of the H8 CPU's resources to process the display interrupts. Programs which are compute-bound may be slowed down by simultaneous operation of PAM-8. In this situation, you may wish to turn off the clock interrupts to improve execution time. See "Using Interrupts" on Page 1-24.

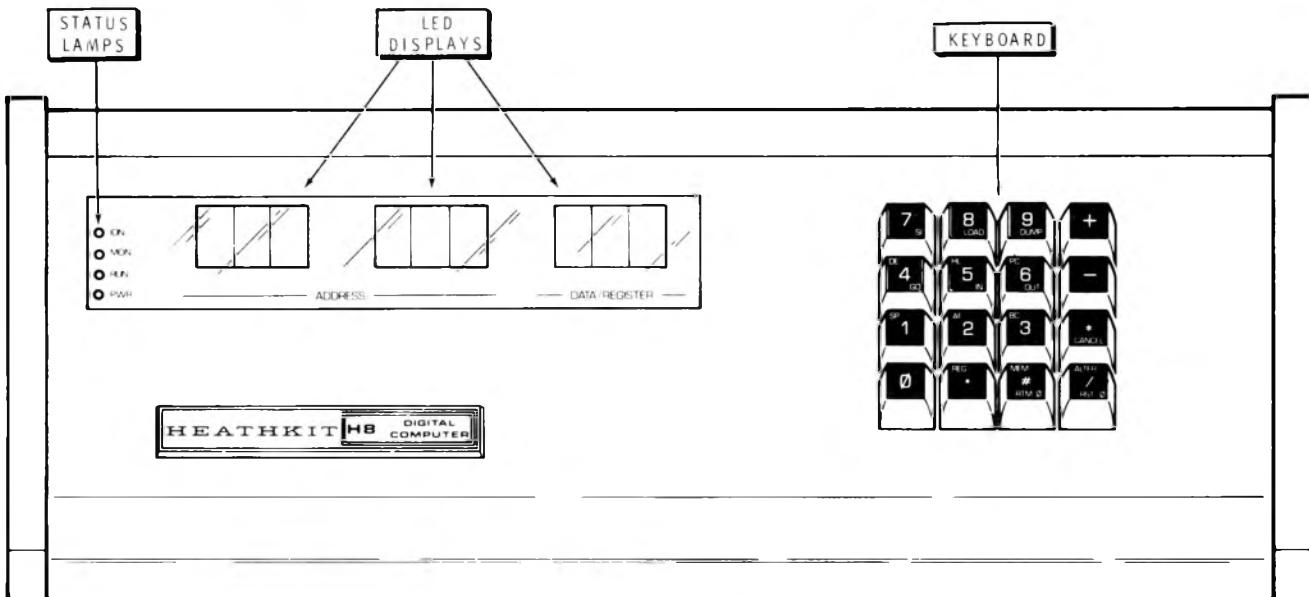


Figure 1-1

H8 Displays

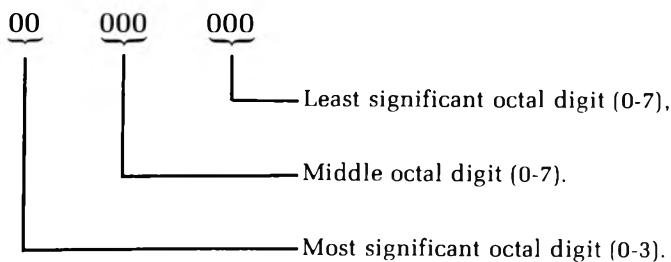
You must understand the H8 front panel presentation in order to use PAM-8. The display is made up of 9 digits, in three groups of three digits each. See Figure 1-1. Each group of three digits displays one byte (eight bits) of information. This information may be the contents of a designated register or memory location, or it may be the address of a memory location itself. The register names are also displayed.

All binary numbers are converted to octal format for display on the H8 front panel. The following table shows binary to octal conversion.

<u>BINARY NUMBER</u>	<u>OCTAL NUMBER</u>
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

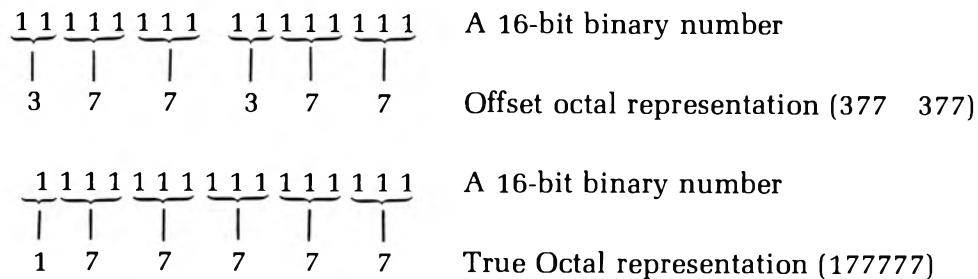


Each byte is displayed as two-and-one-half octal digits. The octal numbers lie in the range of 000 to 377 for binary numbers in the range 00000000 to 11111111, as shown below.



NOTE: As there are only eight bits in a byte, the most significant octal digit only represents two bits and is therefore displayed as 0 to 3. If the user should inadvertently enter the octal digits 4 to 7 into the most significant digit, the most significant bit is lost. Losing this bit converts 4 through 7 into the digits 0 through 3 respectively.

Also note that 16-bit numbers, such as memory addresses and certain register contents, are still displayed as two eight-bit numbers. Therefore, the H8 front panel representation of the number is made up of **two** groups of three octal numbers in the range of 000 to 377. This representation of 16-bit binary numbers is known as **offset octal**, and is used consistently throughout all H8 displays of 16-bit numbers. Offset octal must not be confused with octal. For example:



The lower example shows true octal representation of a 16-bit binary number. This is **not** used by the H8 front panel displays or any H8 software. Occasionally you will see offset octal numbers printed with a decimal point separating the upper and lower bytes. For example:

377.377

Hi Byte Lo Byte



H8 Keypad

The H8 Keypad consists of 16 keys, as shown in Figure 1-1. When the keypad is operating under the control of PAM-8, it exhibits a number of unique properties.

- Each keystroke is verified by a short beep from the audio alert.
- Octal digits are entered using the keys 0 through 7.
- Holding a key down continuously repeats the key's function.
- The + key increments memory port or register locations.
- The - key decrements memory port or register locations.
- The * key cancels previous keypad entries.
- The ALTER key causes PAM-8 to enter the alter mode.
- The MEM key causes PAM-8 to enter the display memory mode.
- The REG key causes PAM-8 to enter the register mode.

Many of the keys on the keypad have multiple functions, depending on the PAM-8 mode being used. In the register mode, for example, the numeric keys (1-6) call the register indicated in the upper left-hand corner of the key. When the PAM-8 is in neither the register nor the memory mode, the keys perform the functions indicated in the lower right-hand corner of the key.

The # and / keys have additional special functions, as indicated earlier. When the / key is pressed simultaneously with the 0 key, the RST (master clear) sequence is initiated. When the # sign key is depressed simultaneously with the 0 key, the RTM (Return To Monitor) function is initiated, the user program is stopped, and PAM-8 regains control.

Each key is covered in greater detail as the various function are discussed.



DISPLAYING AND ALTERING MEMORY LOCATIONS

One of the major features of PAM-8 is its ability to examine the contents of any H8 memory location and to modify the contents of that memory location if it is RAM.

When the H8 is first powered up, PAM-8 is in the display memory mode. This mode is indicated by all digits displaying octal numbers and no decimal points being on.

Specifying a Memory Address

If you wish to display or alter the contents of a memory location. You must first place PAM-8 in the memory address mode and then enter the desired memory address. Place PAM-8 in the memory address mode (if not already there) by pressing the MEM (Memory) key. Specify the address to be displayed or altered by entering the 6-digit address (offset octal).

When you press the MEM key, all the decimal points will light. This indicates that the address may now be entered. Once the full 6-digit address is entered, the decimal points turn off, indicating that address entry is completed. After all 6 digits are entered, the address is displayed in the left-most six displays, and the contents of the addressed memory location are displayed in the right-hand 3 digits.

NOTE: As you press each key, including the MEM key, a short beep indicates successful entry. As each group of three octal digits is successfully entered, a medium beep is sounded. The sequence by which you specify a memory address is shown in Figure 1-2.

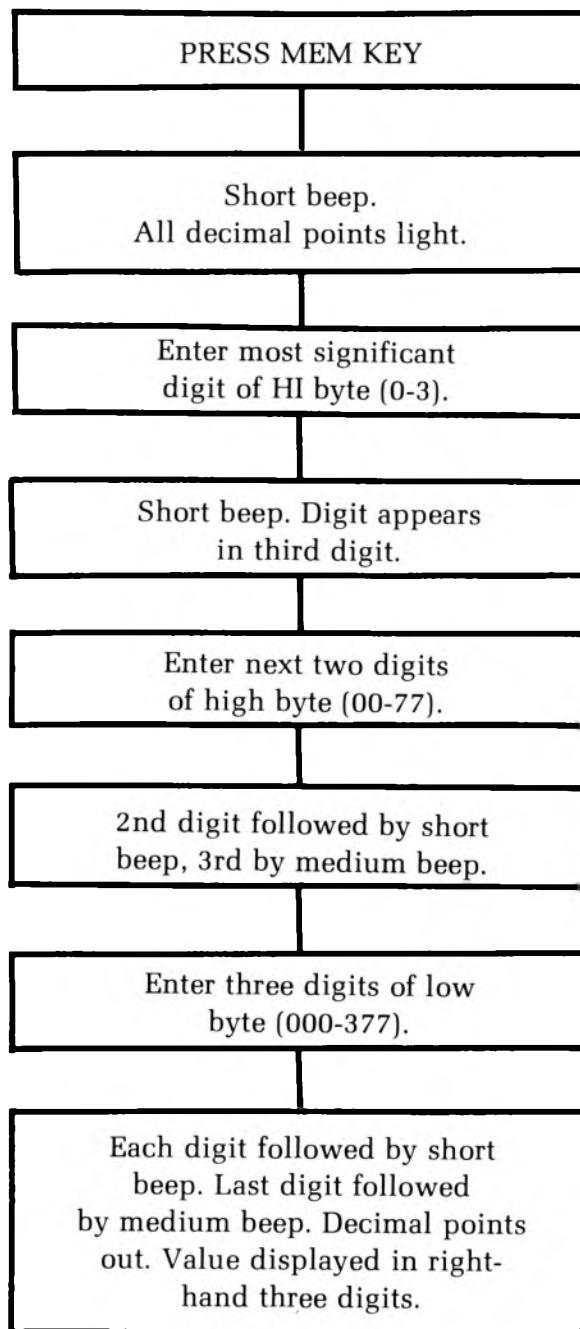


Figure 1-2
Entering a memory address through PAM-8.

NOTE: If you press a non-octal digit key as one of the six address digits, an error is flagged (a long beep). Once this error is flagged, the PAM-8 considers the address complete and extinguishes the decimal points. The entire sequence must be repeated.



Altering a Memory Location

Before you can alter a memory location, you must first display the contents of the memory location by specifying the memory address as described in the preceding paragraphs. After you specify the memory address, press the ALTER key. This will cause PAM-8 to enter the memory alter mode.

When PAM-8 enters the memory alter mode, a single decimal point rotates from right to left through all 9 digits. You can now alter the contents of the displayed location by entering the new octal value (three digits on the keypad). When the three digits have been entered, acoustical verification (a short beep) is given **and the memory address is incremented**. You can then alter this new location by entering three more digits or pressing one of the following keys, causing the monitor to perform the indicated function:

<u>KEY</u>	<u>FUNCTION</u>
+	Increment the address.
-	Decrement the address.
MEM	Specify a new memory address (leave memory alter mode).
REG	Specify a register for display (leave memory alter mode).
ALTER	Exit from the alter mode (into the display mode).

NOTE: PAM-8 automatically increments the memory address as each entry (3 octal digits) is complete. Therefore, you may load a program in sequential locations very rapidly. Each location is modified by simply entering the three octal digits.



The following example reviews each step as the H8 is turned on; the memory address mode is entered; and the location 040 123 is addressed, altered to 345, checked, and closed.

<u>DISPLAY</u>	<u>COMMENTS</u>
X X X X X X X X X	Random memory display at power up (X= random number.)
X.X.X. X.X.X. X.X.X.	MEM key pressed. (In memory address mode, a short beep.)
X.X.0. X.X.X. X.X.X.	0 key pressed. (Short beep.)
X.0.4. X.X.X. X.X.X.	4 key pressed. (Short beep.)
0.4.0. X.X.X. X.X.X.	0 key pressed. (Medium beep.) Contents of location 040 XXX displayed.)
0.4.0. X.X.1. X.X.X.	1 key pressed. (Short beep. Contents of 040 XX1 displayed.)
0.4.0. X.1.2. X.X.X.	2 key pressed. (Short beep. Contents of 040 X12 displayed.)
0 4 0 1 2 3 X X X	3 key pressed. (Medium beep. Contents of desired location 040 123 displayed, decimal points out.)
0.4.0 1.2.3 X.X.X	ALTER key pressed. (Short beep. Decimal points rotate .)
0.4.0. 1.2.3. X.X.3.	3 key pressed. (Short beep. Decimal points rotate .)
0.4.0. 1.2.3. X.3.4.	4 key pressed. (Short beep. Decimal points rotate .)
0.4.0. 1.2.4. X.X.X.	5 key pressed. (Medium beep. Address increments one location. Decimal points rotate .)
0.4.0 1.2.3 3.4.5	-key pressed. (Short beep. Address decrements one location. Decimal points rotate .)
0 4 0 1 2 3 3 4 5	ALTER key pressed. (Short beep. Decimal points go out.)

Stepping Through Memory

When PAM-8 is either in the display memory or alter memory modes, the + and - keys increment and decrement the memory address. Each time you press the key, PAM-8 increments (or decrements) the memory address one location. If you hold the key down, the auto-repeat function of PAM-8 causes the memory address to increment or decrement repeatedly (approximately one location every second).



DISPLAYING AND ALTERING REGISTERS

PAM-8 can display and alter the contents of the 8080 CPU registers, just as it displays and alters the contents of H8 memory locations. Although the process is quite similar, a few special features should be noted.

Specifying a Register for Display

Press the REG key to specify that a register is to be displayed. After you press the REG key, press a second key (SP through PC, see the Table below) to specify the desired register or register pair.

When the REG key is pressed, six decimal points light, indicating that you must now select a register. NOTE: Simply pressing the REG key causes a register name to appear in the right-hand digits. However, you must select a register using the Register Select key before a register is definitely selected and its true contents are displayed. Once a register is selected, the decimal points are extinguished.

The contents of the selected register pair are displayed in the six left-most displays. The register name (or names) are displayed in the two right-most digits of the right-hand three displays. The registers are selected and displayed in accordance with the following table:

<u>KEY</u>	<u>LEFT 3 DIGITS</u>	<u>MIDDLE 3 DIGITS</u>	<u>RIGHT PAIR</u>	<u>COMMENTS</u>
SP (1)	000 to 377	000 to 377	<i>SP</i>	Stack pointer
AF (2)	000 to 377	000 to 377	<i>AF</i>	AF Register pair
BC (3)	000 to 377	000 to 377	<i>bC</i>	DC Register pair
DE (4)	000 to 377	000 to 377	<i>dE</i>	DE Register pair
HL (5)	000 to 377	000 to 377	<i>HL</i>	HL Register pair
PC (6)	000 to 377	000 to 377	<i>PC</i>	Program counter

NOTE: The contents of any single eight-bit register may lie in the range of 000 to 377 octal. The stack pointer (SP) and the program counter (PC) are 16-bit registers and are displayed as two sets of three octal numbers. Each 3-digit grouping corresponds to one byte (8 bit number). When a register pair is displayed, the left three digits correspond to the left register and the middle three digits correspond to the right register. For example:

256 312 AF

Register A contains 256 and F contains 312.



Altering the Contents of a Selected Register

To alter the contents of a register (or register pair), you must first specify it as described in the preceding paragraphs. After you select the register or register pair, press the ALTER key. This will cause the six left-hand decimal points to rotate right to left, indicating that you may enter 6 digits to alter the contents of the indicated register or register pair.

Alternately, you may press one of the following command keys:

<u>KEY</u>	<u>FUNCTION</u>
+	Changes the register pair being displayed.
-	Changes the register pair being displayed.
MEM	Specify a new memory address (leave the alter register mode).
REG	Specify a new register for display (leave alter register mode).
ALTER	Exit the register alter mode.

NOTE: Stack pointer register (SP) is not a direct display of the real stack pointer register, but simply a copy of the real stack pointer register and is used for display purposes only. The stack pointer cannot be altered from the front panel. To alter the stack pointer register, an SPHL (SPHL = 371) instructions must be written into memory. The desired new stack pointer value is then placed in the HL register pair. PAM-8's single instruction mode is used to execute the SPHL swap instructions, loading the stack pointer with the contents loaded in the HL register pair.

Stepping Through the Registers

Use + and - keys to change the register pair being displayed. For example, if the DE register pair is being displayed, press the + key causes the next sequential register pair to be displayed (the HL pair). In the same manner, pressing the - key causes the register to decrement to the preceding pair. For example, if the DE pair is being displayed, pressing the - key displays the BC register pair. NOTE: Holding down either the + key or the - key causes the display to continuously increment or decrement through all the six registers/register pairs.



PROGRAM EXECUTION CONTROL

PAM-8 supports three basic program execution control facilities:

- Beginning or starting execution.
- Breakpointing.
- Single instruction.

Each of these execution controls permits the programmer to execute the desired portions of a program and examine its effects. He may execute the entire program, or a small group of instructions, or a single program instruction.

Initiating Program Execution

To begin the execution of a program residing in H8 memory, place the address of the first instruction to be executed in the PC (program counter). Use the methods described in "Displaying and Altering Registers" (Page 1-14). Once the address of this first instruction is placed in the program counter, press the GO key and program execution will begin. NOTE: Unless the program disables the front panel, the display continues to be actively updated, although the front panel commands are no longer active (except for RST and RTM). If the program counter is displayed when you press the GO key, PAM-8 continuously monitors the program counter.

Breakpointing

Breakpointing permits the programmer to execute small portions of a program and then return to PAM-8. Breakpointing is especially useful when a program is being "debugged." Small portions of the program may be executed and their results observed. If there is an error, it may be corrected before an entire program is involved.

When the H8 executes a program and encounters a halt instruction, it re-enters PAM-8 and sounds the alarm. All of the registers are preserved and the program counter points to the address **following** the address of the halt instruction. Thus, you can breakpoint a program from the front panel by inserting halt instructions (HLT = 166) at the desired points throughout the program. When a particular section of the program is tested and the breakpoint feature is no longer required, you can change the halt to a NOP (NOP = 000). Once the halts are changed to NOPs, execution of the NOP simply passes control to the next successive instruction. Program execution for breakpointing uses the GO key as described above.



NOTE: If you temporarily replace an existing instruction with a halt, you must restore the instruction before resuming program execution. The contents of the program counter point to the address **following** the halt. Therefore, if the instruction which replaced the halt is to be executed, when the program continues, the contents of the program counter must be decremented one location before execution is resumed.

Single Instruction Operation

Any user program may be operated in the single instruction mode. This procedure is identical to the GO command, except that the SI key is pressed rather than the GO key. When the SI key is pressed, a **single instruction** (not a single machine cycle) is executed and then control is returned to PAM-8. Single instruction operation is available for careful inspection of program results and for executing special programs, such as swapping the HL register pair with the stack pointer as discussed in "Altering the Contents of a Selected Register" (Page 1-15).

Interrupting a Program During Execution

You can interrupt a running program (with all registers preserved at the point of interruption) by pressing RTM & 0. You can then examine and/or alter the contents of various memory locations and all the registers as required. Resume execution of the program at the next sequential instruction by simply pressing the GO key. NOTE: Although all registers and memory locations are preserved when RTM & 0 are pressed, it is very difficult to stop a program at an exact location. Therefore, use the breakpoint feature if you want to stop the program at an exact location.

LOAD/DUMP ROUTINES

PAM-8 contains a routine that lets you load and dump memory contents from or to a tape (either paper tape or cassette). This feature is especially important, as most computers require one or two successive "boot strap" routines to be hand-loaded before a desired program can be loaded into the main memory. All these "boot strap" routines are contained within the PAM-8 ROM, and use sophisticated error checking techniques. Thus, a program can be loaded or dumped by simply pressing a single key.



Loading From Tape

To load from a tape, ready the reader device with the tape to be loaded prior to executing the load command. Place PAM-8 in the display memory mode and press the LOAD key. Once the LOAD key is pressed, PAM-8 starts the tape transport and scans the tape for the first file record.

No change will be seen on the front panel displays until PAM-8 finds the first file. When the first file record is located, PAM-8 checks it to see if it is the first (or only) record in a sequence, and the record is a memory dump record. If it is not a memory dump record, a number two error is flagged (see "Tape Errors" on Page 1-20).

Once a correct record is found, loading proceeds. The loading procedure places the entry point address of the program being loaded in the H8 program counter. The H8 memory is then loaded. The displays continuously show the address being loaded and the data being loaded at these addresses. When the load is complete, PAM-8 sounds a long beep and displays the final memory address. If the load is faulty, a number one error is displayed and the audio alert continuously beeps. (See "Tape Errors," Page 1-20.)

NOTE: You may abort a partial load by using the CANCEL key. Naturally, the load image resulting from this action is incorrect, and should not be executed.

Dumping to Tape

Before dumping a memory image onto tape, the following three dump parameters are required:

- The entry point address (the program starting address).
- The dump starting address.
- The dump ending address.

Set the desired entry point address by placing this value in the program counter (PC). This value will be placed in the program counter whenever you load the program so execution will begin at this address when you press the GO key.

Place the dump starting address into the first two H8 RAM cells. These are: 040 000 (offset octal) and 040 001 (offset octal). NOTE: The low order byte of the address should be placed into location 040 000 and the high order byte of the starting address should be placed into location 040 001.



Enter the dump ending address as a memory address using the # (MEM) key. Then ready the tape transport and press the DUMP key. As the tape dump takes place, the number of bytes left to be dumped and the contents of the memory location being dumped are displayed on the front panel. You can abort a dump by using the CANCEL key. If the CANCEL key is used, an incomplete dump image is left on the tape. This cannot be loaded at a future date. NOTE: A successful load automatically sets up the following three dump parameters:

- A. The program starting locations are stored in locations 040 000 and 040 001.
- B. The program ending location is displayed.
- C. The program counter contains the program entry point.

Figure 1-3A shows the steps of a typical dump sequence and Figure 1-3B shows the steps of a typical load sequence.

1. Set PC to 040 100; (040 100 = entry address).
2. Set 040 000 to 100 (100 = low byte of dump start).
3. Set 040 001 to 040 (040 = high byte of dump start).
4. Enter memory address 052 340 (052 340 = end address of dump).
5. Be sure tape is ready.
6. Press DUMP.

Figure 1-3A
The H8 memory image dump.

1. Be sure tape is ready.
2. Press LOAD.

Figure 1-3B
The H8 memory image load.



Copying a Tape

The beginning and final address of the load image are placed at the appropriate points. Thus, to copy a tape, simply load the tape as described in "Loading From Tape" (Page 1-18). Then ready the dump tape drive and press the DUMP key. A dump then takes place, including entry point, initial address, and final address.

In a similar manner, to load, alter, and then dump, enter only the ending address. The other parameters are unchanged from the load if locations 040 000, 040 001 or the program counter have not been modified during the altering procedure.

Tape Errors

PAM-8 detects two types of tape errors: record errors and checksum errors. In either case, when an error is detected, the tape transport is halted. The error number is then displayed in the center three digits (001 for a checksum error, 002 for a record error) and the alarm is repeatedly sounded. To halt the alarm and return to the command mode, press the CANCEL key.

RECORD ERRORS

The following are typical causes of record errors.

- Attempting to load a file which is not a memory image. For example, loading an editor text file or a BASIC program file.
- Attempting to start a load in the middle of a load image. Therefore missing the initialization information at the start of the file.
- A tape error which causes a portion of the load image to be missed so the next record read is not in the proper sequence.

CHECKSUM ERRORS

A checksum error is flagged when the CRC (Cyclical Redundancy Check) checksum following a record does not match the CRC calculated by PAM-8. This error means that the record is either incorrectly recorded or the load is faulty. In either case, the load should be attempted again. If successive loads result in repeated failures, the original tape must be suspected as faulty.



I/O FACILITIES

PAM-8 supports two commands that allow you to perform input and output functions on H8 I/O ports. These front panel instructions permit simple manipulation of the H8 I/O ports without your having to write extensive routines to perform these functions.

Inputting From a Port

To input from a port, press the # key. Then enter three zero digits and the three-digit address (octal) of the desired port. NOTE: The front panel should now display 000 AAA, where AAA is the port address and 000 is meaningless. Press the IN key to read the port, the value is displayed in the three left-most digits of the front panel display.

Outputting to a Port

To output to a specified port, press the # key. Then enter the value to be supplied to the port in the three left-most displays. The port address is entered into the middle three displays. The display is of the form VVV AAA, where V stands for value, and A for address. Pressing the OUT key causes the value to be outputted to the indicated port.

Addressing Port Pairs

Frequently, ports are assigned in pairs, where one of the two port addresses is the control and status register and the other port is the data port. Address port pairs by using the + and - key to change ports. Once the initial port has been defined, the + key increments the port address to a new higher numbered port, and the - key is used to decrement to a lower numbered port.



ADVANCED CONTROL

One of the advanced features of PAM-8 is its provisions allowing sophisticated users to augment or replace PAM-8's functions. Augmenting or replacing PAM-8 functions is usually done in conjunction with assembly language programs, although it is possible to use some of these features by using BASIC's POKE and PEEK commands. The following discussion refers to symbols and locations defined in the PAM-8 program listing, given in its complete form as "Appendix A." It is recommended that you review the PAM-8 listing in order to become familiar with its various features. This can be done in conjunction with reading the following section, or independently. In either case, a first overview followed by a detailed analysis of the listing is probably necessary for a complete understanding.

16-Bit Tick Counter (TICCNT)

PAM-8 maintains a 16-bit (2 byte) tick counter known as TICCNT. The value of this counter is incremented each time a clock interrupt is processed. As an interrupt occurs once every 2 mS, the counter is incremented once every 2 mS. As long as clock interrupts are not disabled, this value can be used by any program to compute elapsed time. The tick counter may be set to any desired value, but it should not be frequently reset, as this interferes with the front panel refresh cycle. The contents of the tick counter are contained in memory locations 040 033 (the least significant byte) and 040 034 (the most significant byte).

Using the Keypad

When the user program is running, PAM-8 does not recognize any single key command. Thus, all single key patterns are available for the user program. To read keypad patterns, you can use one of two routines. First, you may take an input from port IP. PAD; or second, your program may use PAM-8's RCK routine. The input port IP. PAD is permanently assigned to port location 360. Inputting a binary number from this port detects which of the 16 keys are depressed. These results are shown in the table on Page 1-57 of "Appendix A."

A far more sophisticated keypad routine is available to you in the RCK (read Console Keypad) routine. This is also described in "Appendix A" (see Page 1-57). RCK provides keypad decoding, keypad debounce routines, auto-repeat routines, and acoustical feedback.

NOTE: If you use two key combinations, each key must reside in a separate bank. The first bank includes keys 0-7 and the second bank includes keys 8-#. RCK cannot decode two key combinations.



Display Usage

When a user program is running, PAM-8 normally displays the contents of the selected register or memory location. However, you may disable this process and display any arbitrary segment pattern, or completely disable the display to provide greater computational through-put. The display usage is primarily controlled by setting various bits in the .MFLAG memory cell. This memory cell is found at location 040 010. An explanation of the user option bits (UO.XXX) are found in "Appendix A" (see Page 1-29).

MANUAL UPDATING

By setting the UO.DDU (User Option. Disable Display Update) bit in the .MFLAG memory location, you can instruct PAM-8 to continue refreshing the front panel displays but to disable updating of their contents. When this is done, PAM-8 continues to refresh the 9 displays from a 9-byte block of RAM cells called FPLEDS. A description of FPLEDS is found in "Appendix A" (see Page 1-61). When the UO.DDU bit is set in .MFLAG, the contents of these bytes are not altered in any manner by PAM-8. The user program may then put any desired value into these bytes, thereby causing the front panel LED segments to light in the corresponding pattern.

You can use this technique to display numbers, letters, or arbitrary bar patterns on the front panel displays.

MANUAL DISPLAY REFRESHING

By setting the UO.NFR (User Option. No Front Panel Refresh) bit in the .MFLAG memory cell, you can instruct PAM-8 to stop refreshing the front panel displays. Setting the UO.NFR bit does not disable the clock interrupts; therefore, the tick counter (TICCNT) is still incremented. But PAM-8 does not refresh the displays from the information contained in the bytes FPLEDS.

If you desire, you may write a program to refresh the front panel LED displays. Usually this is done using the clock interrupts. If you undertake an independant front panel refresh program, take extreme care to avoid burning the displays due to excessive refreshing. **The total power dissipated in the LEDs is determined by the refresh cycle, and too frequent refreshing will result in excessive display heating.**



Using Interrupts

All H8 interrupts cause control to be transferred into the low 64 bytes of memory. PAM-8 occupies this memory space so all interrupts are first processed by PAM-8. Except for level zero interrupts, which are used as master clears, you can supply an interrupt processing routine for each of the seven additional interrupts. The following sections explain the use of each of these interrupts.

I/O INTERRUPTS

Interrupts numbered 3 through 7 are I/O interrupts. PAM-8 does not process these interrupts in any way. When a level 3 through level 7 interrupt is received, PAM-8 immediately transfers to the user interrupt vectors contained in memory locations 040 037 through 040 064. These locations are listed in "Appendix A" (see Page 1-61). Each location must contain a jump instruction pointing to the appropriate program location which processes these interrupts. NOTE: If any of these interrupts occur, you must supply a processing routine for them. This routine must be complete including both entry and exit processing.

CLOCK INTERRUPTS

The level one interrupts are generated by the front panel hardware every 2 mS. PAM-8 normally processes these interrupts. However, by setting a processing vector in UIVEC and setting the UO.INT bit in the MFLAG cell, PAM-8 enters the users routine each time a lock interrupt is generated. "Appendix A" (see Page 1-31) gives the required entry and exit conditions for processing clock interrupts.

SINGLE INSTRUCTION AND BREAKPOINT INTERRUPTS

Level two interrupts are generated by the single instruction hardware contained on the CPU card. When a single instruction is requested, the result of the interrupt is processed by PAM-8. If the single instruction interrupt was generated by PAM-8 in response to a Monitor Mode Single Instruction register condition, PAM-8 processes it. Otherwise, PAM-8 jumps to the user level two interrupt vector (UIVEC). Since the level two interrupt does not affect PAM-8, a level two restart instruction can be used as a breakpoint instruction by the user programs.



APPENDIX A

This appendix contains a complete listing of the PAM-8 front panel monitor program. PAM-8 resides in the low 1,024 bytes of the H8 computer. It provides all the control for front panel operation, and cassette or paper tape load and dump facilities. It also provides for master clear and front panel interrupt processing. PAM-8 presumes RAM cells are available for its use in locations 040 000 through 040 077 and 80 bytes are available in high memory for a stack. The use of these RAM cells is described on Page 1-61 of this Appendix and in the memory map on Page 0-50.

Pages 1-62, 1-63, and 1-64 of this Appendix are a symbolic reference table. Use this table to find the program locations where each symbolic address is used. Symbolic addresses are listed in alphabetical sequence.

FAM/8 - H8 FRONT PANEL MONITOR \$01.00.00.
INTRODUCTION.

HEATH X8ASM V1.1 06/21/77
15:43:50 01-AFR-77 PAGE 1

```
4 *** FAM/8 - H8 FRONT PANEL MONITOR.
5 *
6 * JGL, 05/01/76.
7 *
8 * FOR *WINTER* INC.
9 *
10 * COPYRIGHT 05/1976, WINTER CORPORATION,
11 * 902 N. 9TH ST.
12 * LAFAYETTE, INDI.
```

```
14 *** FAM/8 - H8 FRONT PANEL MONITOR.
15 *
16 * THIS PROGRAM RESIDES (IN ROM) IN THE LOW 1024 BYTES OF THE HEATH
17 * H8 COMPUTER. IT ACTUALLY CONSISTS OF TWO VIRTUALLY INDEPENDENT
18 * ROUTINES: A TASK-TIME PROGRAM WHICH PROVIDES SOPHISTICATED
19 * FRONT PANEL MONITOR SERVICE, AND AN INTERRUPT-TIME PROGRAM WHICH
20 * PROVIDES BOTH A REAL-TIME CLOCK AND EMULATES AN EFFECTIVE
21 * HARDWARE FRONT PANEL.
```

```
23 *** INTERRUPTS.
24 *
25 * FAM/8 IS THE PRIMARY PROCESSOR FOR ALL INTERRUPTS.
26 * THEY ARE PROCESSED AS FOLLOWS:
27 *
28 * RST USE
29 *
30 * 0 MASTER CLEAR. (NEVER USED FOR I/O OR RST)
31 *
32 * 1 CLOCK INTERRUPT. NORMALLY TAKEN BY FAM/8,
33 * SETTING BIT *U0.CLK* IN BYTE *.MFLAG* ALLOWS
34 * USER PROCESSING (VIA A JUMP THROUGH *UIVEC*).
35 * UPON ENTRY OF THE USER ROUTINE, THE STACK
36 * CONTAINS:
37 * (STACK+0) = RETURN ADDRESS (TO FAM/8)
38 * (STACK+2) = (STACKPTR+14)
39 * (STACK+4) = (AF)
40 * (STACK+6) = (BC)
41 * (STACK+8) = (DE)
42 * (STACK+10) = (HL)
43 * (STACK+12) = (PC)
44 * THE USER'S ROUTINE SHOULD RETURN TO FAM/8 VIA
45 * A *RETN* WITHOUT ENABLING INTERRUPTS.
46 *
47 * 2 SINGLE STEP. SINGLE STEP INTERRUPTS GENERATED
48 * BY FAM/8 ARE PROCESSED BY FAM/8.
49 * ANY SINGLE STEP INTERRUPT RECEIVED WHEN IN
50 * USER MODE CAUSES A JUMP THROUGH *UIVEC*+3.
51 * UPON USER ROUTINE ENTRY:
52 * (STACK+0) = (STACKPTR+12)
53 * (STACK+2) = (AF)
54 * (STACK+4) = (BC)
```

PAM/8 - M8 FRONT PANEL MONITOR \$01.00.00.
INTRODUCTION.

HEATH X8ASM V1.1 06/21/77
15:43:51 01-AFR-77 PAGE 2



```
55 *      (STACK+6) = (IE)
56 *      (STACK+8) = (HL)
57 *      (STACK+10) = (FC)
58 *      THE USER'S ROUTINE SHOULD HANDLE ITS OWN RETURN
59 *      FROM THE INTERRUPT.
60 *
61 *
62 *      THE FOLLOWING INTERRUPTS ARE VECTORED DIRECTLY THROUGH *UIVEC*.
63 *      THE USER ROUTINE MUST HAVE SETUP A JUMP IN *UIVEC* BEFORE ANY
64 *      OF THESE INTERRUPTS MAY OCCUR.
65 *
66 *      3      I/O 3. CAUSES A DIRECT JUMP THROUGH *UIVEC*+6
67 *
68 *      4      I/O 4. CAUSES A DIRECT JUMP THROUGH *UIVEC*+9
69 *      5      I/O 5. CAUSES A DIRECT JUMP THROUGH *UIVEC*+12
70 *
71 *      6      I/O 6. CAUSES A DIRECT JUMP THROUGH *UIVEC*+15
72 *
73 *      7      I/O 7. CAUSES A DIRECT JUMP THROUGH *UIVEC*+18
```

PAM/8 - HB FRONT PANEL MONITOR \$01.00.00.
ASSEMBLY CONSTANTS.

HEATH X8ASM V1.1 06/21/77
15:43:52 01-AFR-77 PAGE 3

77 ** ASSEMBLY CONSTANTS

79 ** I/O PORTS

000.360	81	IF.FAI	EQU	360Q	FAI INPUT PORT
000.360	82	OP.CTL	EQU	360Q	CONTROL OUTPUT PORT
000.360	83	OP.DIG	EQU	360Q	DIGIT SELECT OUTPUT PORT
000.361	84	OP.SEG	EQU	361Q	SEGMENT SELECT OUTPUT PORT
000.371	85	IF.TFC	EQU	371Q	TAPE CONTROL IN
000.371	86	OP.TFC	EQU	371Q	TAPE CONTROL OUT
000.370	87	IF.TFI	EQU	370Q	TAPE DATA IN
000.370	88	OP.TFI	EQU	370Q	TAPE DATA OUT

90 ** ASCII CHARACTERS.

000.026	92	A.SYN	EQU	026Q	SYNC CHARACTER
000.002	93	A.STX	EQU	002Q	STX CHARACTER

95 ** FRONT PANEL HARDWARE CONTROL BITS.

000.020	97	CR.SSI	EQU	00010000B	SINGLE STEP INTERRUPT
000.040	98	CR.MTL	EQU	00100000B	MONITOR LIGHT
000.100	99	CR.CLI	EQU	01000000B	CLOCK INTERRUPT ENABLE
000.200	100	CR.SPK	EQU	10000000B	SPEAKER ENABLE

102 ** DISPLAY MODE FLAGS (IN *DISPMOD*)

000.000	104	DM.MR	EQU	0	MEMORY READ
000.001	105	DM.MW	EQU	1	MEMORY WRITE
000.002	106	DM.RR	EQU	2	REGISTER READ
000.003	107	DM.RW	EQU	3	REGISTER WRITE
000.000	108	XTEXT	TAPE		TAPE DEFINITIONS

110X ** TAPE EQUIVALENCES.

000.001	112X	RT.MI	EQU	1	RECORD TYPE - MEMORY DUMP IMAGE
000.002	113X	RT.BP	EQU	2	RECORD TYPE - BASIC PROGRAM
000.003	114X	RT.CT	EQU	3	RECORD TYPE - COMPRESSED TEXT

115X
116X ** BLOCK SIZE FOR INTER-PRODUCT COMMUNICATION.

002.000	117X	BLNSIZ	EQU	512	
	118X				
	119X				



FAM/B - H8 FRONT PANEL MONITOR \$01.00.00.
ASSEMBLY CONSTANTS.

HEATH X8ASM V1.1 06/21/77
15:43:56 01-APR-77 PAGE 4



121 ** MACHINE INSTRUCTIONS.
122
000.166 123 MI.HLT EQU 01110110B HALT
000.311 124 MI.RET EQU 11001001B RETURN
000.333 125 MI.IN EQU 11011011B INPUT
000.323 126 MI.OUT EQU 11010011B OUTPUT
000.072 127 MI.LIA EQU 00111010B LIA
000.346 128 MI.ANI EQU 11100110B ANI
000.021 129 MI.LXIN EQU 00010001B LX1 II

131 ** USER OPTION BITS.

132 *
133 * THESE BITS ARE SET IN CELL .MFLAG.

134
000.200 135 UO.HLT EQU 10000000B DISABLE HALT PROCESSING
000.100 136 UO.NFR EQU CB.CLI NO REFRESH OF FRONT PANEL
000.002 137 UO.DDU EQU 00000010B DISABLE DISPLAY UPDATE
000.001 138 UO.CLN EQU 00000001B ALLOW CLOCK INTERRUPT PROCESSING

000.000 140 XTEXT U8251 DEFINE 8251 USART BITS

FAM/8 - HB FRONT PANEL MONITOR \$01.00.00.
8251 USART BIT DEFINITIONS.

HEATH X8ASM V1.0 02/18/77
13:23:23 01-APR-77 PAGE 5

	143X **	8251 USART BIT DEFINITIONS.
	144X *	
	145X	
	146X **	MODE INSTRUCTION CONTROL BITS:
	147X	
000.100	148X UMI.1B EQU	01000000B 1 STOP BIT
000.200	149X UMI.HB EQU	10000000B 1 1/2 STOP BITS
000.300	150X UMI.2B EQU	11000000B 2 STOP BITS
000.040	151X UMI.FE EQU	00100000B EVEN PARITY
000.020	152X UMI.FA EQU	00010000B USE PARITY
000.000	153X UMI.L5 EQU	00000000B 5 BIT CHARACTERS
000.004	154X UMI.L6 EQU	00000100B 6 BIT CHARACTERS
000.010	155X UMI.L7 EQU	00001000B 7 BIT CHARACTERS
000.014	156X UMI.L8 EQU	00001100B 8 BIT CHARACTERS
000.001	157X UMI.1X EQU	00000001B CLOCK X 1
000.002	158X UMI.16X EQU	00000010B CLOCK X 16
000.003	159X UMI.64X EQU	00000011B CLOCK X 64
	160X	
	161X **	COMMAND INSTRUCTION BITS.
	162X	
000.100	163X UCI.IR EQU	01000000B INTERNAL RESET
000.040	164X UCI.RD EQU	00100000B READER-ON CONTROL FLAG
000.020	165X UCI.ER EQU	00010000B ERROR RESET
000.004	166X UCI.RE EQU	00000100B RECEIVE ENABLE
000.002	167X UCI.IE EQU	00000010B ENABLE INTERRUPTS FLAG
000.001	168X UCI.TE EQU	00000001B TRANSMIT ENABLE
	169X	
	170X **	STATUS READ COMMAND BITS.
	171X	
000.040	172X USR.FE EQU	00100000B FRAMING ERROR
000.020	173X USR.OE EQU	00010000B OVERRUN ERROR
000.010	174X USR.FE EQU	00001000B PARITY ERROR
000.004	175X USR.TXE EQU	00000100B TRANSMITTER EMPTY
000.002	176X USR.RXR EQU	00000010B RECEIVER READY
000.001	177X USR.TXR EQU	00000001B TRANSMITTER READY

PAM/8 - H8 FRONT PANEL MONITOR #01.00.00.
HARDWARE INTERRUPT VECTORS

HEATH X8ASM V1.0 02/18/77
13:23:25 01-APR-77 PAGE 6



180 *** INTERRUPT VECTORS.
181 *
182

184 ** LEVEL 0 - RESET
185 *
186 * THIS INTERRUPT MAY NOT BE PROCESSED BY A USER PROGRAM.
187

000.000 188 ORG .00A
189
000.000 021 371 003 190 INIT0 LXI I,FRSRDM (DE) = ROM COPY OF FRS CODE
000.003 041 012 040 191 LXI H,FRSRAM+FRSL-1 (HL) = RAM DESTINATION FOR CODE
000.006 303 073 000 192 JMP INIT INITIALIZE
377.073 193 ERRPL INIT-1000A BYTE IN WORD 10A MUST BE 0

195 ** LEVEL 1 - CLOCK
196
000.010 197 INT1 EQU 100 INTERRUPT ENTRY POINT
198
000.000 199 ERRNZ *-110 INTO TAKES UP ONE BYTE
000.011 315 132 000 200 CALL SAVALL SAVE USER REGISTERS
000.014 026 000 201 MVI D,O
000.016 303 201 000 202 JMP CLOCK PROCESS CLOCK INTERRUPT
377.201 203 ERRPL CLOCK-1000A EXTRA BYTE MUST BE 0

205 ** LEVEL 2 - SINGLE STEP
206 *
207 * IF THIS INTERRUPT IS RECEIVED WHEN NOT IN MONITOR MODE.
208 * THEN IT IS ASSUMED TO BE GENERATED BY A USER PROGRAM
209 * (SINGLE STEPPING OR BREAKPOINTING). IN SUCH CASE, THE
210 * USER PROGRAM IS ENTERED THROUGH (UIVEC+3)
211

000.020 212 INT2 EQU 20A LEVEL 2 ENTRY
213
000.000 214 ERRNZ *-21A INT1 TAKES EXTRA BYTE
000.021 315 132 000 215 CALL SAVALL SAVE REGISTERS
000.024 032 216 LDAX D (A) = (CTLFLG)
040.011 217 SET CTLFLG
000.025 303 244 001 218 JMP STPRTN STEP RETURN

220 *** I/O INTERRUPT VECTORS.
221 *
222 * INTERRUPTS 3 THROUGH 7 ARE AVAILABLE FOR GENERAL I/O USE.
223 *
224 * THESE INTERRUPTS ARE NOT SUPPORTED BY PAM/8, AND SHOULD
225 * NEVER OCCUR UNLESS THE USER HAS SUPPLIED HANDLER ROUTINES
226 * (THROUGH UIVEC)
227

PAM/8 - HB FRONT PANEL MONITOR \$01.00.00.
HARDWARE INTERRUPT VECTORS

HEATH XASM V1.0 02/18/77
13:23:26 01-AFR-77 PAGE 7

000.030 228 ORG 30A
000.030 303 045 040 229 INT3 JMP UIVEC+6 JUMP TO USER ROUTINE
000.033 064 064 064 231 DB 44413 HEATH PART NUMBER 444-13

000.040 233 ORG 40A
000.040 303 050 040 234 INT4 JMP UIVEC+9 JUMP TO USER ROUTINE
000.043 100 112 107 236 DB 100Q,112Q,107Q,114Q,100Q SUPPORT CODE

000.050 238 ORG 50A
000.050 303 053 040 239 INT5 JMP UIVEC+12 JUMP TO USER ROUTINE

240
241
242 ** DLY - DELAY TIME INTERVAL.
243 * ENTRY (A) = MILLISECOND DELAY COUNT/2
244 * EXIT NONE
245 * USES A,F
247
000.053 365 248 DLY PUSH FSW SAVE COUNT
000.054 257 249 XRA A DONT SOUND HORN
000.055 303 143 002 250 JMP HRNO PROCESS AS HORN

000.060 252 ORG 60A
000.060 303 056 040 253 INT6 JMP UIVEC+15 JUMP TO USER ROUTINE

254
255
000.063 076 320 256 GO, MVI A,CB,SSI+CB,CLI+CB,SPK OFF MONITOR MODE LIGHT
000.065 303 235 001 257 JMP SST1 RETURN TO USER PROGRAM

000.070 259 ORG 70A
000.070 303 061 040 260 INT7 JMP UIVEC+18 JUMP TO USER ROUTINE



FAM/8 - H8 FRONT PANEL MONITOR \$01.00.00.
MASTER CLEAR PROCESSING

HEATH X8ASM V1.0 02/18/77
13:23:28 01-APR-77 PAGE 8



```
263 ** INIT - INITIALIZE SYSTEM
264 *
265 * INIT IS CALLED WHENEVER A HARDWARE MASTER-CLEAR IS INITIATED.
266 *
267 * SETUP FAM/8 CONTROL CELLS IN RAM.
268 * DECODE HOW MUCH MEMORY EXISTS, SETUP STACKPOINTER, AND
269 * ENTER THE MONITOR LOOP.
270 *
271 * ENTRY FROM MASTER CLEAR
272 * EXIT INTO FAM/8 MAIN LOOP
273
274
000.073 032 275 INIT LDAX D COPY *FRSRUM* INTO RAM
000.074 167 276 MOV M,A MOVE BYTE
000.075 053 277 DCX H DECREMENT DESTINATION
000.076 034 278 INR E INCREMENT SOURCE
000.077 302.073.000 279 JNZ INIT IF NOT DONE
280
.004.000 281 SINCR EQU 4000A SEARCH INCREMENT
282
000.102 .026.004 283 MVI D,SINCR/256 (DE) = SEARCH INCREMENT
000.104 041 000 034 284 LXI H,START-SINCR (HL) = FIRST RAM - SEARCH INCREMENT
285
286 * DETERMINE MEMORY LIMIT.
287
000.107 167 288 INIT1 MOV M,A RESTORE VALUE READ
000.110 031 289 IAD D INCREMENT TRIAL ADDRESS
000.111 176 290 MOV A,M (A) = CURRENT MEMORY VALUE
000.112 065 291 ICR M TRY TO CHANGE IT
000.113 276 292 CMP M
000.114 302 107 000 293 JNE INIT1 IF MEMORY CHANGED
294
000.117 053 295 INIT2 DCX H
000.120 371 296 SFHL SET STACKPOINTER = MEMORY LIMIT -1
000.121 345 297 PUSH H SET *FC* VALUE ON STACK
000.122 041 322 000 298 LXI H,ERROR
000.125 345 299 PUSH H SET 'RETURN ADDRESS'
300
301 * CONFIGURE LOAD/DUMP UART
302
000.126 076 116 303 MVI A,UMI.18+UMI.L8+UMI.16X
000.130 323 371 304 OUT OF.TFC SET 8 BIT, NO PARITY, 1 STOP, x16
```

PAM/B - H8 FRONT PANEL MONITOR \$01.00.00.
INTERRUPT TIME SUBROUTINES

HEATH X8ASM V1.0 02/18/77
13:23:29 01-AFR-77 PAGE 9

```

307 ** SAVALL -- SAVE ALL REGISTERS ON STACK.
308 *
309 * SAVALL IS CALLED WHEN AN INTERRUPT IS ACCEPTED, IN ORDER TO
310 * SAVE THE CONTENTS OF THE REGISTERS ON THE STACK.
311 *
312 * ENTRY CALLED DIRECTLY FROM INTERRUPT ROUTINE.
313 * EXIT ALL REGISTERS PUSHED ON STACK,
314 * IF NOT YET IN MONITOR MODE, REGPTR = ADDRESS OF REGISTERS
315 * ON STACK.
316 * (DE) = ADDRESS OF CTLFLG
317
318
000.132 343 319 SAVALL XTHL SET H,L ON STACK TOP
000.133 325 320 PUSH D
000.134 305 321 PUSH R
000.135 365 322 PUSH FSW
000.136 353 323 XCHG (D,E) = RETURN ADDRESS
000.137 041 012 000 324 LXI H,10
000.142 071 325 DAD SF (H,L) = ADDRESS OF USERS SP
000.143 345 326 PUSH H SET ON STACK AS 'REGISTER'
000.144 325 327 PUSH D SET RETURN ADDRESS
000.145 021 011 040 328 LXI D,CTLFLG
000.150 032 329 LIAX D (A) = CTLFLG
000.151 057 330 CMA
000.152 346 060 331 ANI CR.MTL+CR.SSI SAVE REGISTER ADDR IF USER OR SINGLE-STEP
000.154 310 332 RZ RETURN IF WAS INTERRUPT OF MONITOR LOOP
000.155 041 002 000 333 LXI H,2
000.160 071 334 DAD SF (H,L) = ADDRESS OF 'STACKPTR' ON STACK
000.161 042 035 040 335 SHLD REGPTR
000.164 311 336 RET

```

```

338 ** CUI -- CHECK FOR USER INTERRUPT PROCESSING.
339 *
340 * CUI IS CALLED TO SEE IF THE USER HAS SPECIFIED PROCESSING
341 * FOR THE CLOCK INTERRUPT.
342
343
040.010 344 + SET .MFLAG REFERENCE TO MFLAG
000.165 012 345 LIAX B (A) = .MFLAG
000.000 346 ERRNZ U0.CLK-1 CORE ASSUMED = 01
000.166 017 .
000.167 334 037 040 347 KRC
348 CC UIVEC IF SPECIFIED, TRANSFER TO USER
349
350 * RETURN TO PROGRAM FROM INTERRUPT.
351
000.172 361 352 INTXIT POP PSW REMOVE FAKE 'STACK REGISTER'
000.173 361 353 POP PSW
000.174 301 354 POP B
000.175 321 355 POP D
000.176 341 356 POP H
000.177 373 357 EI
000.200 311 358 RET

```



 HEATHKIT

```

361 *** CLOCK - PROCESS CLOCK INTERRUPT
362 *
363 * CLOCK IS ENTERED WHENEVER A MILLISECOND CLOCK INTERRUPT IS
364 * PROCESSED.
365 *
366 * TICCNT IS INCREMENTED EVERY INTERRUPT.
367
368
000.201 052 033 040 369 CLOCK LHLD TICCNT
000.204 043 370 INX H
000.205 042 033 040 371 SHLD TICCNT INCREMENT TICCOUNT
372
373 ** REFRESH FRONT PANEL.
374 *
375 * THIS CODE DISPLAYS THE APPROPRIATE PATTERN ON THE
376 * FRONT PANEL LENS. THE LENS ARE PAINTED IN REVERSE ORDER,
377 * ONE PER INTERRUPT. FIRST, NUMBER 9 IS LIT, THEN NUMBER 8,
378 * ETC.
379
380
000.210 041 010 040 381 LXI H,MFLAG
000.213 176 382 MOV A,M
000.214 107 383 MOV B,A (B) = CURRENT FLAG
000.215 346 100 384 ANI UD,NFK SEE IF FRONT PANEL REFRESH WANTED
000.217 043 385 INX H
000.000 386 ERRNZ CTLFLG-.MFLAG-1
000.220 176 387 MOV A,M (A) = CTLFLG
000.221 112 388 MOV C,D (C) = 0 IN CASE NO PANEL DISPLAY
000.222 302 237 000 389 JNZ CLK3 IF NOT
000.225 043 390 INX H (H,L) = (REFIND)
000.000 391 ERRNZ REFIND-CTLFLG-1
000.226 065 392 ICR M DECREMENT DIGIT INDEX
000.227 302 234 000 393 JNZ CLK2 IF NOT WRAP-AROUND
000.232 066 011 394 MVI M,9 WRAP DISPLAY AROUND
000.234 136 395 CLK2 MOV E,M
000.235 031 396 DAD D (H,L) = ADDRESS OF PATTERN
000.236 113 397 MOV C,E
000.237 261 398 CLK3 EQU * (A) = CTLNLG
000.237 323 360 399 ORA C (A) = INDEX + FIXED BITS
000.240 323 360 400 OUT OF,DIG SELECT DIGIT
000.242 176 401 MOV A,M
000.243 323 361 402 OUT OF,SEG SELECT SEGMENT
403
404 * SEE IF TIME TO RECOME DISPLAY VALUES.
405
000.245 056 033 406 MVI L,$TICCNT
000.247 176 407 MOV A,M
000.250 346 037 408 ANI 370 EVERY 32 INTERRUPTS
000.252 314 161 003 409 CZ UFI UPDATE FRONT PANEL DISPLAYS
410
411 * EXIT CLOCK INTERRUPT.
412
000.255 001 011 040 413 LXJ B,CTLFLG
000.260 012 414 LIAX B (A) = CTLFLG
000.261 346 040 415 ANI CB,MTL
000.263 302 172 000 416 JNZ INTXIT IF IN MONITOR MODE

```

FAM/8 - H8 FRONT FANEL MONITOR \$01.00.00.
PROCESS CLOCK INTERRUPTS

HEATH x8ASM V1.0 02/18/77
13:23:34 01-APR-77 PAGE 11

```

000.266 013      417    DCX    B
000.000          418    ERRNZ  CTLFLG-.MFLAG-1
000.267 012      419    LDAX   B           (A),=.,MFLAG
000.000          420    ERRNZ  UD.HLT-2000 ASSUME HIGH-ORDER
000.270 027      421    RAL
000.271 332 313 000 422    JC     CLK4      SKIP IT
                            423
                            424 * NOT IN MONITOR MODE. CHECK FOR HALT
                            425
000.274 076 012      426    MVI    A,10      (A) = INDEX OF *F* REG
000.276 315 052 003  427    CALL   LRA:      LOCATE REGISTER ADDRESS
000.301 136          428    MOV    E,M
000.302 043          429    INX    H
000.303 126          430    MOV    D,M      (D,E) = FC CONTENTS
000.304 033          431    DCX   B
000.305 032          432    LDAX   B
000.306 376 166      433    CPI    MI.HLT      CHECK FOR HALT
000.310 312 322 000  434    JE     ERROR      IF HALT, BE IN MONITOR MODE
                            435
                            436 * CHECK FOR 'RETURN TO MONITOR' KEY ENTRY.
                            437
000.313          438    CLK4    EQU    *
000.313 333 360      439    IN     IP,PAI
000.315 376 056      440    CPI    560      SEE IF '0' AND '#'
000.317 302 165 000  441    JNE    CUI1      IF NOT, ALLOW USER PROCESSING OF CLOCK

```



HEATHKIT

PAM/8 - H8 FRONT PANEL MONITOR \$01.00.00.
MTR - MAIN EXECUTIVE LOOP.

HEATH X8ASM V1.1 06/21/77
15:44:09 01-APR-77 PAGE 12

HEATHKIT

Panel Monitor

445 *** ERROR - COMMAND ERROR.
446 *
447 * ERROR IS CALLED AS A 'BAIL-OUT' ROUTINE.
448 *
449 * IT RESETS THE OPERATIONAL MODE, AND RESTORES THE STACKPOINTER.
450 *
451 * ENTRY NONE
452 * EXIT TO MTR LOOP
453 * CTLFLG SET
454 * ,MFLAG CLEARED
455 * USES ALL
456
457
000.322 458 ERROR EQU *
000.322 041 010 040 459 LXI H,.MFLAG
000.325 176 460 MOV A,M (A) = .MFLAG
000.326 346 275 461 ANI 3770-UU,DIU-UU,NFR RE-ENABLE DISPLAYS
000.330 187 462 MOV M,A REPLACE
000.331 043 463 INX H
000.332 068 360 464 MOVI H,LC,SSI+CB,MTL+CH,CLT+CB,SPK RESTORE *CTLFLG*
000.000 465 ERRNZ CTLFLG-.MFLAG-1
000.334 373 466 EI
000.335 052 035 040 467 LHLD REGPTR
000.340 371 468 SPHL RESTORE STACK POINTER TO EMPTY STATE
000.341 315 136 002 469 CALL ALARM ALARM FOR 200 MS

471 ** MTR - MONITOR LOOP.
472 *
473 * THIS IS THE MAIN EXECUTIVE LOOP FOR THE FRONT PANEL EMULATOR.
474
475
000.344 476 MTR EQU *
000.344 373 477 EI
478
000.345 041 345 000 479 MTR1 LXI H:MTR1
000.350 345 480 PUSH H SET 'MTR1' AS RETURN ADDRESS
000.351 001 007 040 481 LXI B,DISPMOD (BC) = #DISPMOD
000.354 012 482 LDAX B
000.355 346 001 483 ANI 1 (A) = 1 IF ALTER
000.357 057 484 CMA
000.360 062 006 040 485 STA DISPROT ROTATE LED PERIODS IF ALTER
486
487 * READ KEY
488
000.363 315 260 003 489 CALL RCK READ CONSOLE KEYPAD
000.366 052 024 040 490 LHLD ARUSS
000.371 376 012 491 CPI 10
000.373 322 005 001 492 JNC MTR4 IF IN 'ALWAYS VALID' GROUP
000.376 137 493 MOV E,A SAVE VALUE
040.007 494 SET DISPMOD
000.377 012 495 LDAX B (A) = DISPMOD
001.000 017 496 RRC
001.001 332 051 001 497 JC MTRS IF IN ALTER MODE

FAM/8 - H8 FRONT PANEL MONITOR \$01.00.00.
MTR - MAIN EXECUTIVE LOOP.

HEATH X8ASM V1.0 02/18/77
13:23:37 01-APR-77 PAGE 13

001.004	173	498	MOV	A,E	(A) = CODE
		499			
		500	*	HAVE A COMMAND (NOT A VALUE)	
		501			
001.005	326 004	502	MTR4	SUI	4 (A) = COMMAND
001.007	332 322 000	503		JC	ERROR IF BAD
001.012	137	504		MOV	E,A
001.013	345	505		PUSH	H SAVE ABUSS VALUE
001.014	041 035 001	506		LXI	H,MTRA
001.017	026 000	507		MVI	D,0
001.021	031	508		IAD	D (H,L) = ADDRESS OF TABLE ENTRY
001.022	136	509		MOV	E,H
001.023	031	510		IAD	D (H,L) = ADDRESS OF PROCESSOR
001.024	343	511		XTHL	SET ADDRESS, (H,L) = (ARUSS)
001.025	021 005 040	512		LXI	D,REGI (D,E) = ADDRESS OF REG. INDEX
040.007		513	.	SET	DSFMOD
001.030	012	514		LDAX	R (A) = DSFMOD
001.031	346 002	515		ANI	2 SET 'Z' IF MEMORY
001.033	012	516		LDAX	R (A) = DSFMOD
001.034	311	517		RET	JUMP TO PROCESSOR
		518			
		519			
001.035		520	MTRA	ERU	*
001.035	165	521	DR	GO-*	4 -- GO
001.036	141	522	DR	IN-*	5 -- INPUT
001.037	143	523	DR	OUT-*	6 -- OUTPUT
001.040	165	524	DR	SSTEP-*	7 -- SINGLE STEP
001.041	220	525	DR	RMEM-*	8 -- CASSETTE LOAD
001.042	332	526	DR	WMEM-*	9 -- CASSETTE DUMP
001.043	067	527	DR	NEXT-*	+ -- NEXT
001.044	104	528	DR	LAST-*	- -- LAST
001.045	102	529	DR	ABORT-*	* -- ABORT
001.046	060	530	DR	R&W-*	/ -- DISPLAY/ALTER
001.047	116	531	DR	MEMM-*	# -- MEMORY MODE
001.050	034	532	DR	REGM-*	. -- REGISTER MODE
		534	**	PROCESS 'MEMORY/REGISTER' ALTERATIONS.	
		535	*		
		536	*	THIS CODE IS ENTERED IF	
		537	*		
		538	*	1) AM IN ALTER MODE, AND	
		539	*	2) A KEY FROM 0-7 WAS ENTERED.	
		540			
001.051	017	541	MTR5	RRC	
001.052	173	542	MOV	A,E	(A) = VALUE
001.053	332 067 001	543	JC	MTR6	IS REGISTER
001.056	067	544	STC		INDICATE 1ST DIGIT IS IN (A)
001.057	315 066 003	545	CALL	I0B	INPUT OCTAL BYTE
001.062	043	546	INX	H	DISPLAY NEXT LOCATION

PAM/8 - H8 FRONT PANEL MONITOR \$01,00,00,
MTR - MAIN EXECUTIVE LOOP.

HEATH X86ASM V1.0 02/18/77
13:23:39 01-APR-77 PAGE 14

HEATHKIT

548 ** SAE - STORE ABUSS AND EXIT.
549 *
550 * ENTRY (HL) = ABUSS VALUE
551 * EXIT TO (RET)
552 * USES NONE
553
001.063 042 024 040 554 SAE SHLD ABUSS
001.066 311 555 RET
556
557 * ALTER REGISTER
558
001.067 365 559 MTR6 PUSH FSW SAVE CODE
001.070 315 047 003 560 CALL LRA LOCATE REGISTER ADDRESS
001.073 247 561 ANA A
001.074 312 322 000 562 JZ ERROR NOT ALLOWED TO ALTER STACKPOINTER
001.077 043 563 INX H
001.100 361 564 PUF FSW RESTORE VALUE AND CARRY FLAG
001.101 303 042 003 565 JMP IOA INPUT OCTAL ADDRESS

PAM/8 - H8 FRONT PANEL MONITOR \$01.00.00.
MONITOR TASK SUBROUTINES.

HEATH X8ASM.V1.1 06/21/77
15:44:14 01-APR-77 PAGE 15

```

569 ** REGM - ENTER REGISTER DISPLAY MODE.
570 *
571 * ENTRY (A) = DSFMDI
572 * (BC) = #DSFMDI
573
001.104 076 002 574 REGM MVI A,2 SET DISPLAY REGISTER MODE
040.007 575 SET DSFMDI
001.106 002 576 STAX B SET DISPLAY REGISTER MODE
000.000 577 ERRNZ DSFMDI-DSFRDT-1
001.107 013 578 ICX B (BC) = #DSFRDT
001.110 257 579 XRA A
001.111 002 580 STAX B SET ALL PERIODS ON
001.112 315 260 003 581 CALL RCK READ KEY ENTRY
001.115 075 582 ICR A DISPLACE
001.116 376 006 583 CPI 6
001.120 322 322 000 584 JNC ERROR NOT 1-6
001.123 007 585 RLC
001.124 022 586 STAX D SET NEW REG IND
040.005 587 SET REGI
001.125 311 588 RET

```

```

590 ** R$W - TOGGLE DISPLAY/ALTER MODE.
591 *
592 * ENTRY (A) = DSFMDI
593 * (BC) = ADDRESS OF DSFMDI
594
040.007 595 SET DSFMDI
001.126 356 001 596 R$W XR1 1
001.130 002 597 STAX B
001.131 311 598 RET

```

```

600 ** NEXT - INCREMENT DISPLAY ELEMENT.
601 *
602 * ENTRY (HL) = (ABUSS)
603 * (IE) = ADDRESS OF REGIND
604
001.132 043 605 NEXT INX H
001.133 312 063 001 606 JZ SAE IF MEMORY, STORE ABUSS AND EXIT
607
608 * IS REGISTER MODE.
609
040.005 610 SET REGI
001.136 032 611 LDAX D (A) = REGI
001.137 306 002 612 ADI 2 INCREMENT REG INNEX
001.141 022 613 STAX D WRAP TO *SF*
001.142 376 014 614 CPI 12
001.144 330 615 RC IF NOT TOO LARGE, EXIT
001.145 257 616 XRA A OVERFLOW
001.146 022 617 STAX D
001.147 311 618 ABORT RET

```



```

620 ** LAST - DECREMENT DISPLAY ELEMENT.
621 *
622 * ENTRY (HL) = (ABUSS)
623 * (DE) = ADDRESS OF REGINI
624
001.150 053 625 LAST IX H
001.151 312 063 001 626 JZ SAE IF MEMORY, STORE AND EXIT
627
628 * IS REGISTER MODE.
629
040.005 630 : SET REGI
001.154 032 631 LS12 LDAX I (A) = REGI
001.155 326 002 632 SUI 2
001.157 022 633 STAX I
001.160 320 634 RNC IF OK
001.161 076 012 635 MVI A,10 UNDERFLOW TO *PC*
001.163 022 636 STAX I
001.164 311 637 RET
638

```

```

640 ** MEMM - ENTER DISPLAY MEMORY MODE.
641 *
642 * ENTRY (BC) = ADDRESS OF DISPMOD
643
001.165 257 644 MEMM XRA A (A) = 0
040.007 645 . SET DISPMOD
001.166 002 646 STAX B SET DISPLAY MEMORY MODE
000.000 647 ERRNZ DISPMOD-DISPROT-1
001.167 013 648 DCX B (BC) = #DISPROT
001.170 002 649 STAX B SET ALL PERIODS ON
001.171 041 025 040 650 LXI H,ABUSS+1
001.174 303 062 003 651 JMP IOA INPUT OCTAL ADDRESS

```

```

653 ** IN - INPUT DATA BYTE.
654 *
655
656 ** OUT - OUTPUT DATA BYTE.
657 *
658 * ENTRY (HL) = (ABUSS)
659
001.177 006 333 660 IN MVI B,MI.IN
001.201 021 661 DB MI,LXII SKIP NEXT INSTRUCTION
001.202 006 323 662 OUT MVI B,MI.OUT
001.204 174 663 MOV A,H (A) = VALUE
001.205 145 664 MOV H,L (H) = PORT
001.206 150 665 MOV L,B (L) = IN/OUT INSTRUCTION
001.207 042 002 040 666 SHLD IOWRK
001.212 312 002 040 667 CALL IOWRK PERFORM IO
001.215 154 668 MOV L,H (L) = PORT
001.216 147 669 MOV H,A (H) = VALUE
001.217 303 063 001 670 JMP SAE STORE ABUSS AND EXIT

```

PAM/8 - H8 FRONT PANEL MONITOR \$01.00.00.
 GO AND *STEP* FUNCTIONS

HEATH X8ASM V1.0 02/18/77
 13:23:43 01-AFR-77 PAGE 18

675 ** GO - RETURN TO USER MODE
 676 *
 677 * ENTRY NONE
 678
 001.222 303 063 000 679 60 JMF 60. ROUTINE IS IN WASTE SPACE

681 ** SSTEP - SINGLE STEP INSTRUCTION.
 682 *
 683 * ENTRY NONE
 684
 001.225 685 SSTEP EQU * SINGLE STEP
 001.225 363 686 ORI CR.SSI DISABLE INTERRUPTS UNTIL THE RIGHT TIME
 001.226 072 011 040 687 LDA CTLFLG
 001.231 356 020 688 XRI CR.SSI CLEAR SINGLE STEP INHIBIT
 001.233 323 360 689 OUT OF.CTL FRAME SINGLE STEP INTERRUPT
 001.235 062 011 040 690 SST1 STA CTLFLG SET NEW FLAG VALUES
 001.240 341 691 POF H CLEAN STACK
 001.241 303 172 000 692 JMF INTXIT RETURN TO USER ROUTINE FOR STEP

694 ** STPRTN - SINGLE STEP RETURN
 695
 001.244 696 STPRTN EQU *
 001.244 366 020 697 ORI CR.SSI DISABLE SINGLE STEP INTERRUPTION
 001.246 323 360 698 OUT OF.CTL TURN OFF SINGLE STEP ENABLE
 040.011 699 SET CTLFLG
 001.250 022 700 STAX I
 001.251 346 040 701 ANI CR.MTL SEE IF IN MONITOR MODE
 001.253 302 344 000 702 JNZ MTR
 001.256 303 042 040 703 JMF UIVEC+3 TRANSFER TO USER'S ROUTINE

705 ** RMEM - LOAD MEMORY FROM TAPE.
 706 *
 707
 001.261 041 244 002 708 RMEM LXI H,TFAPI
 001.264 042 031 040 709 SHLD TPERRX SETUP ERROR EXIT ADDRESS
 710 * JMF LOAD



PAM/8 - H8 FRONT PANEL MONITOR \$01.00.00.
LOAD = LOAD MEMORY FROM TAPE

HEATH X8ASM V1.1 06/21/77
15:44:19 01-APR-77 PAGE 19



712 *** LOAD - LOAD MEMORY FROM TAPE.
713 *
714 * READ THE NEXT RECORD FROM THE CASSETTE TAPE.
715 *
716 * USE THE LOAD ADDRESS IN THE TAPE RECORD.
717 *
718 * ENTRY (HL) = ERROR EXIT ADDRESS
719 * EXIT USER P-REG (IN STACK) SET TO ENTRY ADDRESS
720 * TO CALLER IF ALL OK
721 * TO ERROR EXIT IF TAPE ERRORS DETECTED.
722
723
001.267 724 LOAI EQU *
001.267 001.000.376 725 LXI B,1000A-RT.MI*256-256 (BC) = - REQUIRED TYPE AND #
001.272 315 265 002 726 LOAO CALL SRS SCAN FOR RECORD START
001.275 157 727 MOV L,A (HL) = COUNT
001.276 353 728 XCHG (DE) = COUNT, (HL) = TYPE AND #
001.277 015 729 DCR C (C) = - NEXT #
001.300 011 730 DAD B
001.301 174 731 MOV A,H SAVE TYPE AND #
001.302 305 732 PUSH B
001.303 365 733 FUSH PSW SAVE TYPE CODE
001.304 346 177 734 ANI 1770 CLEAR END FLAG BIT
001.306 265 735 ORA L
001.307 076 002 736 MVI A,2 SEQUENCE ERROR
001.311 302 205 002 737 JNE TPIERR IF NOT RIGHT TYPE OR SEQUENCE
001.314 315 325 002 738 CALL RNP READ ADDR
001.317 104 739 MOV B,H
001.320 117 740 MOV C,A (BC) = P-REG ADDRESS
001.321 076 012 741 MVI A,10
001.323 325 742 PUSH D SAVE (DE)
001.324 315 052 003 743 CALL LRA LOCATE REG ADDRESS
001.327 321 744 PUP D RESTORE (DE)
001.330 161 745 MOV M,C SET P-REG IN MEM
001.331 043 746 INX H
001.332 160 747 MOV M,B
001.333 315 325 002 748 CALL RNP READ ADDRESS
001.336 157 749 MOV L,A (HL) = ADDRESS, (DE) = COUNT
001.337 042 000 040 750 SHLD START
751
001.342 315 331 002 752 LOAI CALL RNB READ BYTE
001.343 167 753 MOV M,A
001.346 042 024 040 754 SHLD ABUSS SET ABUSS FOR DISPLAY
001.351 043 755 INX H
001.352 033 756 DCR D
001.353 172 757 MOV A,D
001.354 263 758 ORA E
001.355 302 342 001 759 JNZ LOAI IF MORE TO GO
760
001.360 315 172 002 761 CALL CTC CHECK TAPE CHECKSUM
762
763 * READ NEXT BLOCK
764
001.363 361 765 POF PSW (A) = FILE TYPE BYTE
001.364 301 766 POF B (BC) = -(LAST TYPE, LAST #)
001.365 007 767 RLC

PAM/B - H8 FRONT PANEL MONITOR #01.00.00.
LOAD - LOAD MEMORY FROM TAPE

HEATH X8ASM V1.1 06/21/77
15:44:21 01-APR-77 PAGE 20

001.366 332 133 002 768 JC TFT ALL DONE - TURN OFF TAPE
001.371 303 272 001 769 JMP LOAD READ ANOTHER RECORD



PAM/B - HB FRONT PANEL MONITOR \$01.00.00,
DUMP = DUMP MEMORY TO MAG/PAPER TAPE

HEATH X8ASM V1.0 02/18/77
13:23:47 01-APR-77 PAGE 21



772 *** DUMP = DUMP MEMORY TO MAG TAPE.
773 *
774 * DUMP SPECIFIED MEMORY RANGE TO MAG TAPE.
775 *
776 * ENTRY (START) = START ADDRESS
777 * (AHSS) = END ADDRESS
778 * USER PC = ENTRY POINT ADDRESS
779 * EXIT TO CALLER.
780
781
001.374 041 244 002 782 WMEM EQU *
001.374 042 031 040 783 LXI H,TPC
001.377 042 031 040 784 SHLD TPFRRX SETUP ERROR EXIT
001.377 042 031 040 785
002.002 076 001 786 DUMP MVI A,UC1,TE
002.004 323 371 787 OUT OF,TPC SETUP TAPE CONTROL
002.006 076 026 788 MVI A,A,SYN
002.010 046 040 789 MVI H,32 (H) = # OF SYNC CHARACTERS
002.012 315 024 003 790 WME1 CALL WNE
002.015 045 791 DCR H
002.016 302 012 002 792 JNZ WME1 WRITE SYN HEADER
002.021 076 002 793 MVI A,A,STX
002.023 315 024 003 794 CALL WNB WRITE STX
002.026 154 795 MOV L,H (HL) = 00
002.027 042 027 040 796 SHLD CRCSUM CLEAR CRC 16
002.032 041 001 201 797 LXI H,R7,MJ+80H*256# FIRST AND LAST MJ RECORD
002.035 315 017 003 798 CALL WNF WRITE HEADER
002.040 052 000 040 799 LHLD START
002.043 353 800 XCNS (D,E) = START ADDRESS
002.044 052 024 040 801 LHLD ABUS (H,L) = STOP ADDR
002.047 043 802 INX H COMPUTE WITH STOP#
002.050 175 803 MOV A,L
002.051 223 804 SUB E
002.052 157 805 MOV L,A
002.053 174 806 MOV A,H
002.054 232 807 SBB D
002.055 147 808 MOV H,A (HL) = COUNT
002.056 315 017 003 809 CALL WNF WRITE COUNT
002.061 345 810 PUSH H
002.062 076 012 811 MVI A,10
002.064 325 812 PUSH D SAVE (DE)
002.065 315 052 003 813 CALL LRA LOCATE P-REG ADDRESS
002.070 176 814 MOV A,M
002.071 043 815 INX H
002.072 146 816 MOV H,M
002.073 157 817 MOV L,A (HL) = CONTENTS OF PC
002.074 315 017 003 818 CALL WNF WRITE HEADER
002.077 341 819 POP H (HL) = ADDRESS
002.100 321 820 POP D (DE) = COUNT
002.101 315 017 003 821 CALL WNF
002.101 315 017 003 822
002.104 176 823 WME1 MOV A,M
002.105 315 024 003 824 CALL WNB WRITE BYTE
002.110 042 024 040 825 SHLD ABUS SET ADDRESS FOR DISPLAY
002.113 043 826 INX H
002.114 033 827 DEX D

PAM/8 - H8 FRONT PANEL MONITOR \$01.00.00.
 DUMP - DUMP MEMORY TO MAG/PAPER TAPE

HEATH X8ASM V1.0 02/18/77
 13:23:49 01-AFR-77 PAGE 22

002.115	172	828	MOV	A,I	
002.116	263	829	ORA	E	
002.117	302 104 002	830	JNZ	WME2	IF MORE TO GO
		831			
		832	*	WRITE CHECKSUM	
		833			
002.122	052 027 040	834	LHLI	CRCSUM	
002.125	315 017 003	835	CALL	WNF	WRITE IT
002.130	315 017 003	836	CALL	WNF	FLUSH CHECKSUM
		837	*	JMP	TFT

		839	**	TFT	- TURN OFF TAPE.
		840	*		
		841	*	STOP THE TAPE TRANSPORT.	
		842	*		
		843			
002.133	257	844	TFT	XRA	A
002.134	323 371	845	OUT	OF,TFC	TURN OFF TAPE

		847	**	HORN - MAKE NOISE.	
		848	*		
		849	*	ENTRY (A) = (MILLISECOND COUNT)/2	
		850	*	EXIT NONE	
		851	*	USES A,F	
		852			
		853			
002.136	076 144	854	ALARM	MVI	A,200/2 200 MS BEEP
002.140	385	855	HORN	PUSH	FSW
002.141	076 200	856		MVI	A,CH,SPK TURN ON SPEAKER
		857			
002.143	343	858	HRNO	XTHL	SAVE (HL), (H) = COUNT
002.144	325	859		PUSH	D SAVE (DE)
002.145	353	860		XCHG	(H) = LOOP COUNT
002.146	041 011 040	861		LXI	H,CTLFLG
002.151	256	862		XRA	M
002.152	136	863		MOV	E,M (E) = OLD CTLFLG VALUE
002.153	167	864		MOV	M,A
002.154	056 033	865		MVI	L,*TICOUNT TURN ON HORN
		866			
002.156	172	867		MUV	A,I (A) = CYCLE COUNT
002.157	206	868		AII	M
002.160	276	869	HRN2	CMF	M WAIT REQUIRED TICCOUNTS
002.161	302 160 002	870	JNE	HRN2	
002.164	056 011	871	MVI	L,*CTLFLG	
002.166	163	872	MOV	M,E	TURN HORN OFF.
002.167	321	873	POF	I	
002.170	341	874	POF	H	
002.171	311	875	RET		



PAM/8 - H8 FRONT PANEL MONITOR \$01.00.00.
TAPE PROCESSING SUBROUTINES

HEATH X8ASM V1.1 06/21/77
15:44:25 01-APR-77 PAGE 23



880 ** CTC -- VERIFY CHECKSUM.
881 *
882 * ENTRY TAPE JUST BEFORE CRC
883 * EXIT TO CALLER IF OK
884 * TO *TFERR* IF BAD
885 * USES A,F,H,L
886
887
002.172 315 325 002 888 CTC CALL RNP READ NEXT FAIR
002.175 052 027 040 889 LHLD CRCSUM
002.200 124 890 MOV A,H
002.201 265 891 ORA L
002.202 310 892 RZ RETURN OF ON
002.203 076 001 893 MVI A,1 CHECKSUM ERROR
894 * JMP TFERR (B) = CODE

896 ** TFERR -- PROCESS TAPE ERROR.
897 *
898 * DISPLAY ERR NUMBER IN LOW BYTE OF ABUSS
899 *
900 * IF ERROR NUMBER EVEN, DONT ALLOW *
901 * IF ERROR NUMBER ODD, ALLOW *
902 *
903 * ENTRY (A) = NUMBER
904
905
002.205 062 024 040 906 TFERR STA ABUSS
002.210 107 907 MOV B,A (B) = CODE
002.211 315 133 002 908 CALL TFT TURN OFF TAPE
909
910 * IS *, RETURN (IF PARITY ERROR)
911
002.214 346 912 DB MI.ANI FALL THROUGH WITH CARRY CLEAR
002.215 170 913 TER3 MOV A,B
914
002.216 017 915 RRC
002.217 330 916 RC RETURN IF OK
917
918 * BEEP AND FLASH ERROR NUMBER
919
002.220 334 136 002 920 TER1 CC ALARM ALARM IF PROPER TIME
002.223 315 252 002 921 CALL TFXIT SEE IF *
002.226 333 360 922 IN IF.FAD
002.230 376 057 923 CPI 00101111B CHECK FOR *
002.232 312 215 002 924 JE TER3 IF *
002.235 072 034 040 925 LIA TICCNT#1
002.240 032 926 RAR 'C' SET IF 1/2 SECOND
002.241 303 220 002 927 JMP TER1

FAM/8 -- H8 FRONT PANEL MONITOR \$01.00.00.
TAPE PROCESSING SUBROUTINES

HEATH X8ASM V1.0 02/18/77
13:23:52 01-APR-77 PAGE 24

```

929 ** TFABRT - ABORT TAPE LOAD OR DUMP.
930 *
931 * ENTERED WHEN LOADING OR DUMPING, AND THE '*' KEY
932 * IS STRUCK.
933
934
002.244 257 935 TFABRT XRA A
002.245 323 371 936 OUT OF.TFC OFF TAPE
002.247 303 322 000 937 JMF ERROR

```

```

939 ** TFXIT -- CHECK FOR USER FORCED EXIT.
940 *
941 * TFXIT CHECKS FOR AN '*' KEYFAI ENTRY. IF SO, TAKE
942 * THE TAPE DRIVER ABNORMAL EXIT.
943 *
944 * ENTRY NONE
945 * EXIT TO *RET* IF NOT '*'
946 * (A) = PORT STATUS
947 * TO (TFERRX) IF '*' DOWN
948 * USES A,F
949
950
002.252 333 360 951 TFXIT IN IF.FAI
002.254 376 157 952 CPI 01101111B *
002.256 333 371 953 IN IF.TFC READ TAPE STATUS
002.260 300 954 RNE NOT '*', RETURN WITH STATUS
002.261 052 031 040 955 LHLW TFERRX
002.264 351 956 FCHL ENTER (TFERRX)

```

```

958 ** SRS -- SCAN RECORD START
959 *
960 * SRS READS BYTES UNTIL IT RECOGNIZES THE START OF A RECORD.
961 *
962 * THIS REQUIRES
963 * AT LEAST 10 SYNC CHARACTERS.
964 * 1 STX CHARACTER.
965 *
966 * THE CRC-16 IS THEN INITIALIZED.
967 *
968 * ENTRY NONE
969 * EXIT TAPE POSITIONED (AND MOVING), CRCSUM = 0
970 * (IE) = HEADER BYTES
971 * (HA) = RECORD COUNT
972 * USES A,F,D,E,H,L
973
974
002.265 975 SRS EQU *
002.265 026 000 976 SRS1 MVI D,O
002.267 142 977 MOV H,D
002.270 152 978 MOV L,D (HL) = 0

```



PAM/8 - H8 FRONT PANEL MONITOR \$01.00.00,
TAPE PROCESSING SUBROUTINES

HEATH X8ASM V1.0 02/18/77
13:23:54 01-APR-77 PAGE 25



```
002.271 315.331.002 979 SRS2 CALL RNB READ NEXT BYTE
002.274 024 980 INR D
002.275 326.026 981 CPI A,SYN
002.277 312.271.002 982 JE SRS2 HAVE SYN
002.302 326.002 983 CPI A,STX
002.304 302.265.002 984 JNE SRS1 NOT STX - START OVER
002.307 076.012 986 MVI A,10
002.311.272 987 CMP D SEE IF ENOUGH SYN CHARACTERS
002.312 322.265.002 988 JNC SRS1 NOT ENOUGH
002.315 042.027.040 989 SHLD CROSUM CLEAR CRC-16
002.320 315.325.002 990 CALL RNP READ LEADER
002.323 124 991 MOV D,H
002.324 137 992 MOV E,A
002.325 * 993 * JMF RNP READ COUNT
```

```
995 ** RNP = READ NEXT PAIR.
996 * RNP READS THE NEXT TWO BYTES FROM THE INPUT DEVICE.
997 *
998 *
999 *
1000 * ENTRY NONE
1000 * EXIT (H,A) = BYTE PAIR
1001 * USES A,F,H
1002
1003
002.325 315.331.002 1004 CALL RNB READ NEXT BYTE
002.330 147 1005 MOV H,A
1006 * JMF RNP READ NEXT BYTE
```

```
1008 ** RNB = READ NEXT BYTE
1009 *
1010 * RNB READS THE NEXT SINGLE BYTE FROM THE INPUT DEVICE.
1011 * THE CHECKSUM IS TAKEN FOR THE CHARACTER.
1012 *
1013 *
1014 * ENTRY NONE
1014 * EXIT (A) = CHARACTER
1015 * USES A,F
1016
1017
002.331 026.064 1018 RNB MVI A,UCI.RU+UCI.EF+UCI.RE TURN ON READER FOR NEXT BYTE
002.333 323.331 1019 OUT OF,TFC
002.335 315.252.002 1020 RNP1 CALL TXIT CHECK FOR *, READ STATUS
002.340 346.002 1021 ANI USR,RXR
002.342 312.335.002 1022 JZ RNP1 IF NOT READY
002.345 333.370 1023 IN TF,TFR INPUT DATA
1024 * JMF CRC CHECKSUM
```

PAM/8 - HB FRONT PANEL MONITOR \$01.00.00.
TAPE PROCESSING SUBROUTINESHEATH X8ASM V1.0 02/18/77
13:23:56 01-APR-77 PAGE 26

```

1026 **      CRC = COMPUTE_CRC-16
1027 *
1028 *      CRC COMPUTES A CRC-16 CHECKSUM FROM THE POLYNOMIAL
1029 *
1030 *      (X + 1) * (X^15 + X + 1)
1031 *
1032 *      SINCE THE CHECKSUM GENERATED IS A DIVISION REMAINDER,
1033 *      A CHECKSUMMED DATA SEQUENCE CAN BE VERIFIED BY RUNNING
1034 *      THE DATA THROUGH CRC, AND THEN RUNNING THE PREVIOUSLY OBTAINED
1035 *      CHECKSUM THROUGH CRC. THE RESULTANT CHECKSUM SHOULD BE '0'.
1036 *
1037 *      ENTRY  (CRCSUM) = CURRENT CHECKSUM
1038 *              (A) = BYTE
1039 *      EXIT   (CRCSUM) UPDATED
1040 *              (A) UNCHANGED
1041 *      USES   R
1042
1043
002.347 305    1044 CRC    PUSH   B      SAVE (BC)
002.350 006 010  1045 MVI    B,B      (B) = BIT COUNT
002.352 345    1046 PUSH   H
002.353 052 027 040 1047 LHLD   CRCSUM
002.356 007    1048 CRC1   RLC
002.357 117    1049 MOV    C,A      (C) = BIT
002.360 175    1050 MOV    A,L
002.361 207    1051 ADD    A
002.362 157    1052 MOV    L,A
002.363 174    1053 MOV    A,H
002.364 027    1054 RAL
002.365 147    1055 MOV    H,A
002.366 027    1056 RAL
002.367 251    1057 XRA    C
002.370 017    1058 RRC
002.371 322 004 003 1059 JNC    CRC2   IF NOT TO XOR
002.374 174    1060 MOV    A,H
002.375 356 200  1061 XRI    200Q
002.377 147    1062 MOV    H,A
003.000 175    1063 MOV    A,L
003.001 356 005  1064 XRI    SQ
003.003 157    1065 MOV    L,A
003.004 171    1066 CRC2   MOV    A,C
003.005 005    1067 DCR    B
003.006 302 356 002 1068 JNZ    CRC1   IF MORE TO GO
003.011 042 027 040 1069 SHLD   CRCSUM
003.014 341    1070 POF    H      RESTORE (HL)
003.015 301    1071 POF    B      RESTORE (BC)
003.016 311    1072 RET

```



PAN8 - H8 FRONT PANEL MONITOR \$01.00.00.
TAPE PROCESSING SUBROUTINES

HEATH X8ASM V1.0 02/18/77
13:23:58 01 APR 77 PAGE 27



1074 ** WNB - WRITE NEXT PAIR.
1075 *
1076 * WNB WRITES THE NEXT TWO BYTES TO THE CASSETTE DRIVE.
1077 *
1078 * ENTRY (H,L) = BYTES
1079 * EXIT WRITTEN.
1080 * USES A,F
1081
1082
003.017 174 1083 WNB MOV A,H
003.020 315 024 003 1084 CALL WNB
003.023 175 1085 MOV A,L
1086 * JMP WNB WRITE NEXT BYTE

1088 ** WNB - WRITE BYTE
1089 *
1090 * WNB WRITES THE NEXT BYTE TO THE CASSETTE TAPE.
1091 *
1092 * ENTRY (A) = BYTE
1093 * EXIT NONE.
1094 * USES F
1095
1096
003.024 365 1097 WNB PUSH PSW
003.025 315 252 002 1098 WNB1 CALL TXIT CHECK FOR *, READ STATUS
003.030 346 001 1099 ANI USR,TXR
003.032 312 025 003 1100 JZ WNB1 IF MORE TO GO
003.035 076 021 1101 MOV A,UCI,ERFLCT,IE ENABLE TRANSMITTER
003.037 323 371 1102 OUT OF,TPI TURN ON TAPE
003.041 361 1103 POP PSW
003.042 323 370 1104 OUT OF,TPI OUTPUT DATA
003.044 303 347 002 1105 JMP CRC COMPUTE CRC

PAM/8 - HB FRONT FANEL MONITOR \$01.00.00.
SUBROUTINES

HEATH X8ASM V1.0 02/18/77
13:23:59 01-APR-77 PAGE 28

```

1109 ** LRA - LOCATE REGISTER ADDRESS.
1110 *
1111 * ENTRY NONE.
1112 * EXIT (A) = REGISTER INDEX
1113 * (H,L) = STORAGE ADDRESS
1114 * (H,E) = (0,A)
1115 * USES A,I,E,H,L,F
1116
1117
1118
003.047 072 005 040 1119 LRA LDA REGI
003.052 137 1120 LRA MUV E,A
003.053 026 000 1121 MVI I,O
003.055 052 035 040 1122 LHLD REGPTR
003.060 031 1123 DAD I
003.061 311 1124 RET (DE) = (REGPTR)+(REGI)

1126 ** IOA - INPUT OCTAL ADDRESS.
1127 *
1128 * ENTRY (H,L) = ADDRESS OF RECEPTION DOUBLE BYTE.
1129 * EXIT TO *RET* IF ERROR.
1130 * TO *RET*+1 IF OK, VALUE IN MEMORY.
1131 * USES A,I,E,H,L,F
1132
1133
003.062 315 066 003 1134 IOA CALL IOB INPUT BYTE
003.065 053 1135 IORX H

1137 ** IOB - INPUT OCTAL BYTE.
1138 *
1139 * READ ONE OCTAL BYTE FROM THE KEYSET.
1140 *
1141 * ENTRY (H,L) = ADDRESS OF BYTE TO HOLD VALUE
1142 * TO SET IF FIRST DIGIT IN (A)
1143 * EXIT TO *RET* IF ALL OK
1144 * TO *ERRUR* IF ERROR
1145 * USES A,I,E,H,L,F
1146
1147
1148
003.066 026 003 1149 IOB MVI I,3 (D) = DIGIT COUNT
003.070 324 260 003 1150 IOBI CNC RCK READ CONSOLE KEYSET
1151
003.073 376 010 1152 CPI 8
003.075 322 322 000 1153 JNC ERROR IF ILLEGAL DIGIT
1154
003.100 137 1155 MOV E,A (E) = VALUE
003.101 126 1156 MOV A,M
003.102 007 1157 RLC SHIFT 3
003.103 007 1158 RLC

```



PAM/S - HS FRONT PANEL MONITOR \$01.00.00.
SUBROUTINES

HEATH X8ASM V1.0 02/18/77
13:24:01 01 APR-77 PAGE 29



003.104	007	1159	RLC	
003.105	346 370	1160	ANI	370Q
003.107	263	1161	ORA	E
003.110	167	1162	MOV	M,A
003.111	025	1163	BCR	D
003.112	302 070 003	1164	JNZ	I0R1
003.115	076 017	1165	MVI	A,30/2
003.117	303 140 002	1166	JMP	HORN

1168 ** D0B = DECODE FOR OCTAL DISPLAY.
1169 *
1170 * ENTRY (H,L) = ADDRESS OF LED REFRESH AREA

(B) = *OR# PATTERN TO FORCE ON BARS OR PERIODS

(A) = OCTAL VALUE

1173 * EXIT (H,L) = NEXT DIGIT ADDRESS

1174 * USES A,B,U,B,H,L

1175

1176

003.122	325	1177	POP	PUSH B
003.123	028 003	1178	MVI	B,D0DA/256
003.125	016 003	1179	MVI	C,3
003.127	027	1180	POB1	RAL LEFT 3 PLACES
003.130	027	1181	RAL	
003.131	027	1182	RAL	

003.132	368	1183	PUSH PSW	SAVE FOR NEXT DIGIT
003.133	346 007	1184	ANI	/
003.135	306 356	1185	ABI	*D0DA

003.137	137	1186	MOV	E,A (D) = INDEX
003.140	032	1187	LIAx	B (A) = PATTERN

003.141	250	1188	XRA	B
003.142	346 177	1189	ANI	17%Q

003.144	250	1190	XRA	B
003.145	167	1191	MOV	M,A SET IN MEMORY

003.146	043	1192	INX	H
003.147	170	1193	MOV	A,B

003.150	007	1194	RLC	
003.151	107	1195	MOV	B,A

003.152	361	1196	POP	PSW (A) = VALUE
003.153	015	1197	BCR	C

003.154	302 127 003	1198	JNZ	POB1 IF MORE TO GO
003.157	321	1199	POP	D

003.160	311	1200	RET	RETURN
---------	-----	------	-----	--------

FAM/8 - H8 FRONT FANEL MONITOR \$01.00.00.
UFD - UPDATE FRONT FANEL DISPLAYS.

HEATH X8ASM V1.0 02/18/77
13:24:02 01-AFR-77 PAGE 30

```

1203 ** UFD - UPDATE FRONT PANEL DISPLAYS.
1204 *
1205 *
1206 * UFD IS CALLED BY THE CLOCK INTERRUPT PROCESSOR WHEN IT IS
1207 * TIME TO UPDATE THE DISPLAY CONTENTS. CURRENTLY, THIS IS DONE
1208 * EVERY 32 INTERRUPTS, OR ABOUT 32 TIMES A SECOND.
1209 *
1210 * ENTRY (H,L) = ADDRESS OF REGCNT
1211 * EXIT NONE
1212 * USES ALL
1213
1214
003.161 076 002 1215 UFD EQU *
003.161 076 002 1216 MVI A,00.100
003.163 240 1217 ANA B
003.164 300 1218 RNZ IF NOT TO HANDLE UPDATE
1219
003.165 056 006 1220 MVI L,DISPROT
003.167 176 1221 MOV A,M
003.170 007 1222 RLC
003.171 167 1223 MOV M,A ROTATE PATTERN
003.172 107 1224 MOV B,A
003.173 043 1225 INX H
000.000 1226 ERRNZ DISPMDI-DISPROT-1
003.174 176 1227 MOV A,M (A) = DISPMDI
003.175 346 002 1228 ANI 2
003.177 052 024 040 1229 LHLD ABUS$ IF MEMORY
003.202 312 227 003 1230 JZ UFD1
1231
1232 * AM DISPLAYING REGISTERS.
1233
003.205 315 047 003 1234 CALL LR A LOCATE REGISTER ADDRESS
003.210 345 1235 PUSH H
003.211 041 342 003 1236 LXI H,DISPA
003.214 031 1237 DAD H (H,L) = ADDRESS OF REG NAME PATTERNS
003.215 176 1238 MOV A,H
003.216 043 1239 INX H
003.217 146 1240 MOV H,M
003.220 157 1241 MOV L,A (H,L) = REG NAME PATTERN
003.221 343 1242 XTHL
003.222 264 1243 ORA H CLEAR Z
003.223 176 1244 MOV A,M
003.224 043 1245 INX H
003.225 146 1246 MOV H,M
003.226 157 1247 MOV L,A (HL) = ADDRESS OF REGISTER FAIR CONTENTS
1248
1249 * SETUP DISPLAY
1250
003.227 365 1251 UFD1 PUSH PSW
003.230 353 1252 XCHG
003.231 041 013 040 1253 LXI H,ALEIS
003.234 172 1254 MOV A,I
003.235 315 122 003 1255 CALL NOI FORMAT ABANK HIGH HALF
003.240 173 1256 MOV A,E
003.241 315 122 003 1257 CALL NOI FORMAT ABANK LOW HALF
003.244 361 1258 POP PSW

```

RAM/8 - H6 FRONT PANEL MONITOR \$01.00.00,
OFB - UPDATE FRONT PANEL DISPLAYS.

HEATH X8ASM V1.0 02/18/77
13:24:04 01-APR-77 PAGE 31



Panel Monitor

003.245	03A	1259	LDAH	H	
003.246	312 122 003	1260	JZ	00P	IF MEMORY, DECODE BYTE VALUE
		1261			
		1262 *	IS REGISTER. SEE REGISTER NAME.		
		1263			
003.251	066 377	1264	MVI	M, 3770	CLEAR DIGIT
003.253	341	1265	PUP	H	
003.254	042 022 040	1266	SHLD	M, EBS+1	
003.257	311	1267	RET		



FAM/8 - HB FRONT PANEL MONITOR \$01.00.00.
RCK - READ CONSOLE KEYPAD.

HEATH X8ASM V1.1 06/21/77
15:44:39 01-APR-77 PAGE 32

```

1271 ** RCK - READ CONSOLE KEYPAD.
1272 *
1273 * RCK IS CALLED TO READ A KEYSTROKE FROM THE CONSOLE KEYPAD.
1274 * WHENEVER A KEY IS ACCEPTED.
1275 * RCK PERFORMS DEBOUNCING, AND AUTO-REPEAT. A *BIF* IS SOUNDED
1276 * WHEN A VALUE IS ACCEPTED.
1277 *
1278 * KEY PAD VALUES:
1279 *
1280 * 1111 1110 -- 0
1281 * 1111 1100 -- 1
1282 * 1111 1010 -- 2
1283 * 1111 1000 -- 3
1284 * 1111 0110 -- 4
1285 * 1111 0100 -- 5
1286 * 1111 0010 -- 6
1287 * 1111 0000 -- 7
1288 * 1110 1111 -- 8
1289 * 1100 1111 -- 9
1290 * 1010 1111 -- +
1291 * 1000 1111 -- -
1292 * 0110 1111 -- *
1293 * 0100 1111 -- /
1294 * 0010 1111 -- #
1295 * 0000 1111 -- :
1296 *
1297 *
1298 * ENTRY NONE
1299 * EXIT TO CALLER WHEN A KEY IS HIT
1300 * (A) = 0 -- 0
1301 * 1 -- 1
1302 * 2 -- 2
1303 * 3 -- 3
1304 * 4 -- 4
1305 * 5 -- 5
1306 * 6 -- 6
1307 * 7 -- 7
1308 * 10 -- 8
1309 * 11 -- 9
1310 * 12 -- 10
1311 * 13 -- 11
1312 * 14 -- 12
1313 * 15 -- 13
1314 * 16 -- 14
1315 * 17 -- 15
1316 * USES A,F
1317
1318
003.260 1319 RCK EQU *
003.260 .345 1320 PUSH H
003.261 305 1321 PUSH B
003.262 .016 .024 1322 MVI C,400/20 WAIT 400 MS
003.264 041 026 040 1323 LXI H,RCKA
1324
003.267 333 360 1325 RCK1 IN IP,FAI INPUT FAI VALUE
003.271 .107 1326 MOV B,A (B) = VALUE

```

PAM/B - H8 FRONT PANEL MONITOR #01.00.00.
RCK - READ CONSOLE KEYPAD.

HEATH X8ASM V1.1 .06/21/77
15:44:41 01-APR-77 PAGE 33



```
003.272 026.012 1327    MVI   A,20/2
003.274 315 053 000 1328    CALL  ILT      WAIT 20 MS
003.277 170 1329    MOV   A,B
003.300 276 1330    CMF   M
003.301 302 310.003 1331    JNE   RCK2      HAVE A CHANGE
003.304 015 1332    PCK   C
003.305 302 267.003 1333    JNZ   RCK1      WAIT N CYCLES
003.306 1334
003.307 1335 *      HAVE KEY VALUE
003.308 1336
003.310 167 1337 RCK2    MOV   M,A      UPDATE RCKA
003.311 356 376 1338    XRI  3760      INVERT ALL BUT GROUP 0 FLAG
003.313 017 1339    RRC
003.314 322 326 003 1340    JNC   RCK3      HIT BANK 0
003.317 017 1341    RRC
003.320 017 1342    RRC
003.321 017 1343    RRC
003.322 017 1344    RRC
003.323 322 267.003 1345    JNC   RCK1      NO HIT AT ALL
003.326 107 1346 RCK3    MOV   B,A      (B) = CODE
003.327 076.002 1347    MVI  A,4/2
003.331 315 140.002 1348    CALL  HORN      MAKE BIP
003.334 170 1349    MOV   A,B
003.335 346 017 1350    ANI  170
003.337 301 1351    POP   B
003.340 341 1352    POP   H
003.341 311 1353    RET
003.342 1354
```



RAM/8 - H8 FRONT PANEL MONITOR #01.00.00.
SEGMENT PATTERNS AND CONSTANTS.

HEATH X8ASM.V1.1 06/21/77
15:44:42 01-AFR-77 PAGE 34

```
1357 ** DISPLAY SEGMENT CODING:  
1358 *  
1359 * BYTE = 76 543 210  
1360 *  
1361 * 1  
1362 * 6 2  
1363 * 0  
1364 * 5 3  
1365 * 4  
1366 * 7
```

```
1370 ** REGISTER INDEX TO 7-SEGMENT PATTERN.  
1371  
003.342 1372 USPA DS 0  
003.342 244 230 1373 DW 1001100010100100B SF  
003.344 220 234 1374 DW 1001110010010000B AF  
003.346 206 215 1375 DW 1000110110000110B RC  
003.350 302 214 1376 DW 1000110011000010B DE  
003.352 222 217 1377 DW 1000111110010010B HL  
003.354 230 316 1378 DW 1100111010011000B FC
```

```
1380 ** OCTAL TO 7-SEGMENT PATTERN  
1381  
003.356 1382 D0DA DS 0  
003.356 001 1383 DR 00000001B 0  
003.357 163 1384 DR 01110011B 1  
003.360 110 1385 DR 01001000B 2  
003.361 140 1386 DR 01100000B 3  
003.362 062 1387 DR 00110010B 4  
003.363 044 1388 DR 00100100B 5  
003.364 004 1389 DR 00000100B 6  
003.365 161 1390 DR 01110001B 7  
003.366 000 1391 DR 00000000B 8  
003.367 040 1392 DR 00100000B 9
```

```
1394 ** I/O ROUTINES TO BE COPIED INTO AND USED IN RAM.  
1395 *  
1396 * MUST CONTINUE TO 3777A FOR PROPER COPY.  
1397 * THE TABLE MUST ALSO BE BACKWARDS TO THE FINAL RAM  
1398  
003.371 1399 ORG 4000A-7  
1400  
.003.371 1401 FRSRDM EQU *  
003.371 001 1402 DR 1 REFINI  
.003.372 000 1403 DR 0 CTLFLG  
003.373 000 1404 DR 0 .MFLAG
```

PAM/8 - H8 FRONT PANEL MONITOR \$01.00.00.
CONSTANTS AND TABLES.

HEATH X8ASM V1.1 06/21/77
15:44:44 01-AFR-77 PAGE 35

003.374	000	1405	DB	0	DISPMOD
003.375	000	1406	DB	0	DISPROT
003.376	012	1407	DB	10	REGI
003.377	311	1408	DB	M1.RET	
		1409			
000.000		1410		ERRNZ *-4000A	



PAM/8 - H8 FRONT PANEL MONITOR \$01.00.00.
RAM CELLS

HEATH X8ASM V1.1 06/21/77
15:44:44 01-AFR-77 PAGE 36

```

1413
1414 ** THE FOLLOWING ARE CONTROL CELLS AND FLAGS USED BY THE KEYPAD
1415 * MONITOR.
1416
040.000 1417 ORG 40000A 8192
040.000 1418 START DS 2 DUMP STARTING ADDRESS
040.002 1419 IOWRN DS 2 IN OR OUT INSTRUCTION
040.004 1420 FRSRAM EQU * FOLLOWING CELLS INITIALIZED FROM ROM
040.004 1421 DS 1 RET
1422
040.005 1423 REGI DS 1 INDEX OF REGISTER UNDER DISPLAY
040.006 1424 DSFPROT DS 1 PERIOD FLAG BYTE
040.007 1425 DSFMOD DS 1 DISPLAY MODE
1426
040.010 1427 MFLAG DS 1 USER FLAG OPTIONS
1428 * SEE *U0.XXX* BITS DESCRIBED AT FRONT
1429
040.011 1430 CTLFLG DS 1 FRONT PANEL CONTROL BITS
040.012 1431 REFINI DS 1 REFRESH INDEX (0 TO 7)
000.007 1432 FRSL EQU *-FRSRAM END OF AREA INITIALIZED FROM ROM
1433
040.013 1434 FFLEDS EQU * FRONT PANEL LED PATTERNS
040.013 1435 ALEDS DS 1 ADDR 0
040.014 1436 DS 1 ADDR 1
040.015 1437 DS 1 ADDR 2
1438
040.016 1439 DS 1 ADDR 3
040.017 1440 DS 1 ADDR 4
040.020 1441 DS 1 ADDR 5
1442
040.021 1443 DLEDS DS 1 DATA 0
040.022 1444 DS 1 DATA 1
040.023 1445 DS 1 DATA 2
1446
040.024 1447 ABUS DS 2 ADDRESS BUS
040.026 1448 RCRA DS 1 RC SAVE AREA
040.027 1449 CRCSUM DS 2 CRC-16 CHECKSUM
040.031 1450 TFERRX DS 2 TAPE ERROR EXIT ADDRESS
040.033 1451 TICCNT DS 2 CLOCK TIC COUNTER
1452
040.035 1453 REGPTR DS 2 REGISTER CONTENTS POINTER
1454
040.037 1455 UIVEC DS 0 USER INTERRUPT VECTORS
040.037 1456 DS 3 JUMP TO CLOCK PROCESSOR
040.042 1457 DS 3 JUMP TO SINGLE STEP PROCESSOR
040.045 1458 DS 3 JUMP TO I/O 3
040.050 1459 DS 3 JUMP TO I/O 4
040.053 1460 DS 3 JUMP TO I/O 5
040.056 1461 DS 3 JUMP TO I/O 6
040.061 1462 DS 3 JUMP TO I/O 7
1463
040.064 1464 END
ASSEMBLY COMPLETE.
1464 STATEMENTS
0 ERRORS DETECTED
22310 BYTES FREE

```



CROSS REFERENCE TABLE.

XREF V1.0 PAGE 37

040011	2175	3445	4945	5135	5745	5865	5945	6095	6305	6455	6995
,MFLAG	040010	344	381	386	418	452	465	1427L			
A,STX	000002	93E	793	983							
A,SYN	000026	92E	788	961							
ABORT	001147	529	617L								
ABUSG	040024	490	554	650	754	801	829	903	1229	1147L	
ALARM	002136	469	854L	920							
ALEIS	040013	1293	1435L								
BLKSIZE	002900	118E									
CR,CLI	000100	99	136	256	464						
CR,MTL	000640	78E	331	415	464	701					
- CR,SPK	000200	100E	258	464	856						
CR,SS1	000020	97E	256	451	464	688	697				
CLK2	002334	393	395L								
CLK3	000237	389	398E								
CLK4	000313	422	438E								
CLOCK	000201	202	203	369L							
CRC	002347	1044L	1105								
CRC1	002350	1048L	1068								
CRC2	003004	1059	1066L								
CRCSUM	040027	798	834	889	989	1047	1069	1449L			
CTC	002172	761	888L								
CTRLFLG	040011	212	328	386	391	413	418	465	687	870	899
		1430L									
CU11	000165	345L	441								
DELEIS	040021	1266	1443L								
DLY	000053	248L	1328								
IM.MR	000000	104E									
IM.MW	000001	105E									
IM.RR	000002	106E									
IM.RW	000003	107E									
IOU1	003122	1177L	1255	1257	1260						
IOU1	003127	1180L	1198								
IOUA	003356	1178	1185	1382L							
ISPA	003342	1236	1372L								
ISPMOD	040007	481	494	513	574	576	594	645	647	1226	1425L
ISPROMT	040006	485	576	847	1220	1226	1424L				
IUMP	002002	786L									
ERROR	000322	298	434	458E	503	562	583	937	1153		
FFLEIS	040013	1434E									
GO	001222	521	679L								
GO,	000063	256L	379								
HORN	002140	855L	1166	1348							
HRNO	002143	250	858L								
HRN2	002160	869L	370								
IN	001177	522	660L								
INIT	000073	192	193	275L	279						
INIT0	000006	190L									
INIT1	000107	288L	293								
INIT2	000117	295L									
INT1	000010	197E									
INT2	000020	212E									
INT3	000030	229L									
INT4	000040	234L									
INT5	000050	239L									
INT6	000060	253L									
INT7	000070	260L									
INTXIT	000172	392L	416	692							

CROSS REFERENCE TABLE.

XREF V1.0 PAGE 38

IOA	003062	565	651	1134L
IOB	003066	545	1134	1149L
IOR1	003070	1150L	1164	
IOWRK	040002	666	667	1419L
IP.FAD	000360	81E	439	922 951 1325
IP.TPC	000371	85E	953	
IP.TPI	000370	87E	1023	
LAST	001150	528	625L	
LOAO	001272	726L	769	
LOA1	001342	752L	759	
LOAD	001267	724E		
LRA	003047	560	1119L	1234
LRA.	003052	427	743	813 1120L
LST2	001154	631L		
MEMM	001165	531	644L	
MI.ANI	000346	128E	912	
MI.HLT	000166	123E	433	
MI.IN	000333	125E	660	
MI.LDA	000072	127E		
MI.LXID	000021	129E	661	
MI.OUT	000323	126E	662	
MI.RET	000311	124E	1408	
MTR	000344	476E	702	
MTR1	000345	479	479L	
MTR4	001005	492	502L	
MTR5	001051	497	541L	
MTR6	001067	543	559L	
MTRA	001035	506	520E	
NEXT	001132	527	604L	
UF.CTL	000360	82E	689	698
OF.DIG	000360	83E	400	
OF.SEG	000361	84E	402	
OF.TPC	000371	86E	304	787 845 936 1019 1102
OF.TPI	000370	88E	1104	
OUT	001202	523	662L	
FRSL	000007	191	1432E	
PRSRAM	040004	191	1420E	1432
FRSRROM	003371	190	1401E	
R\$W	001126	530	595L	
RCK	003260	489	580	1150 1319E
RCK1	003267	1325L	1333	1345
RCK2	003310	1331	1337L	
RCK3	003326	1340	1346L	
RCKA	040026	1323	1448L	
REFIND	040012	391	1431L	
REGI	040005	512	586	609 630 1119 1423L
REGM	001104	532	573L	
REGFTR	040035	335	467	1122 1453L
RMEM	001261	525	708L	
RNB	002331	752	979	1004 1018L
RNR1	002335	1020L	1022	
RNP	002325	738	748	888 990 1004L
RT.RP	000002	113E		
RT.CT	000003	114E		
RT.MI	000001	112E	725	797
SAE	001063	554L	605	626 670
SAVALL	000132	200	215	319L
SINCR	004000	281E	283	284



CROSS REFERENCE TABLE.

XREF V1.0 PAGE 39

SRS	002265	726	975E					
SRS1	002265	926L	984	988				
SRS2	002271	979L	982					
SST1	001235	357	690L					
SGTER	001225	524	685E					
START	040000	284	750	799	1418L			
SFRTN	001244	218	696E					
TER1	002220	920L	927					
TER3	002215	913L	924					
TFT	002133	768	844L	908				
TICNT	040033	369	371	406	885	929	1451L	
TFABT	002244	708	783	935L				
TFERR	002205	737	906L					
TFERRX	040031	709	784	956	1450L			
TRXIT	002252	921	951L	1020	1098			
UC1.EF	000020	165E	1018	1101				
UC1.EI	000002	167E						
UC1.IR	000100	163E						
UC1.RE	000004	166E	1018					
UC1.RD	000040	164E	1018					
UC1.TE	000001	168E	786	1101				
UDP	003161	409	1215E					
UDH1	003227	1230	1251L					
UDEV1	040037	229	234	239	253	260	348	703
UM1.16X	000002	158E	303					
UM1.1R	000100	148E	303					
UM1.1X	000001	157E						
UM1.2B	000300	150E						
UM1.64X	000003	159E						
UM1.HB	000200	149E						
UM1.L5	000000	153L						
UM1.L6	000004	154E						
UM1.L7	000010	155E						
UM1.L8	000014	156E	303					
UM1.PA	000020	152E						
UM1.PE	000040	151E						
UO_CLK	000001	138E	346					
UO_DDU	000002	137E	461	1216				
UO_HLT	000200	135E	420					
UO_NFR	000100	136E	384	461				
USR.FE	000040	172E						
USR.DE	000020	173E						
USR.FE	000010	174E						
USR.RXR	000002	176E	1021					
USR.TXE	000004	175E						
USR.TXR	000001	177E	1099					
WME1	002012	790L	792					
WME2	002104	823L	830					
WMEM	001324	526	782E					
WNF	003024	790	794	824	1084	1097L		
WNF1	003025	1098L	1100					
WNF	003017	798	809	818	821	835	836	1083L

25434 BYTES FREE



INDEX

- Addressing Port Pairs, 1-21
- Advanced Control, 1-22
- Alter Key, 1-9, 1-12
- Altering a Memory Location, 1-12 ff
- Altering a Selected Register, 1-15
- Audio Alert, 1-9
- Binary to Octal Conversion, 1-7
- Breakpoint Interrupts, 1-24
- Breakpointing, 1-16
- Cancel, 1-9, 1-19
- Checksum Errors, 1-20
- Clock, 1-35, 1-6
- Clock Interrupts, 1-5, 1-24
- Copying a Tape, 1-20
- Decrements Memory, 1-9
- Disable Interrupt, 1-5
- Displays, 1-7, 1-23
- Dump Routines, 1-17 ff
- Dumping, 1-18, 1-46
- Execution Control 1-16 ff
- Entry Point, 1-19
- Ending Address, 1-20
- GO Key, 1-16
- Halt Instruction, 1-16
- I/O, 1-21
- I/O Interrupts, 1-24
- IP.PAD, 1-22
- Increment Memory, 1-9
- Initialization, 1-5
- Inputting, 1-21
- Interrupt Vectors, 1-31
- Interrupts, 1-24, 1-26
- Interrupting a Program, 1-17
- Keypad, 1-9, 1-22
- Load Routines, 1-17 ff
- Loading, 1-18, 1-44
- MEM Key, 1-9
- MFLAG, 1-23
- Manual DI, 1-23
- Manual Updating, 1-23
- Master Clear, 1-5
- Monitor Mode, 1-6
- Offset Octal, 1-8
- Outputting, 1-21
- Port I/O, 1-21
- PAM/8 RAM Cells, 1-61
- Power-Up, 1-5
- RAM High Limit, 1-5
- RCK, 1-22, 1-57
- REG Key, 1-9
- RST, 1-5 ff, 1-9
- RTM, 1-5 ff, 1-17, 1-9
- Record Errors, 1-20
- Refreshing, 1-23
- Repeats, 1-9
- Single Instruction, 1-17
- Single Instruction Interrupts, 1-24
- Specifying a Memory Address, 1-10 ff
- Specifying a Register for Display, 1-14
- Stack Space, 1-5
- Stepping Through Memory, 1-13
- Stepping Through the Registers, 1-15
- TICCNT, 1-22
- Tape Errors, 1-20, 1-48
- Tick Counter, 1-22
- USART Bit Definitions, 1-30
- User Mode, 1-6
- User Option Bits, 1-23, 1-29

Section 2

CONSOLE DEBUGGER

BUG-8



TABLE OF CONTENTS

INTRODUCTION	2-3
MEMORY COMMANDS	
The Format Control	2-4
Range	2-6
Displaying Memory Contents	2-7
Altering Memory — Decimal or Octal	2-8
Altering Memory ASCII Format	2-9
REGISTER COMMANDS	
Displaying All Registers	2-10
Displaying Individual Registers	2-10
Altering Register Contents	2-11
EXECUTION CONTROL	
Single Stepping	2-12
Breakpointing	2-13
GO	2-16
TAPE HANDLING	2-17
Tape Errors	2-18
TERMINAL CONTROL CHARACTERS	
Input Control Characters	2-18
Output Control Characters	2-19
COMMAND COMPLETION	2-19
APPENDIX A	
Loading From the Software Distribution Tape	2-20
Loading From a Configured Tape	2-21
APPENDIX B	
Memory Commands	2-22
Range	2-22
Register Commands	2-23
Execution Control	2-24
Tape Handling	2-24
INDEX	2-25



INTRODUCTION

The Heath Console Debugger, BUG-8, is designed to allow you to enter and debug machine language programs from a console terminal. BUG-8 occupies the lowest 3K of RAM, starting at 040 100. A user program can be loaded into any free RAM (random access memory) locations, and can be manipulated via BUG-8.

BUG-8 contains facilities to perform the following nine major functions:

- Display the contents of a selected memory location.
- Alter the contents of a selected memory location.
- Display the contents of any 8080 register.
- Alter the contents of any 8080 register.
- Execute the user program a single instruction at a time.
- Execute the program.
- Insert breakpoints and execute the user program.
- Load user programs from tape storage.
- Dump user program to tape storage.

A number of features were designed into BUG-8 for your convenience. Memory locations and memory and register contents may be displayed as bytes or as words, in octal, decimal, or ASCII format. With these features, you can select the most familiar or desirable format. BUG-8 also contains a single instruction facility that permits you to execute your program a single instruction at a time. And for more advanced program analysis, a breakpointing feature is included that permits you to execute several instructions in a program and then return to control to BUG-8 for analysis and/or modification.

The standard Heath console driver is incorporated in BUG-8; therefore, it responds to all the normal console input and output control characters. For this reason BUG-8 includes error detection and command completion as outlined on Page 2-3 of the "Introduction" to this Software Reference Manual.



MEMORY COMMANDS

The memory commands permit you to display and alter the contents of indicated memory locations. The format for memory display commands is:

< FORMAT CONTROL > < range > < blank >

The form for the alter memory command is:

< FORMAT CONTROL > < range > = < value list >

Format control specifies that memory display/alteration is in word or byte format, and whether octal, decimal or ASCII notation is to be used. The range specifies the memory address or addresses to be displayed or altered, and the command is executed by the typing of a blank using the space bar on the console terminal.

The Format Control

The format control consists of two characters which specify the form of the values that are to be displayed and entered. The format control field may take on a number of different forms. They are:

<u>FORMAT CONTROL</u>	<u>DESCRIPTION</u>
< null > < null >	Display/alter octal integers, byte format.
F < null >	Display/alter as octal integers, word format.
< null > A	Display/alter as ASCII characters, byte format.
FA	Display/alter as ASCII characters, word format.
< null > D	Display/alter as decimal integers, byte format.
FD	Display/alter as decimal integers, word format.



WORD FORMAT (F)

If an F is specified as the first character of the format control field, it indicates that the values are to be displayed/ altered as "full words." This is to say that memory locations are taken as two byte pairs. The second byte is considered to be the high order (most significant) byte and is displayed first. The first byte is considered to be the low order (least significant) byte and is displayed last.

BYTE FORMAT (NULL)

If an F is not specified, the first character is null, indicating that the values are to be displayed/ altered as single bytes. You can create a NULL by not typing any character for the format control portion of the memory command.

OCTAL FORMAT (NULL)

If no option (a NULL) is specified as the second character of the format control field, the values to be displayed/ altered are taken to be octal integers. The NULL was chosen to specify both byte format and octal notation, as byte octal is the most commonly used format. A blank separates each octal integer, or octal integer pair if the F is specified.

DECIMAL FORMAT (D)

If a D is specified as the second character of the format control field, the values to be displayed/ altered are taken to be decimal integers. A blank separates each decimal integer, or decimal integer pair if the F is specified.

ASCII FORMAT (A)

If an A is specified as the second character of the format control field, the values to be displayed/ altered are converted from/to eight bit representations of ASCII characters. A blank separates each character, or character pair if the F is specified.



Range

The range field consists of a beginning address and an ending address. You can specify addresses by using the appropriate offset octal integers; or you can use the NULL, #, and cnt (count) as indicated below.

<u>RANGE FORM</u>	<u>DESCRIPTION</u>
ADDR < null >	Range specifies the single memory location ADDR.
ADDR1-ADDR2	Range specifies the memory locations ADDR1 through ADDR2 inclusive.
ADDR/cnt	Range specifies cnt memory locations starting at location ADDR. NOTE: cnt is a decimal integer ≤ 255 .
#-ADDR	Range specifies the memory locations starting at the beginning of the previous range and ending at ADDR.
#/cnt	Range specifies cnt memory locations starting at the beginning or the previous range. NOTE: cnt is a decimal integer ≤ 255 .
< null>/cnt	Range specifies cnt memory locations starting at the address following the last address of the previous range. NOTE: cnt is a decimal integer ≤ 255 .
< null > -ADDR	Range specifies memory locations starting at the address following the last address of the previous range and extending to memory location ADDR.

For example, to display memory location 000 043 through 000 047, BUG-8 simply requires the user to type 43-47 followed by a blank (a blank is generated by using the console terminal space bar). For example:

B: 43-47 100 112 107 114 100
B:

B: /4 303 053 040 365
B:



NOTE: In the first example, the contents of memory locations 000 043 through 000 047 are displayed on the first line in byte format octal. The next four bytes (locations 000 050 to 000 053) are displayed when the command /4 is typed. The contents of these next four bytes are displayed as soon as a blank is typed after the /4.

If the first address specified is greater than the second address specified, an error message is generated and only the contents of the first memory location are displayed. The form of the error message is:

LWA>FWA

For example:

```
B: 47-43 LWA>FWA  
100  
B:
```

NOTE: If you attempt to enter a numerical address which does not fit the offset octal format, BUG-8 rejects the improper entry and sounds the console terminal bell. For example, the number 067777 does not fit the offset octal format; therefore, BUG-8 does not allow the second 7 to be entered.

Displaying Memory Contents

To display the values in the specified range and in the specified format, type a blank following the format and range fields. BUG-8 immediately executes the command. In the following examples, the contents of a number of locations, 000 043 to 000 070 in the PAM-8 ROM, are displayed in octal byte format, in octal word format, in decimal byte format, in decimal word format, in ASCII byte format, and finally in ASCII word format. NOTE: When all the bytes or words in the specified range cannot be displayed on the line, a new line is started. BUG-8 supplies the starting address of the new line.

```
B: 43-70 100 112 107 114 100 303 053 040 365 257 303 143 002 303 056  
000062 040 076 320 303 235 001 303
```

```
B: F43-70 112100 114107 303100 040053 257365 143303 303002 040056  
000063 320076 235303 303001
```

```
B: D43-70 064 074 071 076 064 195 043 032 245 175 195 099 002 195  
000061 046 032 062 208 195 157 001 195  
B: FD43-70 19008 19527 49984 08235 45045 25539 49922 08238 53310 40387  
000067 49921
```

```
B: A43-70 @ J G L @ + C . >  
B: FA43-70 J@ LG @ + C . >
```



Altering Memory — Decimal or Octal

To alter memory in decimal or octal formats, type an = after the format control and range fields. BUG-8 will then type the value of the first byte, or double byte if an F was used in the format control and follow this with a /. You can then type a new value if you want to change the contents of this location. If the contents of the location are not to be changed, or if sufficient new digits have been entered to complete the change, type a space or a carriage return.

If you type a space, BUG-8 offers the next byte (if there is one in the range) for alteration. If you type a carriage return, BUG-8 returns to the command mode.

In the following example, memory locations 60000 through 60031 are loaded with the octal values of the ASCII characters A through Z. NOTE: On the first three lines, the initial address is followed by the = sign, the current octal value in that memory location, and then a /. The octal value for the letter is entered following the slash. On the successive lines, a range of successive locations are opened and then changed to the sequentially ascending ASCII characters.

After the letters have been entered, the 26 memory locations are examined in byte format as ASCII characters. The 26 locations are then examined in word format as ASCII characters. Note that the second byte is treated as the most significant byte. Finally, the 26 locations are opened in byte octal format, using the # as the first address of the range.

```

B: 60000 = 000/101          (load A)
B: 60001 = 353/102          (load B)
B: 60002 = 341/103          (load C)
B: 60003/23=311/104 357/105 365/106 257/107 062/110 237/111 061/112
060012 303/113 274/114 061/115 315/116 230/117 062/120 012/121 376/122
042/123 302/124 135/125 057/126 120/127 131/130 023/131 046/132

B: A60000/26 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
B: FA#/26 BA DC FE HG JI LK NM PO RQ TS VU XW ZY
B: #-60031 101 102 103 104 105 106 107 110 111 112 113 114 115 116
060016 117 120 121 122 123 124 125 126 127 130 131 132
B;
  
```



The BACK SPACE and RUBOUT keys are not effective when you are entering memory locations. Values placed in memory are taken as modulus 256 numbers (if they are entered in byte format) or as modulus 65,535 numbers (if they are entered in full word format). Thus, if you make a mistake, simply type the correct value with enough leading zeros to cause the bad digit to be eliminated. For example, if byte 70,000 is to be set to 123 and the mis-type 125 occurs, it may be correctly entered as:

```
B: 70000=111/125123  
B: 70000 123
```

NOTE: Only the three least significant digits are accepted for this byte location.

Altering Memory — ASCII Format

To alter memory in ASCII format, type an = after the format control (A for ASCII) and range fields. The processing is similar to decimal or octal format memory alterations. The contents of the opened locations should then be followed by a /. You can then enter the replacement character (or two characters if the word format is used). However, the space and the carriage return are considered to be ASCII character values. To exit the command prematurely, use the ESCAPE or CONTROL-C key to avoid altering a location.

```
B: A70000=/A  
B: A70001=>/B  
B: A70002=/C  
B: A70003-70031=2/D /E /F /G /H /I /J /K" /L /M /N /OW/P#/Q /RZ/S /T  
070024 /U /V8/W /X /Y//Z  
B: 67374-67377 076 000 303 122  
B: A-70031 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z  
B:
```



REGISTER COMMANDS

BUG-8 permits you to display the contents of all registers using octal, decimal, or ASCII, or to display the contents of individual registers using octal, decimal, or ASCII. In addition to displaying the contents of these registers, you can alter the various registers in any of the three modes. NOTE: If the F command is used in the format field, a register command is rejected, as register size is predetermined.

Displaying All Registers

To display the contents of all registers, enter a command of the form.

<FORMAT> <CNTRL-R>

BUG-8 displays the register contents in a specified format. NOTE: An M register is displayed in the all register command and can be specified in other commands. This register is the concatenation of the H and L registers. For example:

```
B: <CNTRL-R>
A=027 B=000 C=000 D=004 E=000 H=041 L=031 F=106 P=043324 M=041031
S=137377
B:

B: D<CNTRL-R>
A=023 B=000 C=000 D=004 E=000 H=033 L=025 F=070 P=09172 M=08473 S=24575
B:

B: A<CNTRL-R>
A= B= C= D= E= H=! L= F=F P=# M!=! S=<
```

A Control R (CNTRL-R) should be typed after each command. However, no character is actually displayed. Also note that the ASCII display is not particularly meaningful unless printing ASCII characters are contained in the desired registers.

Displaying Individual Registers

To display the contents of any single register, use a command in the following format:

< FORMAT > REG < REG-NAME > < blank >



For example, to display the contents of register A, type:

```
B: REGA =101  
B: DREGA =065  
B: AREGA =A  
B:
```

In the above example, the first line calls for the contents of register A to be displayed in octal format. In the second line, the contents of register A are displayed in the decimal format, and in the third line, the contents of register A are displayed in ASCII format. In the following example, the contents of the 16-bit register pair H and L, known as the M or memory register, are displayed in octal format.

```
B: REGM =041031  
B: REGH =041  
B: REGL =031  
B:
```

Altering Register Contents

To alter the contents of a register, use a command in the following format:

```
< FORMAT > REG < REG-NAME > =
```

BUG-8 will then display the previous contents of the register (in the specified format octal, decimal or ASCII), followed by a /. It then accepts a new value if one is typed in. When you are using octal or decimal format, use a carriage return to close the entry or to skip the change. When you are using the ASCII format, type a single ASCII character to close the register. However, as the carriage return is a valid ASCII character, you must use ESCAPE or CONTROL-C to skip the change. The following examples demonstrate the altering of register contents.

B: REGA=102/103	(Change contents of A from 102_8 (ASCII B) to 103_8 (ASCII C).)
B: DREGA=067/066	(Change contents of A from 67_{10} (ASCII C) to 66_{10} (ASCII B).)
B: AREGA=B/C	(Change contents of A from ASCII B to ASCII C.)
B: REGA=103/ ⑧	(A carriage return skips the change.)
B: AREGA=C/ < CNTRL-C >	(A CONTROL-C skips the change.)
B: AREGA=C	(The location is unaltered.)

NOTE: The last three are examples of skipping the change (leaving the location unaltered).



EXECUTION CONTROL

One of the primary functions of BUG-8 is execution control. It lets you step through the program, one or more instructions at a time, while examining register and memory contents. In addition, complete breakpointing is available, permitting you to execute a number of instructions and then return to BUG-8 control to examine register and memory contents. Execution control is divided into the areas of single stepping, breakpointing, and the GO command.

Single Stepping

The form of the single step command is:

STEP ADDR/CNT

where ADDR is an offset octal address (or a null) and “CNT” is a decimal step count, ≤ 255 . If an address is not specified, BUG-8 starts stepping at the current PC-register address. When the instructions are completed, BUG-8 types the PC-register value and returns to the command mode. If an address is specified, BUG-8 starts stepping at the specified address and, when the instructions are completed, displays the terminating address value before returning to the command mode.

The following program increments the contents of memory location 055 100 each time the BC register pair is incremented from 000 000 to 027 000. This program is used to demonstrate a number of the execution control features of BUG-8.

<u>ADDRESS</u>	<u>LABEL</u>	<u>INSTRUCTION</u>	<u>OCTAL</u>	<u>COMMENT</u>
055 000		LXI H	041	Point HL to 055 100.
055 001			100	
055 002			055	
055 003		MVI M	066	Load memory with 000.
055 004			000	
055 005	L2	INX B	003	Increment BC pair.
055 006		MOVA B	170	Load A with B.
055 007		CPI 027Q	376	Compare. Is B at 027 yet?
055 010			027	
055 011		JNZ L2	302	Jump back. Not 027 yet.
055 012			005	
055 013			055	
055 014		INR M	064	Passed 027. Increment memo.



055 015	MOVA M	176	Load A with memory.
055 016	CPI 377Q	376	Compare. Is memory at 377 yet?
055 017		377	
055 020	MVI B	006	Reset B to 000.
055 021		000	
055 022	JNZ L1	302	Jump back. Not at 377 yet.
055 023		005	
055 024		055	
055 025	RST2	327	Memory is 377. Back to BUG-8.

NOTE: The RST2 instruction is used to return this program to BUG-8. When BUG-8 encounters an RST2 instruction, it checks the breakpoint table. If there is nothing set, it returns to BUG-8.

For example, to load the above program using BUG-8,

```
B: 55000-55025=164/041 055/100 376/055 110/066 312/000 252/003 055/170
055007 376/376 103/027 312/302 334/005 055/055 376/064 102/176 312/376
020/377 056/006 376/000 114/302 312/005 101/055 056/327
B:
```

Set the front panel of the H8 to display the BC pair. Zero the BC pair and step through the first 6 program steps using BUG-8 as in the following example. The BC pair will momentarily display 000 001 as the steps are executed.

```
B: REGB=053/000
B: REGC=132/000
B: STEP 55000/6
-P=055005-
```

BUG-8 returns the value of the PC once the first six steps are executed.

Breakpointing

BUG-8 contains several commands to set, display, and clear breakpoints in your program. Breakpointing permits you to execute portions of a program once (or a number of times if the portion of a program is in a loop). Breakpointing is especially useful in de-bugging programs which have a tendency to destroy themselves or obliterate the cause of the problem in the process of complete execution.

SETTING BREAKPOINTS

The breakpoint command is used to set a breakpoint. The form of the breakpoint command is:

```
BKPT ADDR1/CNT1, . . . , ADDRn/CNTn
```



BUG-8 allows up to 6 breakpoints. They are entered in the breakpoint table within BUG-8, replacing any previously defined breakpoints at those addresses. No more than six breakpoints may be entered in the breakpoint table.

The CNT field may be used to specify the breakpoint repeat count. It is a decimal number in the range of 1 to 255. Using the breakpoint count means the breakpoint does not cause control to return to the monitor mode until the breakpoint is executed CNT-1 times. Thus, you may execute a loop a number of times prior to returning to the command mode via a breakpoint instruction. As noted, the Breakpoint Instruction executes CNT-1 times, without recognizing the breakpoint. The last time through the loop, the instruction at the breakpoint address is not executed. The breakpoint returns control to BUG-8. NOTE: If CNT is not specified, the value 1 is assumed.

For example, the program of the previous example is run with the H8 front panel displaying memory location 055 100.

```
B: 55100=001/000
B: BKPT 55015/6
B: GO 55000
-P=055015-
B:
```

NOTE: 055 100 is incremented by 6.

```
B: 55100=006/000
B: BKPT 55015/6,55014/10,55022/30
B: GO 55000
-P=055015-
B: GO
-P=055014-
B: GO
-P=055022-
B:
```

DISPLAYING BREAKPOINTS

To display the current status of the breakpoint table, use the breakpoint display command. BUG-8 can display the contents of the breakpoint table. The form of the breakpoint command is:

```
BKPT DSPLY
```

BUG-8 provides a listing of the current breakpoints in the form:

```
BKPT DSPLY ADDR1/CNT1 ADDR2/CNT2 . . . . . ADDRn/CNTn
```



where ADDR is the address of the breakpoint instruction, and CNT are the loop counts remaining on the designated breakpoints. NOTE: When the breakpoint count is exhausted, it causes control to return to BUG-8. The breakpoint is removed from the breakpoint table, and nonexhausted breakpoints are saved. For example:

```
B: 55100=036/000
B: BKPT 55015/6,55014/10,55022/30
B: BKPT DSPLY 055015/006 055014/010 055022 030
B: GO 55000
-P=055015
B: BKPT DSPLY 055014/004 055022/025
B: GO
-P=055014-
B: BKPT DSPLY 055022/021
B: GO
-P=055022-
B: BKPT DSPLY
B:
```

CLEARING INDIVIDUAL BREAKPOINTS

To clear an individual breakpoint, use the command

```
CLEAR ADDR1, . . . , ADDRn
```

where ADDR1, . . . , ADDRn specifies the address of the breakpoint to be removed from the table.

CLEARING ALL BREAKPOINTS

To complete clear all breakpoints from the breakpoint table, use the breakpoint clear command

```
CLEAR ALL
```

For example:

```
-P=040261-
B: BKPT 55012/10,55014/15,55020/20,55022/200
B: BKPT DSPLY 055012/010 055014/015 055020/020 055022/200
B: CLEAR 55014,55022
B: BKPT DSPLY 055012/010 055020/020
B: CLEAR ALL
B: BKPT DSPLY
B:
```



EXEC

The EXEC (execute) command is a combination of the GO and BKPT commands. The form of the EXEC command is:

```
EXEC SADDR-ADDR1, . . . , ADDRn
```

where "SADDR" is the starting address for execution. If the starting address is omitted, execution starts at the current program counter register value. ADDR1 through ADDRn are the addresses of breakpoints to be set before execution. Thus, for example, to start at byte 055 000 and to execute to byte 055 015, the command is typed as:

```
B: EXEC 55000-55015  
-P=055015-  
B:
```

GO

Use the GO command to transfer control to your program. You can set breakpoints before via the BKPT command. The form of the GO command is:

```
GO [SADDR]
```

If you specify "SADDR," execution begins at this specified address. If you do not specify "SADDR," execution begins at the current value of the program counter register. For example, simple execution of the previous program is accomplished by

```
B: GO  
-P=0550250  
B:
```



TAPE HANDLING

BUG-8 offers three commands for tape handling. With these commands you can dump to a tape, load from a tape, or verify a tape.

DUMP

The DUMP command allows BUG-8 to write a file that is an image of the desired memory range. The operation is very much like the H8 front panel dump. The form of the DUMP command is:

```
DUMP ADDR1-ADDR2
```

ADDR1 and ADDR2 are specified in offset octal. Ready the tape transport before typing the carriage return after ADDR2. The contents of the Program Counter are also saved. Set the PC to the program entry point before you type a return.

LOAD

The LOAD command allows BUG-8 to read a file containing a memory image. The form of this command is:

```
LOAD
```

Once the load is complete, control is returned to BUG-8 for execution or other manipulations. When control is returned to BUG-8, the Program Counter is set to the entry point on the image tape. During a load a tape error may occur. These are explained in the “Tape Errors” section below.

VERIFY

The VERIFY command allows BUG-8 to verify a memory image file. The form of this command is:

```
VERIFY
```

BUG-8 reads the file without loading it and responds with

- OK

if the CRC on the tape matches its computed CRC. If the CRCs do not match, a tape error is generated. (See “Tape Errors.”)



Tape Errors

During a LOAD or VERIFY, one of two error message may be generated. A checksum error is generated if the computed CRC for the record in question does not match the CRC at the start of the record. The form of the checksum error message is

CHKSUM ERR-IGNORE?

A Y in response to the question "ignore?" aborts the error message and the next consecutive record is accepted. NOTE: Ignoring the checksum error is not recommended unless there is no other way to recover the data. If a checksum error is flagged, the chances are very good the data in the designated record is faulty.

A sequence error (SEQ ERR) is generated if the file records are not in the proper sequence. For example, if two record numbers are not consecutive, an error is generated. The form of the sequence error is

SEQ ERR

Typing a space after the SEQ ERR message generates a tape error message and the entire file must be reread.

TERMINAL CONTROL CHARACTERS

BUG-8 recognizes a number of control characters. Some of these are valid when you are entering commands, and others are only valid when BUG-8 is printing information on the console terminal. You can make a control character by simultaneously depressing the appropriate key plus the CONTROL (CNTRL) key. This is similar to depressing the SHIFT key along with a letter.

Input Control Characters

The following characters are only valid when you are entering commands.

BACKSPACE (CONTROL-H)

The BACKSPACE character causes the **last character** on the line to be discarded. A backspace code is sent to the CRT terminal so it can perform a cursor backspace, but the backspace is ignored by teleprinters, and some other printing terminals. If you attempt to BACKSPACE into column zero, a bell code is echoed noting this illegal operation. NOTE: Backspace can be changed when BUG-8 is configured. See "Appendix A," or Page 3 in "Appendix B" in the "Introduction" to this Software Reference Manual.



RUBOUT

The RUBOUT key causes the **current line** to be discarded. A carriage return/line feed is sent to the terminal. Once the RUBOUT is typed, the entire line may be re-entered. NOTE: Rubout can be changed when BUG-8 is configured. See "Appendix A," or Page 2-20 in the "Introduction" to this Software Reference Manual.

Output Control Characters

The following characters are only valid during output.

OUTPUT SUSPENSION AND RESTORATION, CNTRL-S AND CNTRL-Q

Typing Control S during an output suspends the output to the console terminal and suspends program execution. This command is particularly useful when you are using a video terminal, since you can use the Control S or suspend feature each time a screen is nearly filled and information at the top will be lost due to scrolling.

Typing Control Q permits BUG-8 to continue execution and output information to the terminal. The Control Q cancels the Control S function.

THE DISCARD FLAG, CNTRL-O AND CNTRL-P

Typing Control O toggles the DISCARD FLAG. This stops the output on the terminal but program execution does not halt until the program terminates. Typing a Control P (or typing Control O again) clears the discard flag. Control O is often used to discard the remainder of long outputs. Control P clears the discard flag.

COMMAND COMPLETION

When BUG-8 is in the command mode, each terminal keystroke is considered for validity. If the character belongs to no possible command, it is refused and the bell code is echoed to the terminal. If the command syntax allows only one next character, BUG-8 supplies and prints this character for the user.

In addition to simple syntax checking, BUG-8 also processes command range expressions as they are being entered. Should the user enter a range expression referring to nonexistent memory, BUG-8 refuses further entry and echoes a bell code. Thus, should apparently valid characters be rejected by BUG-8, it indicates that the command range expression may be invalid.



APPENDIX A

Loading Procedures

Loading From the Software Distribution Tape

1. Load the tape in the reader.
2. Ready the tape transport.
3. Press LOAD on the H8 front panel.
4. A single beep indicates a successful load.
5. Press GO on the H8 front panel.
6. The console terminal will respond with:

```
HEATH/WINTEK TERMINAL DEBUGGER.  
BUG-8 ISSUE # 02.01.00.  
COPYRIGHT WINTEK CORP., 01/77
```

7. Configure BUG-8 as desired, answering the following questions. Prompt each question by typing its first character on the console terminal keyboard.
 - AUTO NEW-LINE (Y/N)?
 - BKSP = 00008/
 - CONSOLE LENGTH = 00080/
 - HIGH MEMORY = 16383/
 - LOWER CASE (Y/N)?
 - PAD = 4/
 - RUBOUT = 00127/
 - SAVE?
 -
8. Before executing SAVE, have the transport ready at the DUMP port.
9. To use BUG-8 directly from the distribution tape, type the return key at any time rather than a question prompt key. BUG-8 responds:

```
HEATH BUG-8 # 02.01.00.  
B:
```

BUG-8 is ready to use.



Loading From a Configured Tape

1. Load the tape in the tape transport.
2. Ready the tape transport.
3. Press LOAD on the H8 front panel.
4. A single beep indicates a successful load.
5. Press GO on the H8 front panel.
6. The console terminal should respond with:

HEATH BUG-8 # 02.01.00.

B:

BUG-8 is ready to use in the configured form.



APPENDIX B

BUG-8 Command Summary

Memory Commands

Memory display form:

FORMAT CONTROL range =

Memory Alter form:

FORMAT CONTROL range blank

<u>FORMAT</u>	<u>RESULT</u>
< null > < null >	byte octal
F < null >	word octal
< null > A	byte ASCII
FA	word ASCII
< null > D	byte decimal
FD	word decimal

Range

The range field consists of a beginning address and an ending address. You can specify addresses by using the appropriate offset octal integers; or you can use the NULL, #, and cnt (count) as indicated below.

<u>RANGE FORM</u>	<u>DESCRIPTION</u>
ADDR < null >	Range specifies the single memory location ADDR.



ADDR1-ADDR2	Range specifies the memory locations ADDR1 through ADDR2 inclusive.
ADDR/cnt	Range specifies cnt memory locations starting at location ADDR. NOTE: cnt is a decimal integer ≤ 255 .
#-ADDR	Range specifies the memory locations starting at the beginning of the previous range and ending at ADDR.
#/cnt	Range specifies cnt memory locations starting at the beginning of the previous range. NOTE: cnt is a decimal integer ≤ 255 .
< null >/cnt	Range specifies cnt memory locations starting at the address following the last address of the previous range. NOTE: cnt is a decimal integer ≤ 255 .
< null > -ADDR	Range specifies memory locations starting at the address following the last address of the previous range and extending to memory location ADDR.

Register Commands

All registers:

FORMAT CNTRL-R

Single register:

REG REG-NAME blank

Altering register:

REG REG-NAME=



Execution Control

Single stepping:

STEP ADDR/CNT

Breakpointing:

BKPT ADDR1/cnt 1, , ADDRn/cntn (n-6)

Breakpoint display:

BKPT DSPLY

Clearing breakpoints:

CLEAR ADDR1, , ADDRn
CLEAR ALL

GO:

GO (ADDR) (Starts at PC value if ADDR is not specified)

Execute:

EXEC SADDR-ADDR1, , ADDRn (combines GO AND BKPT)

Tape Handling

DUMP ADDR1-ADDR2 @
LOAD @
VERIFY @



INDEX

- ASCII Characters, 2-4
- Load, 2-17
- ASCII Format, 2-9
- Altering Memory, 2-8
- Altering Register, 2-11
- Memory Commands, 2-4
- Backspace, 2-18
- Breakpointing, 2-13
- Breakpoints, 2-15
- Byte Format, 2-4
- Octal Integers, 2-4
- Output Control Characters, 2-19
- Clearing Breakpoints, 2-15
- Command Completion, 2-19
- Range, 2-6
- Register Commands, 2-10
- Rubout, 2-19
- Decimal Integers, 2-4
- Displaying Breakpoints, 2-14
- Dump, 2-17
- Setting Breakpoints, 2-13
- Single Stepping, 2-12
- Exec, 2-16
- Execution Control, 2-12
- Tape Errors, 2-18
- Tape Handling, 2-17
- Format Control, 2-4
- Verify, 2-17
- GO, 2-16
- Word Format, 2-4

Section 3

HEATH TEXT EDITOR

TED-8



TABLE OF CONTENTS

INTRODUCTION	3-3
EDITOR MODES OF OPERATION	3-4
The Command Mode	3-4
The Text Mode	3-4
THE COMMAND STRUCTURE	3-5
Range Expressions	3-5
The Verb	3-10
The Option Field	3-11
The Qualifier String	3-12
THE PARAMETER FIELD	3-12
THE COMMANDS	3-12
TERMINAL CONTROL CHARACTERS	3-25
Input Control Characters	3-25
Output Control Characters	3-26
Tape Errors	3-26
COMMAND COMPLETION	3-27
APPENDIX A	3-28
Loading From the Software Distribution Tape	3-28
Loading From a Configured Tape	3-29
APPENDIX B	3-30
Command Structure	3-30
Range Expression Forms	3-30
Line Expression Forms	3-30
Verb (Command) Forms	3-31
Option	3-33
Qualifier String	3-34
Parameter Field	3-34
Terminal Control Characters	3-34
APPENDIX C	3-35
INDEX	3-37



INTRODUCTION

The Heath H8 Text Editor (TED-8) converts your H8 computer and terminal into a very sophisticated typewriter. This typewriter is not only capable of generating text, but also has powerful editing capabilities. With these capabilities, even a poor typist can create error-free text, organized as desired.

The principle purpose of TED-8 is the preparation of a special form of text for the Heath H8 Assembly Language program called source code. The format for assembly language source code is discussed in Section 4, the Heath Assembly Language Manual (HASL-8).

TED-8 is not restricted to producing source code for assembly language. It can also be used to prepare reports, write letters, and edit manuscripts.

Text is stored in a section of memory called the buffer. All memory not used by the text editor program is available as buffer. Editing can be done by command, referencing the desired line. When the buffer is full, text can be placed onto magnetic or punched paper tape files. Additional text can be read in from previously created magnetic or punched paper tape files, or can be placed in the buffer from the terminal keyboard. TED-8's tape handling capabilities allow it to edit large files on a piecemeal basis.

Slightly more than 4096 bytes of H8 memory are occupied by TED-8 and in addition to its program size, 200 additional bytes of working space are also required. The balance of H8 memory is available for use as buffer. An 8K memory, therefore, has approximately 4K of buffer space, which provides sufficient room for a well documented 300-line program. With less program documentation, more lines of code can be accommodated in the buffer. The same buffer accommodates approximately 175 lines of solid text, such as found in a report or letter.



A compression technique is used in the Text Editor so large quantities of blanks will use very little buffer. This allows you to use sufficient blanks between the source code columns to make the source code easy to read, without using proportionate amounts of memory.

TED-8 has many unique features that are discussed in detail on the following pages. Some of these features are:

- 16 commands for text editing versatility.
- Terminal control of output and input operations.
- Command completion and command error analysis.

EDITOR MODES OF OPERATION

Two modes of operation called the "Command Mode" and the "Text Mode," are available in TED-8. These two modes distinguish between editing commands and text being entered into the buffer.

The Command Mode

The command mode can be subdivided into three areas: input commands, output commands, and editing commands. You execute all commands by typing the appropriate command on the terminal, and follow this with a carriage return; this will be indicated throughout this reference Manual by the symbol \circledcirc . In actual use, no symbol is printed when a carriage return is typed on the terminal, as the carriage or cursor simply moves to the first column of the next line. In the command mode, the prompt character - - (a double dash) appears in the first two columns.

The Text Mode

In this mode you can add text to the buffer from the terminal keyboard, the normal source for most text. But once a source file has been created, it can be stored on a tape file. Later, you can use this tape file as a source of text to be added to the buffer.

Type the ESCAPE (ESC) key when you wish to return from the text mode to the command mode. This performs two functions. First, it deletes the line of text which it was on when the key was struck. Second, it returns the Text Editor to the command mode. To preserve the last line of text, type a carriage return to generate a new line before striking the ESCAPE key. If the console terminal does not have an ESCAPE key, use CONTROL-C.

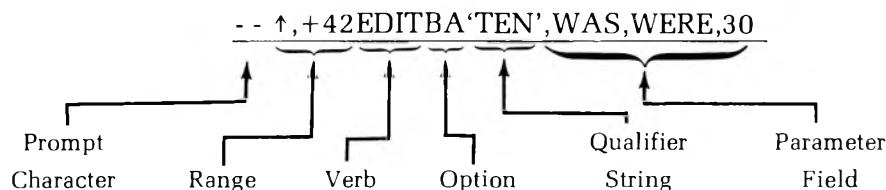


THE COMMAND STRUCTURE

The basic commands for TED-8 have the following form:

```
[<range>] [<verb>] [<option>] [<qualifier string>] [<parameters>]
```

The **range** indicates what lines in the buffer the command affects; the **verb** is the basic command. The **option** permits you to view the line before or after (or both) the command is executed. The **qualifier string** limits the command to those lines containing a given string; and the **parameters** (parameter field) contains specific instructions for some commands. For example, the command



is read as follows:

The lines starting with the first line (`↑`) and ending with the 43rd line (`+42`) are to be edited (EDIT). The EDIT command replaces one string with another. The option (BA) indicates that you wish to view the lines before and after editing. The affected lines are limited to those containing the string "TEN" (the optional qualifier string). And in this particular case, the parameter field specifies the old and new strings and the count. The word WAS is to be replaced by the word WERE, a maximum of 30 times. NOTE: TED-8 strings used in the range expression or qualifier strings must be enclosed in a single quote ('') (apostrophe).

Range Expressions

The range expressions define the buffer lines on which the command is to operate. They may define a single line, or two expressions may be used to define the range over which the command works.

A range expression may consist of:

- A line expression.
- A two-line expression.
- A null.
- A blank.
- An equal sign.



These different range expressions are explained below.

NOTE: Text must be in the buffer before a range expression will be accepted. A bell code will be sound as you try to complete the range expression. To use the following examples, use the INSERT command (see Page 3-12).

SINGLE-LINE EXPRESSIONS

Either one of the following two line expressions may be used to define a single line:

<u>THIS SYMBOL</u>	<u>DEFINES</u>
↑	The first line.
\$	The last line.

TED-8 is always pointing to some line of text. Once you have explicitly pointed to a line by referencing the first line (↑) or the last line (\$), you may make future line references by referencing to the current line pointer. NOTE: Once a DELETE has been executed, the current line pointer is reset and you must explicitly reference a line to reestablish the line pointer.

- A. + count. A plus (+) followed by a decimal number (n) refers to the nth line past the current line pointer.
- B. - count. A minus – (n) refers to the nth line preceding the current line pointer.
- C. + 'string'. The + 'string' refers to the first line in the text buffer which contains the designated 'string' after the current line pointer.
- D. - 'string'. The - 'string' refers to the first line in the buffer, preceding the current line pointer, containing the indicated 'string'.



For example, assume the buffer contains:

```
THIS IS THE FIRST LINE
MARY HAD A LITTLE LAMB
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH LINE
AND EVERYWHERE THAT MARY WENT
THIS IS THE SEVENTH LINE
THE LAMB WAS SURE TO GO
THIS IS THE LAST LINE
--
```

TED-8 responds as follows to the range commands.

```
--↑PRINT
THIS IS THE FIRST LINE
--$PRINT
THIS IS THE LAST LINE
--↑+2PRINT
THIS IS THE THIRD LINE
--↑+'FLEECE'PRINT
ITS FLEECE WAS WHITE AS SNOW
--$-1PRINT
THE LAMB WAS SURE TO GO
--$-'EVERYWHERE'PRINT
AND EVERYWHERE THAT MARY WENT
--
```

MULTIPLE LINE EXPRESSIONS

Use a two-line expression when you want to define a group of lines to be operated on by the command. Use the comma as a delimiter to separate the start line from the stop line. The symbols \uparrow , $\$, +$ count, $-$ count, $+$ 'string', and $-$ 'string' have the same meaning as they do with a single-line command. NOTE: A wide range of combinations may be used to identify the first and last lines of a two-line expression.



For example, you could print the contents of the previous buffer using these commands:

```
--↑,↑+3PRINT
THIS IS THE FIRST LINE
MARY HAD A LITTLE LAMB
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW

--↑+2,+3PRINT
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH LINE
AND EVERYWHERE THAT MARY WENT

--↑+'FLEECE',+1PRINT
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH LINE

--$-'THAT',+'SURE'PRINT
AND EVERYWHERE THAT MARY WENT
THIS IS THE SEVENTH LINE
THE LAMB WAS SURE TO GO

--$-'THAT',+2PRINT
AND EVERYWHERE THAT MARY WENT
THIS IS THE SEVENTH LINE
THE LAMB WAS SURE TO GO

--
```

THE BLANK

When the verb is preceded by a single blank (space), the range is the entire buffer. For example, printing the entire buffer is accomplished by:

```
-- PRINT
THIS IS THE FIRST LINE
MARY HAD A LITTLE LAMB
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH LINE
AND EVERYWHERE THAT MARY WENT
THIS IS THE SEVENTH LINE
THE LAMB WAS SURE TO GO
THIS IS THE LAST LINE

--
```



Note the blank (space) between the prompt character (--) and the word PRINT. The blank is created by typing the space bar on the console terminal.

THE NULL

The NULL expression (the absence of any character) sets a single-line range at the first line of the previous command.

For example:

```
--↑+2,+FIFTH'PRINT  
THIS IS THE THIRD LINE  
ITS FLEECE WAS WHITE AS SNOW  
THIS IS THE FIFTH LINE  
--PRINT  
THIS IS THE THIRD LINE  
--
```

Note that there is no blank (a NULL) between the prompt (--) and the word PRINT.

```
--↑+2,+1PRINT  
THIS IS THE THIRD LINE  
ITS FLEECE WAS WHITE AS SNOW  
--.+2PRINT  
THIS IS THE THIRD LINE  
ITS FLEECE WAS WHITE AS SNOW  
THIS IS THE FIFTH LINE  
--
```

NOTE: In this example, the NULL starts the range at the first line of the previous range, and the ,+2 directs TED-8 to print two additional lines.



THE EQUAL (=)

The = expression sets the range to the range of the previous command. For example:

```
--$-'MARY',+'GO'PRINT  
  
AND EVERYWHERE THAT MARY WENT  
THIS IS THE SEVENTH LINE  
THE LAMB WAS SURE TO GO  
---=PRINT  
AND EVERYWHERE THAT MARY WENT  
THIS IS THE SEVENTH LINE  
THE LAMB WAS SURE TO GO  
--
```

Note that the command =PRINT causes the same lines to be printed in the first and second halves of the example.

The Verb

The verb specifies the action to be taken by the Editor. For example, the command PRINT or the command EDIT are verbs within the text editor's vocabulary.

All verbs are command completed. As soon as the Text Editor receives enough characters to know that only one command is possible, it prints the balance of the command without any additional keys being struck.

The verb will be refused if it is not valid for the current TED-8 condition. For example, an attempt to PRINT an empty buffer is meaningless and the PRINT verb will be rejected.

For example, when you type the P key, the Text Editor knows that no other command begins with P. Therefore, the P is immediately followed by RINT. However, if you type the N, it does not know if the command NEWIN, NEWOUT, or NEXT is to be used. So the Editor prints NE and waits for a W or X. If it receives a W, it then waits for an I or an O. It completes these by filling in the N or UT. If it receives an X following the E, it then prints the T. A complete list of all of the command verbs, and their specific actions and limitations follows in "The Commands" (Page 3-12).



The Option Field

The option field contains characters which let you view the line to be worked on, and/or the line after it has been worked on.

As its name implies, it is completely optional. You may insert any one of the following three option forms, or none at all.

1. B The BEFORE option displays the line **before** the command is executed.
2. A The AFTER option displays the line **after** the command execution.
3. BA This is a combination of the previous two commands.
The line is displayed **before** and **after** command execution.

For example, presume that the buffer erroneously contained

```
THIS IS THE FIRST LINE
MARY HAD A LITTLE LAMB
THIS IS THE THIRD LINE
ITS FLEECE WAS RED AS SNOW
THIS IS THE FIFTH LINE
AND EVERYWHERE THAT MARY WENT
THIS IS THE SEVENTH LINE
THE LAMB WAS SURE TO GO
THIS IS THE LAST LINE
--
```

It may be edited to read correctly by

```
--↑+'RED'EDITBA,RED, WHITE, 1
ITS FLEECE WAS RED AS SNOW
ITS FLEECE WAS WHITE AS SNOW
--
```

Note that after the command EDIT, the options B & A are used to check the work.

NOTE: There are certain times when the command renders the option meaningless. DELETE BA, for example. The AFTER portion of the command is meaningless, as the line (or lines) cannot be displayed after deletion.



The Qualifier String

The qualifier string is a further restriction upon the range expression. It is completely optional. If it is not indicated, it is not used.

The range expression may indicate work over a certain number of sequential lines, starting with a given line. The qualifier string further limits these lines to those within the range containing the string specified in the qualifier field. This string is enclosed in single quotes (') and contains all normal ASCII characters with the exception of the single quote (').

For example, to print the entire buffer (of the previous example), but only those lines with the string 'line':

```
-- PRINT'LINE'  
THIS IS THE FIRST LINE  
THIS IS THE THIRD LINE  
THIS IS THE FIFTH LINE  
THIS IS THE SEVENTH LINE  
THIS IS THE LAST LINE
```

THE PARAMETER FIELD

This is a special field used with the EDIT, TAB, NEWIN, and NEWOUT commands. These commands require additional operating information, which is placed in the parameter field. The nature of this information depends upon the command used. The exact format is discussed under those commands.

THE COMMANDS

The following paragraphs give a complete description of each of the command verbs. Examples of many commands are also given to demonstrate some of the combinations of range expressions, options, qualifier strings, and parameter fields (if required) that may be used with this command. NOTE: All possible combinations of range expressions, options, qualifier strings, and parameter fields are not given. You must review the section for each of these expressions to determine all possible command structures.

INSERT

The INSERT command places TED-8 in the text mode, and is used to add text to the buffer from the console keyboard. This command adds text to the buffer on



the next line following the first line of the range expression. The second line of the range expression is meaningless. Also, if the range expression is given as a line minus a count, the appending point is still the next line following the indicated line. The option, qualifier string, and parameter fields are ignored for the INSERT command. The range command blank INSERT is a special case that is used to insert a line before the first line in the buffer.

For example:

```
-- PRINT
THIS IS THE FIRST LINE
MARY HAD A LITTLE LAMB
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH LINE
AND EVERYWHERE THAT MARY WENT
THIS IS THE SEVENTH LINE
THE LAMB WAS SURE TO GO
THIS IS THE LAST LINE
--

--$INSERT
THIS IS AN ADDITIONAL LAST LINE
<CNTRL-C>
--

--↑+'FIFTH'INSERT
THIS IS A NEW LINE INSERTED AFTER THE FIFTH LINE
--

-- PRINT
THIS IS THE FIRST LINE
MARY HAD A LITTLE LAMB
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH LINE
THIS IS A NEW LINE INSERTED AFTER THE FIFTH LINE
AND EVERYWHERE THAT MARY WENT
THIS IS THE SEVENTH LINE
THE LAMB WAS SURE TO GO
THIS IS THE LAST LINE
THIS IS AN ADDITIONAL LAST LINE
--
```

NOTE: Use the ESCAPE key (or the CONTROL-C) to return to the command mode. This will cause the current line to be erased. If you strike the ESCAPE key after inserting a partial line, that partial line will be lost. Therefore, to preserve the last line of inserted text, type a carriage return (this will create a new blank line in the text buffer) prior to typing the ESCAPE key.



REPLACE

This command causes the eligible line(s) in the command range to be replaced. Once the executing carriage return is typed, the terminal cursor moves to the start of the first line to be replaced. Type the replacement lines one at a time. Tabs, backspaces, and rub-outs may be used as part of the replacement text. However, typing a carriage return signifies replacement of another line within the command range.

After the lines indicated by the range expression have been replaced, TED-8 reverts to the command mode. Typing ESCAPE (or CONTROL-C) returns the editor to the command mode, erasing the current line.

The option and qualifier strings may be used. There is no parameter field for the REPLACE command.

For example:

```
THIS IS THE FIRST LINE
MARY HAD A LITTLE LAMB
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH LINE
THIS IS A NEW LINE INSERTED AFTER THE FIFTH LINE
AND EVERYWHERE THAT MARY WENT
THIS IS THE SEVENTH LINE
THE LAMB WAS SURE TO GO
THIS IS THE LAST LINE
THIS IS AN ADDITIONAL LAST LINE
--

--↑+5REPLACEB
THIS IS A NEW LINE INSERTED AFTER THE FIFTH LINE
THIS LINE IS REPLACED
--

--↑+6,+LAST'REPLACE'LINE'
THIS REPLACES THE OLD SEVENTH LINE
THIS REPLACES THE OLD LAST LINE
--
```



```
-- PRINT
THIS IS THE FIRST LINE
MARY HAD A LITTLE LAMB
THIS IS THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH LINE
THIS LINE IS REPLACED
AND EVERYWHERE THAT MARY WENT
THIS REPLACES THE OLD SEVENTH LINE
THE LAMB WAS SURE TO GO
THIS REPLACES THE OLD LAST LINE
THIS IS AN ADDITIONAL LAST LINE
--
```

Note: Only lines containing the string 'line' are replaced in the second example, as the string 'line' is used as a qualifier.

DELETE

The DELETE command causes the eligible line(s) in the command range to be deleted. You may use the option B; however, the option A is meaningless. You may also use the qualifier string. There is no parameter field accompanying the DELETE command. You may not delete the entire buffer. To do this, use the BLITZ command.

For example:

```
-- PRINT
THIS IS THE FIRST LINE
MARY HAD A LITTLE LAMB
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH LINE
THIS LINE IS REPLACED
AND EVERYWHERE THAT MARY WENT
THIS REPLACES THE OLD SEVENTH LINE
THE LAMB WAS SURE TO GO
THIS REPLACES THE OLD LAST LINE
THIS IS AN ADDITIONAL LAST LINE
--

--$DELETE
--$PRINT
THIS REPLACES THE OLD LAST LINE
--
```



```
--↑,$-1DELETEB
THIS IS THE FIRST LINE
MARY HAD A LITTLE LAMB
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH LINE
THIS LINE IS REPLACED
AND EVERYWHERE THAT MARY WENT
THIS REPLACES THE OLD SEVENTH LINE
THE LAMB WAS SURE TO GO
--PRINT
THIS REPLACES THE OLD LAST LINE
--
```

EDIT

Use the EDIT command to replace one string with another. The string to be replaced and the new string are given in the parameter field of the EDIT command. The parameter field of the EDIT command takes the form:

```
<delimiter> old string <delimiter> new string <delimiter> <count>
```

The **delimiter** is an arbitrary delimiting character. (It may not be a carriage return). The **count** is a decimal count showing the number of replacements to be made. NOTE: Only ONE replacement is made PER LINE. If the count is left null, it defaults to one. If an asterisk (*) is substituted for the count, the value 65,536 is assumed.

The EDIT command makes full use of all range expressions, options, and qualifier strings; and as noted, a specific parameter field.

The following is an example of a text buffer prior to using an EDIT command, and text buffer after the EDIT command is executed.

```
-- PRINT
THIS IS THE FIRST LINE
MARY HAD A LITTLE LAMB
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH LINE
AND EVERYWHERE THAT MARY WENT
THIS IS THE SEVENTH LINE
THE LAMB WAS SURE TO GO
THIS IS THE LAST LINE
--
--EDIT'LINE',LINE,OF MANY LINES,5
--
```



```
-- PRINT
THIS IS THE FIRST OF MANY LINES
MARY HAD A LITTLE LAMB
THIS IS THE THIRD OF MANY LINES
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH OF MANY LINES
AND EVERYWHERE THAT MARY WENT
THIS IS THE SEVENTH OF MANY LINES
THE LAMB WAS SURE TO GO
THIS IS THE LAST OF MANY LINES
```

```
--
```

NOTE: The use of an arbitrary delimiting character permits you to choose a delimiting character not contained in the old or new strings. For example, if a comma (,) is used as a delimiter, it may not appear in either string. If either string contains a comma, you can use a slash (/) as a delimiter.

PRINT

The PRINT command causes the eligible line(s) in the command range to be printed. This is the most common method for viewing portions of the text buffer or the entire text buffer. All forms of the range expressions are utilized. The option commands have no effect. You may use the qualifier string to limit the range expression. There is no parameter field for the PRINT command.

NOTE: While the PRINT command is executing, you may type control characters to aid in viewing the text buffer. See "Terminal Control Characters," Page 3-25.

TAB

The TAB command sets tab stops for entering text. This is especially useful when you are entering source code for assembly programs so the label field, op code field, operand field, and comments field can be clearly separated. TED-8 converts a TAB into an appropriate number of blanks and inserts these into the text. The TAB character itself is not inserted into the text. NOTE: Blanks inserted by a TAB do not use proportional quantities of buffer space. The TAB command uses the parameter field to set the tab stop up to a maximum of six tabs. The format of the parameter field is:

```
TAB,1,....,n
```

The numbers 1 through n specify the columns, by number, which the TAB is to stop. Previously-set tab stops are discarded once the TAB command is called.



Once the tabs are set, you may space out to the next tab by typing control 1. (This is sometimes labeled as TAB on a keyboard.) If no more tabs are available, a bell code is echoed. The TAB command does not use the range field, option field, or qualifier field.

The following example shows setting the tabs and inserting text using these tabs.
 NOTE: The circled I (I) indicates that the user typed a tab. The cursor automatically moves over to the beginning of the new columns. The user did not type in the blanks between columns. This was performed by the TAB command.

```
--TAB,10,20,30,40
--INSERT
*      DETERMINE MEMORY LIMIT

INIT1(I)  MOV(I)      M,A(I)      MOVE BYTE
          D(I)       D(I)       INCREMENT TRIAL ADDRESS
          MOVI(I)    A,M(I)    
          DCR(I)    M(I)      
          CMP(I)    M(I)      
          JNE(I)   INIT1(I)    IF MEMORY CHANGED

INIT2(I)  DCX(I)      H(I)        SET STACK POINTER=MEMORY LIMIT-1
          SPHL(I)   H(I)        SET *PC* VALUE ON STACK
          PUSH(I)   H,I,ERROR
          LXI(I)    H(I)        SET:RETURN ADDRESS:
          PUSH(I)   A,UMI.1B+UMI,18+UMI.16X
          MVI(I)    OUT(I)     OP.TPC(I) SET 8 BIT,NO PARITY,1STOP,X16
          --
          
```

BLITZ

The BLITZ command discards all text in the working buffer. Because of the drastic action this command takes, BLITZ followed by a carriage return results in the question SURE? A Y in response to SURE? proceeds with BLITZ. If you inadvertently type a B, thus causing a BLITZ, you may abort the BLITZ by typing an N or any character except Y. The range option, and qualifier and parameter fields are ignored in a BLITZ command. The BLITZ command **does not** reset tab stops.



USE

The USE command displays the number of lines in the command range, the number of memory bytes currently used, and the number of free bytes. The USE command replies giving three values:

1. A line count. The number of lines within the command range. Type USE to display the total number of lines presently used within the buffer. A null USE results in a single line indication.
2. A byte count. The number of bytes used by the entire working buffer, not simply the lines within the command range.
3. A bytes free count. This is the number of remaining bytes in memory.

You can use the range expression and qualifier strings, but they have little meaning. There is NO parameter field. The following example demonstrates the USE command. The H8 employed in this example has 16 K of memory with the following text:

```
*      DETERMINE MEMORY LIMIT

INIT1  MOV  M,A    MOVE BYTE
       DAD  D      INCREMENT TRIAL ADDRESS
       MOV  A,M
       DCR  M
       CMP  M
       JNE  INIT1  IF MEMORY CHANGED

INIT2  DCX  H
       SPHL     SET STACK POINTER=MEMORY LIMIT-1
       PUSH H   SET *PC* VALUE ON STACK
       LXI  H,ERROR
       PUSH H   SET :RETURN ADDRESS:
       MVIA,UMI.1B+UMI,L8+UMI.16X
       OP.TPC SET 8 BIT,NO PARITY,1 STOP,X16

--
```

NOTE: TED-8 requires some room for work space. It refuses to allow more text when there are still 200 free bytes. These 200 bytes are its work space.



```
--USE
LINES=00017
USED=00338
FREE=11537
--
```

NEWIN

The NEWIN command opens a tape file for reading. It permits you to search a tape that contains a number of files for a particular named file to be used by the Editor. (It is not necessary to have an input tape file to do editing. You may create a new file in the buffer by using the INSERT command.) The name of the text file to be opened is contained in the NEWIN command parameter field. The parameter field for the NEWIN command is in the form:

```
<delimiter> <name> <delimiter>
```

Be sure the tape unit is made ready before you type the carriage return. TED-8 will then scan the tape until it finds the named text file.

Note: A file is acceptable if the leading characters in its name match all the characters supplied in the NEWIN parameter field. In other words, the full name need not be supplied. Thus, supplying null as a name (simply two delimiter characters together) allows a match on the first text file found.

After the label record is found, the tape stops before the first block of text. TED-8 responds by indicating the name of the file that has been found. Use a FILL or READ command to input text from this file. The NEWIN command does not use range expressions, options, or qualifier strings.

NOTE: If an input file is opened but not read to the end-of-file record, the NEWIN command asks for a go-ahead prior to searching for a new file. The reply Y, to SURE?, instructs NEWIN to proceed to find the new file.

For example:

```
--NEWIN"USR"
OLD "IN" NOT FINISHED. SURE?YFOUND USR PROGRAM FOR BASIC #1
--
```

FILL

This command fills the buffer with text from an input file opened by the NEWIN command. FILL causes successive records of text to be read from the input tape until:

1. An end-of-file is read; or
2. Less than 512 bytes (1 block) of free buffer space is left.



The FILL command ignores range, expressions, options, and qualifier strings. It does not use the parameter field. When the prompt returns, FILL is complete.

READ

The READ command is used to input one **record** of text from the tape. If insufficient buffer room exists to hold a record of text, an error message is given. In this situation, you must first empty the buffer before the READ command can be executed. The range option expression and qualifier fields are ignored. The READ command does not use the parameter field.

Text inputted by the READ command is appended to the working buffer. NOTE: The READ command differs from the FILL command, as the READ command only inputs one record from the tape. When the prompt returns, READ is complete.

NEWOUT

The NEWOUT command is used to open a new output file. The file name is supplied in the parameter field, in the same manner as in the NEWIN command. The form of the parameter field is:

```
<delimiter> <name> <delimiter>
```

The output tape should be readied before you type the carriage return. TED-8 will write a label record on the tape and then stop it, leaving it ready for text. Use the WRITE, FLUSH, or SAVE commands, as appropriate, to write text onto the tape.

The NEWOUT command does not use range expressions, options, or qualifier strings.

If you use NEWOUT without closing a previously opened file, NEWOUT responds with SURE? A reply of Y permits NEWOUT to write the label for the new file without closing the previous file. **WARNING:** This procedure results in a partially complete file being left on the output tape. The FLUSH or STOP OUTPUT commands are used to write an end-of-file.

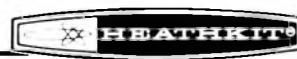
WRITE

The WRITE command outputs text from the buffer onto tape. It starts at the top of the buffer and continues to the first line of the command range. Thus, the command

```
$WRITE
```

writes the entire buffer.

This command uses range expressions, options, and qualifier strings. It has not parameter field. **NOTE: After lines are written, they are deleted from the buffer.**



FLUSH

The FLUSH command is used when editing is complete. It causes the working buffer to be written to the output tape, and then any remaining text on the input tape is copied over to the output tape without modification. The FLUSH command then writes an EOF (end-of-file) record on the output tape. This EOF record is needed for subsequent reading of the output file. The FLUSH command does not use range expression, options, qualifier strings, or parameter fields. **NOTE: After the lines are written, they are deleted from the buffer.**

NEXT

The NEXT command performs the following sequence:

```
$WRITE  
FILL
```

This command causes all the text in the buffer to be written to the output tape, and then refills the buffer from the input tape. This command is particularly useful when you are generating long tape files or when you perform minor editing on long tape files. The NEXT command does not use a range expression, options, or qualifier strings. It has no parameter field.

SAVE

The SAVE command outputs text from a working buffer onto tape. The SAVE command starts at the first line in the buffer and continues to the first line of the command range. Thus the command

```
$SAVE
```

saves the entire buffer.

The SAVE command uses range expressions, options, and qualifier strings. It has no parameter field. **NOTE: After the buffer is saved, you can use a VERIFY to be sure the tape record contains the desired information before you erase the buffer.** SAVE is identical to WRITE except the buffer is not deleted after SAVE outputs to the tape. **NOTE: The command SAVE (no range field) will only save one line.**

STOP

There are two forms of the STOP command. They are:

```
STOP INPUT
```

and

```
STOP OUTPUT
```



The STOP INPUT command sets the flag in TED-8, which indicates that an EOF has been read. Once TED-8 executes a STOP INPUT, the previously opened input file is presumed closed.

The STOP OUTPUT command instructs TED-8 to write an EOF record on the current output tape. STOP OUTPUT is used following a \$SAVE or \$WRITE command to close the output file.

SEARCH

The SEARCH command scans through a text file for a given character string. The desired character string is specified in the qualifier field. This command begins the scan at the first line in the command range and continues to the end of the buffer (regardless of the last line in the command range). If the given character string is still not found, the buffer is written onto the output tape and more text is added from the input file. The SEARCH stops when an EOF is read, or when the string is found.

When the string is found, the command range is set to that line. Subsequent commands can reference the line containing the desired string by using an equal (=) for the range expression. The SEARCH command uses the range expression, but does not use option or parameter fields.

For example:

```
-- PRINT
THIS IS THE FIRST LINE
MARY HAD A LITTLE LAMB
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTHE LINE
AND EVERYWHERE THAT MARY WENT
THIS IS THE SEVENTH LINE
THE LAMB WAS SURE TO GO
THIS IS THE LAST LINE
--
--SEARCH"FIGTHE"
---EDITBA,FIFTHE,FIFTH,
THIS IS THE FIFTHE LINE
THIS IS THE FIFTH LINE
--
```

NOTE: The given character string is enclosed in double quotes ("") not single quotes ('').



VERIFY

The form of the VERIFY command is:

```
VERIFY n
```

where n is the number of records to be verified. If no value is specified, VERIFY verifies to the last record written and stops the tape (a record written by a SAVE for example). In either case, VERIFY stops when an EOF is read.

The VERIFY command is used to check a mass storage record without loading it in the buffer. The VERIFY command checks the sequence and CRC of each record in the file. If a valid ASCII data file is found preceded by a label record, VERIFY responds with

```
RECORD #000 OK
RECORD #001 OK
RECORD #002 OK      etc.
--
```

NOTE: Record #000 is the label record that contains the file name, etc. All other records are compressed ASCII records containing text. The number of records depends on the length of the file.

XPORT

The XPORT command allows you to specify an I/O port other than the Console Terminal for the INSERT (XINSERT) and PRINT (XPRINT) commands. The form of the XPORT command is:

```
XPORT, nnn
```

where nnn is the decimal number of the desired alternate I/O port. When this command is executed, TED-8 responds:

```
CONFIGURE PORT: GO'
```

and the H8 audio alert is sounded, indicating a return to monitor. You must configure the alternate Port UART and press GO on the H8 front panel to restart TED-8. For example, to configure port 376₈ (254₁₀):

1. Type XPORT, 254 cr.
2. TED-8 Responds.

```
CONFIGURE PORT: GO'
```

3. Enter 377 116 using the MEM key (316 for a 110 baud device).
4. Press OUT.



5. Enter 377 005 using the MEM key.
6. Press OUT.
7. Press GO.

Port 376_8 (377_8 is the Mode/Command word for data port 376) is configured with a mode of 116_8 (eight bit word with a single stop bit) and with a mode of 005 (T \times EN and R \times E). TED-8 is returned to the operational mode. This port is now configured until an RST command is executed by PAM-8. Once configured, you may use XPRINT and XINSERT as desired.

XINSERT, nnn

The XINSERT command is identical to the INSERT command except that you insert text from port nnn, which you previously configured by XPORT. The XINSERT command is useful for loading text in full ASCII (not compressed), such as that found on paper tapes used with a teleprinter.

XPRINT

The XPRINT command is identical to the PRINT command except that the Output is directed to port nnn, which you previously configured by XPORT. The XPRINT command is used to divert output to a line printer or other similar device.

TERMINAL CONTROL CHARACTERS

TED-8 recognizes a number of control characters. Some of these are valid when you are typing into the TED-8, and others are only valid when TED-8 is printing information on the terminal or line printer. You can make a control character by simultaneously depressing the appropriate key plus the CONTROL (CNTRL) key. This is similar to depressing the SHIFT key along with a letter.

Input Control Characters

The following characters are only valid when you are inputting.

BACKSPACE (CONTROL-H).

The BACKSPACE character causes the **last character** on the line to be discarded. A BACKSPACE code is sent to the terminal so CRT terminals can perform a cursor BACKSPACE. If you attempt to BACKSPACE into column zero, a bell code is echoed noting this illegal operation. BACKSPACE is valid in the command and text modes.

BACKSPACE can be changed when the Text Editor is configured. See "Appendix A," or "Product Installation" on Page 0-19 of the "Introduction" to this "Software Reference Manual."



NOTE: The BACKSPACE is ignored by teleprinters and some other printing terminals.

RUBOUT

The RUBOUT key causes the **current line** to be discarded. A carriage return/line feed is sent to the terminal. Once the RUBOUT is typed, the entire line may be re-entered. RUBOUT is valid in the command and text modes. NOTE: RUBOUT can be changed when the text editor is configured. See "Appendix A," or "Product Installation" on Page 0-19 of the "Introduction to this "Software Reference Manual."

TAB (CONTROL I)

The TAB character is valid only when you are adding text to the buffer from the keyboard. It is used with the INSERT and REPLACE commands. Typing TAB (CONTROL I) causes the cursor to advance to the next tab column. If there is no next tab column, a bell code is echoed. TED-8 converts TAB's into the equivalent number of blanks and stores these in compressed form.

Output Control Characters

The following characters are only valid during execution of the PRINT command.

OUTPUT SUSPENSION AND RESTORATION CNTRL-S AND CNTRL-Q

If you type CONTROL S during an output, it suspends the output to the terminal by setting the suspend flag.

Typing CONTROL Q permits TED-8 to continue execution and outputting information to the terminal. The CONTROL Q clears the suspend flag.

THE DISCARD FLAG, CNTRL-O AND CNTRL-B

If you type a CONTROL O, it toggles the DISCARD FLAG. This stops output on the terminal but does not halt program execution until the program terminates. Typing CONTROL O again toggles the discard flag, resuming output. Type a CONTROL P to clear the discard flag. CONTROL O is often used to discard the remainder of long listings and other similar outputs.

Tape Errors

During a NEWIN, FILL, READ, or VERIFY, either one of the following two error messages may be generated:

SEQ ERR
CHKSUM ERR.



A sequence error (SEQ ERR) is generated if the file records are not in the proper sequence. For example, if two record numbers are not consecutive, an error is generated. The form of the sequence error is

SEQ ERR

Typing a blank after the SEQ ERR message generates a tape error message and the entire file must be reread.

A checksum error (CHKSUM ERR) is generated if the computed CRC for the record in question does not match the CRC recorded at the start of the record. The form of the checksum error message is

CHKSUM ERR IGNORE?

A Y in response to the question "ignore" aborts the error message and the next consecutive record is processed. NOTE: Ignoring the checksum error is not recommended unless there is no other way to recover the data. If a checksum error is flagged, the chances are very good that the data in the designated record is faulty, and may cause TED-8 to crash.

COMMAND COMPLETION

When TED-8 is in the command mode, each terminal keystroke is considered for validity. If the character belongs to no possible command, it is refused and the bell code is echoed to the terminal. If the character is accepted and the command syntax allows only one next character, TED-8 supplies and prints this character for you.

In addition to simple syntax checking, TED-8 processes command range expressions as they are being entered. If you should enter a range expression referring to nonexistent lines, TED-8 refuses further entry and echoes a bell code. Thus, should valid characters be rejected, it indicates that the command range expression is invalid. For example, if you should attempt to type

-↑+ 'TUESDAY'

and TED-8 refuses the "P" in PRINT, it means that it found no line containing Tuesday. NOTE: This is done before the command was executed, so you can use a backspace to eliminate the TUESDAY and replace this string with a string valid for the text. NOTE: If no information exists in the text buffer, TED-8 will not accept commands which need text to be valid. For example, the command PRINT is invalid when no text exists in the text buffer, as there is nothing to print.



APPENDIX A

Loading Procedures

Loading From the Software Distribution Tape

1. Load the tape in the reader.
2. Ready the tape transport.
3. Press LOAD on the H8 front panel.
4. A single beep indicates a successful load.
5. Press GO on the H8 front panel.
6. The console terminal will respond with:

HEATH/WINTEK H8 EDITOR
TED-8 #3.01.00.
COPYRIGHT WINTEK CORP.,01/77

7. Configure TED-8 as desired, answering the following questions. Prompt each question by typing its first character on the console terminal keyboard.
 - AUTO NEW-LINE (Y/N) ?
 - BKSP=00008/
 - CONSOLE LENGTH=00080/
 - HIGH MEMORY=16383/
 - LOWER CASE(Y/N) ?
 - PAD=4/
 - RUBOUT=00127/
 - SAVE? .



8. Before executing SAVE, have the tape transport ready at the DUMP port.
9. To use TED-8 directly from the distribution tape, type the RETURN key at any time rather than a question prompt key. TED-8 responds

HEATH TED-8 #3.01.00.
--

The Text Editor is ready to use.

Loading From Configured Tape

1. Load the tape in the tape transport.
2. Ready the tape transport.
3. Press LOAD on the H8 front panel.
4. A single beep indicates a successful load.
5. Press GO on the H8 front panel.
6. The console terminal responds with:

HEATH/WINTEK TEXT EDITOR #3.01.00.
--

The Text Editor is ready to use in the configured form.



APPENDIX B

Command Summary

Each of the commands for the Heath Text Editor are summarized in this Appendix. For a detailed explanation of each command, refer to "The Commands," Page 3-12 to 3-24, and to "Terminal Control Characters," Page 3-25.

Command Structure (Page 3-5)

The general form for an editor command is:

[<RANGE EXPRESSION>] [<VERB>] [<OPTION>] [<QUALIFIER STRING>] [<PARAMETERS>]

Range Expression Forms (Page 3-5)

1. NULL — First line in previous command range.
2. BLANK — The entire working buffer.
3. = — The previous command range.
4. A single line expression.
5. Two-line expressions.

Line Expression Forms (Pages 3-6 and 3-7)

EXPRESSION	DEFINITION
1. ↑	The first line in the buffer.
2. \$	The last line in the buffer.
3. NULL	The first line in the previous command range.
4. COMMA	Separates multiple line expressions.
5. +COUNT	Move forward count decimal lines.
6. -COUNT	Move backward count decimal lines.
7. +'STRING'	Move forward until 'STRING' is located.
8. -'STRING'	Move backward until 'STRING' is located.



Verb (Command) Forms

INSERT	Add to text buffer from keyboard. ESCAPE returns Editor to command mode (Page 3-12).
REPLACE	Replace eligible lines in command range from keyboard. ESCAPE returns Editor to command mode (Page 3-14).
DELETE	Deletes eligible lines in command range (*except entire buffer) (Page 3-15).
EDIT	Replaces old string with new string once per line. Parameter field: <arbitrary delimiter> old string <arbitrary delimiter> count. Count is decimal number of replacements (Page 3-16).
PRINT	Print eligible lines on console terminal (Page 3-17).
TAB	Sets TAB stops. Parameter field format is Tab 1, . . . , n. 1-n are up to 6 column numbers (Page 3-17).
BLITZ	Discards all text after a Y reply to SURE ? (Page 3-18).
USE	Displays number of lines in buffer, bytes used, and bytes free (Page 3-19).
NEWIN	Opens a new tape file to be read in. Parameter field specifies file name <delimiter> <name> <delimiter> (Page 3-20).



FILL	Fills the working buffer with text from input tape. Stops at end-of-file or less than 512 free bytes (Page 3-20).
READ	Reads one additional block (512 bytes) from input tape (Page 3-21).
NEWOUT	Opens a new tape file for output. Parameter field specifies file name <delimiter> <name> <delimiter> (Page 3-21).
WRITE	Writes text from the working buffer to output tape. Writes start of buffer to first line of common range. After writing, lines are deleted (Page 3-21).
FLUSH	Writes working buffer, balance of input file, and end-of-file character onto output tape. Use when editing is complete. Text is deleted when complete. (Page 3-22).
NEXT	Writes working buffer onto output tape. Then fills buffer from input tape (Page 3-22).
SAVE	Saves text from the working buffer on the output tape. Saves start of buffer to first line of command range. Buffer is not deleted (Page 3-22).
SEARCH	Searches text buffer and input tape after initial line for 'STRING'. Search stops when STRING is found or end-of-file is found. Command range set to line containing 'STRING' (Page 3-23).



VERIFY	Verifies the contents of file 'name.' Performs a CRC and a sequence test on each record and compares this computed CRC to the recorded CRC (Page 3-24).
XINSERT	Add to the Text buffer from the XPORT. Text at the load port is uncompressed ASCII. ESCAPE returns TED-8 to the command mode (Page 3-24).
STOP INPUT	Sets the EOF flag, indicating a closed file (Page 3-23).
STOP OUTPUT	Writes an EOF record to tape. Closes current output file (Page 3-23).
XPORT, nnn	Initializes port nnn (Page 3-24).
XPRINT, nnn	Diverts printing to line printer at port nnn (Page 3-25).

Option (Page 3-11)

The option field is:

- B Print line before operating on it.
- A Print line after operating on it.
- BA Print line before and after operating on it.
- NULL No option specified.



Qualifier String (Page 3-12)

Qualifier string, if present, takes the form 'string'. A qualifier string limits range expression to those lines containing the qualifier string.

Parameter Field (Page 3-12)

This field contains extra parameters needed by the EDIT, TAB, NEWIN, and NEWOUT commands. Format is command dependent.

Terminal Control Characters (Page 3-25)

BACKSPACE (CONTROL H)	Deletes one character in command and text modes.
RUBOUT	Deletes line in command and text modes.
TAB (CONTROL-I)	Advances cursor to next TAB column.
CONTROL S	Sets suspend output flag.
CONTROL Q	Resets suspend output flag.
CONTROL O	Toggles discard output flag.
CONTROL P	Resets discard output flag.



APPENDIX C

The following edited source file is typical of one you might create using TED-8. It is intended to show examples of some of TED-8's editing capabilities. The assembly of this example is shown on Page 4-53 and it is used as a special BASIC function in "Appendix E" of Section 5.

```
-- PRINT
      ORG    100000A-160Q
FPNRM  EQU    073173A
          INX    B      INC
          INX    B      TO
          INX    B      EXPONENT
          LDAX   B
          ANA    B      SET CONDX CODE
          RZ
          DCR    A      /2
          JZ     USRI   IF UNDER FLOW
          DCR    A      /2 again (/4)
USRI   STAX   B
          CALL   FPNRM
          RET
          END    START

--↑+'EQU'INSERTB
FPNRM  EQU    073173A
START   INX    B      INC
--↑+'073173A'EDITBA,073173A,063207A,1
FPNRM  EQU    073173A
FPNRM  EQU    063207A
-- PRINT
      ORG    100000A-160Q
FPNRM  EQU    063207A
START   INX    B      INC
          INX    B      TO
          INX    B      EXPONENT
          LDAX   B
          ANA    B      SET CONDX CODE
          RZ
          DCR    A      /2
          JZ     USRI   IF UNDER FLOW
          DCR    A      /2 again (/4)
USRI   STAX   B
          CALL   FPNRM
          RET
          END    START
```



```

--↑+' EXPONENT' INSERTB
    INX      B      EXPONENT
    LDAX     A      (A)=ACCX EXP
--' ACC'+1DELETEB
    LDAX     A
--↑+' /4' INSERTB
    DCR      A      /2AGAIN(/4)
    USRI    STAX   B      RET TO ACCX
--' ACCX'+1DELETEB
    USRI    STAX   B
--' ACCX' INSERTB
    USRI    STAX   B      RET TO ACCX
    CALL    FPNRM  NORMALIZE
--' NORMALIZE'+1DELETEB
    CALL    FPNRM
-- PRINT
    ORG    120000A-160Q
FPNRM  EQU    063207A
START   INX    B      INC UP
        INX    B      TO
        INX    B      EXPONENT
        LDAX   B      (A)=ACCX EXP
        ANA     A      SET CONDX CODE
        RZ
        DCR     A      /2
        JZ      USRI   IF UNDER FLOW
        DCR     A      /2AGAIN(/4)
USRI    STAX   B      RET TO ACCX
        CALL   FPNRM  NORMALIZE
        RET
END     START

--NEWOUT"USR PROGRAM FOR BASIC #1.0"
--$SAVE
--STOP OUTPUT
SURE?Y
--VERIFY,5
RECORD #000 OK
RECORD #001 OK
RECORD #002 OK
*EOF*
--
```



INDEX

- After A, 3-11
Arbitrary Delimiting Character, 3-17

Backspace, 3-25
Before (B), 3-11
Blank, 3-8
Blitz, 3-18
Buffer, 3-3

Checksum Error, 3-27
Command Completion, 3-27
Command Mode, 3-4
Command Structure, 3-5 ff.
Command Summary, 3-30
Compression (Blanks), 3-4
Control Characters, 3-25
Control C, 3-4, 3-13, 3-14
Control H, 3-25
CNTRL-O, 3-26
CNTRL-P, 3-26
CNTRL-Q, 3-26
CNTRL-S, 3-26

Delete, 3-15

Equal (=), 3-10
Escape key, 3-4, 3-13, 3-14

Fill, 3-20
Flush, 3-22

Insert before first line, 3-13
Insert, 3-12

Loading TED-8. 3-28 ff,

Modes, 3-4
Multiple Line Expression, 3-7

Newin, 3-20
Newout, 3-21
Next, 3-22
Null, 3-9

Option Field, 3-11

Parameter Field, 3-12
Print, 3-17

Qualifier String, 3-12

Range Expressions, 3-6, 3-12
Read, 3-21
Replace, 3-14
Rubout, 3-26

Save, 3-22
Search, 3-23
Sequence Error, 3-27
Single-Line Expressions, 3-6
Stop, Input, 3-23
Stop, Output, 3-23
Sure, 3-18, 3-20

Tab, 3-17, 3-26
Tape Errors, 3-26
Text Mode, 3-4

Use, 3-19

Verb, 3-10
Verify, 3-24

Write, 3-21

XINSERT, 3-25
XPRT, 3-24
XPRINT, 3-25

SECTION 4

HEATH ASSEMBLY LANGUAGE

HASL-8



TABLE OF CONTENTS

WRITING H8 ASSEMBLY LANGUAGE PROGRAMS	4-3
THE CHARACTER SET	4-4
STATEMENTS	4-4
The Label Field	4-5
The Opcode Field	4-5
The Operand Field	4-6
The Comment Field	4-6
Format Control	4-7
OPERAND EXPRESSIONS	
Operators	4-7
Tokens	4-8
THE 8080 OPCODES	4-10
Terms, Symbols, & Nomenclature	4-11
Data Transfer Group	4-17
Arithmetic Group	4-21
Logical Group:	4-28
Branch Group	4-34
Stack, I/O, and Machine Control Group	4-38
PSEUDO OPCODES/ASSEMBLER DIRECTIVES	4-43
Define Byte, DB	4-44
Defined Space, DS	4-44
Define Word, DW	4-45
Conditional Assembly Pseudo Operators	4-45
Listing Control	4-47
USING THE ASSEMBLER	4-50
Errors	4-54
Control Characters	4-56
APPENDIX A	4-57
Loading From the Software Distribution Tape	4-57
Loading From a Configured Tape	4-58
Index	4-59



WRITING H8 ASSEMBLY LANGUAGE PROGRAMS

The Heath Assembly Language program (HASL-8) lets you use source (symbolic) programs using letters, numbers, and symbols that are meaningful as they are abbreviated in English statements. These source programs must be generated with the Heath Text Editor (TED-8).

Heath Assembly Language assembles the source program into a listing and an object program in absolute binary format executable by the H8 Computer. The listing and the object program are produced after two passes through the assembler (the assembler must read the source tape twice before it is able to produce the absolute binary output). If there are sufficient memory locations in the host computer, the binary program may be stored directly in memory. If there is not room, as is often the case when you are assembling large programs, the object program is written onto a mass storage device. During the second pass, HASL-8 produces a complete octal/symbolic listing of the assembled program. This listing is especially useful for documentation and debugging purposes.

This Manual assumes that you are already familiar with the writing of assembly language programs. Also, because of the many cross-references in this Section, we recommend that you read all of this Section to get a good "feel" for HASL-8. If you are not familiar with assembly language programming, we recommend that you study material such as the Heathkit Continuing Education "8080 Programming" course.

HASL-8 is designed to produce programs which run in an H8 system; therefore, it assembles 8080 symbolic assembly code. It requires about 8K of memory (depending upon the number of symbols defined during assembly) and one mass storage unit. Separately controllable mass storage readers and recorders may be necessary for large assemblies. This can take the form of one paper tape/reader punch or two independent cassette recorders. When used with 8K of memory, HASL-8 provides for approximately 250 user-defined symbols.

This Software Reference Manual presumes that you have read the H8 operation Manual and are familiar with the 8080 instruction set, I/O formats, memory formats, and front panel configuration. A thorough knowledge of these facts is vital to efficient assembly language programming.



THE CHARACTER SET

The Heath Assembly Language source program is composed of symbols, numbers, expressions, symbolic instructions, argument separators, assembly directives, and line terminators, all using ASCII characters. Those characters that are acceptable to HASL-8 are listed below.

1. The letters A through Z (lower case letters are acceptable for quoted strings and comments, provided that HASL-8 is configured properly in accordance with the software configuration guide).
2. The numerals 0 through 9.
3. The characters period (.) and dollar sign (\$), which are considered alphabetic.
4. The symbols

: = % # () , ; " ' + - _ ! ?

LINE FEED AND CARRIAGE RETURN

STATEMENTS

A source program is composed of a sequence of statements, designed to solve a problem. Each statement must be on a single line.

A statement is composed of up to four fields, identified by the order of appearance and by separating characters. The four fields are:

LABEL OPCODE OPERAND COMMENT

The label and comment fields are optional. The opcode and operand fields are interdependent; either may be omitted, depending upon the contents of the other.



The Label Field

The label field always starts in column one. A label is a user-defined symbol assigned the current value of the memory location counter. It is a symbolic means of referring to a specific memory location within a program. Most statements do not require a label. If you do not want a label, column one must be left blank. Although the label is usually used to allow symbolic reference to the address of the labeled instruction, the SET and EQU pseudos make special use of the label field.

A label must start with an alphabetic character, and it consists entirely of alphabetic or numeric characters. The maximum length of a label is 7 characters. Note that the characters "\$" and ":" are considered alphabetic. Therefore, the following are valid labels.

```
A A3 C9D4 .START .. $END END$PGM
```

For example, if the current location counter is set to 040 100 and the statement

```
.START MOV A,B
```

is the next statement, the assembler assigns the value 040 000 to the label .START. Subsequent references to .START refer to location 040 100.

The Opcode Field

All statements (except the comment statements) must have an opcode field. The opcode field need not be located in any particular column. However, it must be separated from the label field by at least one blank. If no label is specified, the opcode field may start in or after column 2.

The opcode is either an instruction mnemonic or an assembler directive. When the entry in the opcode field is an instruction mnemonic, it specifies a machine operation to be performed on any following operands. When it is an assembler directive, it specifies certain functions or actions to be performed by the assembler during program assembly.

The opcode field is terminated by a blank or by the end of a line.



The Operand Field

The operand field follows the opcode field and must be separated from it by at least one blank. Not all opcodes require operands. The operand contains information used by a machine instruction or, in the case of assembler directives (pseudo opcodes or pseudo ops), it contains information to be used by the pseudo op.

Operands may be symbols, expressions, or numbers. When multiple operands appear with a statement, each is separated from the next by a comma. An operand may be followed by a comment.

The operand field is terminated by a blank when followed by a comment, or by the end of a line when the operand ends the assembly statement. For example,

```
.START    MOV A,B  THIS IS A COMMENT
```

The space between .START and MOV terminates the label field; the blank between MOV and A,B terminates the opcode field and begins the operand field. The comma separates the operands A and B and the blank terminates the operand field and begins the comment field.

The Comment Field

The comment field follows the operand field, or the opcode field if no operand field is present. It must be separated from its preceding field by at least one blank. The comment field is not processed by the assembler and it is designed to contain documentary information. The comment field is optional and may contain any printing ASCII character. All other characters, even those with special significance to the assembler, are ignored by the assembler when used in the comment field.

A statement with an asterisk (*) in column one is taken as a comment statement and is not otherwise processed by the assembler. A totally blank line is also taken as a comment.



Format Control

The format of an assembly language program is controlled by the blank character. Format control is primarily used to produce a program which is easily read. Format control has no affect on the assembly process of the source program, and because HASL-8 uses compression techniques, the use of multiple blanks does not take up extra memory space. The following two statements are identical with the exception that the first one does not use any tabs and the second one uses tabs.

```
.START MOV A,B THIS IS A COMMENT  
.START    MOV A,B    THIS IS A COMMENT
```

The TABs were converted to the appropriate number of blanks by TED-8. Therefore, HASL-8 sees no TAB characters.

OPERAND EXPRESSIONS

Except when the opcode is a machine instruction requiring that an 8080 register be specified as the operand, all operand fields may be coded as operand expressions. Such operand expressions are made up of integers, symbols, a special origin symbol, and character strings which may be combined, using certain operators. The operand may also be the origin symbol. The expressions are said to be made up of operators and tokens. No parentheses are allowed nor is any operator precedence recognized. Therefore, evaluation is strictly left to right. The result of any expression must fall between -32,767 and 65,534.

Operators

HASL-8 recognizes 5 operators. They are:

- + Addition of an integer arithmetic expression.
- Subtraction of an integer arithmetic expression.
- * Multiplication of an integer arithmetic expression.
- / Division of an integer arithmetic expression.
- (unary) negation of a standard integer arithmetic expression.



Note, the unary minus is valid only as the first character in an expression. The following are examples of legitimate assembler operand expressions.

$3 + 5$
 -2 (unary)
 $1 + 2 * 3$

Note that the last example evaluates to 9 rather than 7, as the **assembler does not recognize any operator precedence**. Therefore it evaluates the expression from left to right.

Tokens

Heath Assembly Language recognizes four different tokens: integers, symbols, character strings, and the origin symbol. Each of these tokens has the limitations described in the following sections.

INTEGERS

Decimal integers ranging from 0 to 65,535 are allowed, but no decimal place may be specified. The radix of an integer expressions is assumed to be decimal. However, you may specify binary, octal, offset octal, decimal, or hexadecimal. Specify them by using a post-radix symbol following the integer expression.

B	Binary
O or Q	Octal
D	Decimal
H	Hexadecimal
A	Offset Octal

For example:

EXPRESSION	RADIX	DECIMAL VALUE
000 00011B	Binary	3
160Q	Octal (also 112O)	112
3200	Decimal (also 3200D)	3200
77000A	Offset octal	16128
021AH	Hexadecimal	282



<u>LEGAL INTEGER EXPRESSIONS</u>	<u>ILLEGAL INTEGER EXPRESSIONS</u>	<u>COMMENTS</u>
----------------------------------	------------------------------------	-----------------

232	232.1	Decimals may not be specified
10111B	226B	Not a binary number
177Q	888	Not an octal number
A1FH	21C	No hex radix specified

If an integer expression evaluates to less than -32,767, or greater than 65,534, an error code is flagged.

SYMBOLS

An expression may contain any user defined symbol. Although most symbols do not need to be defined sequentially before the referencing statement, some pseudo operators require all their operand symbols to be defined in earlier statements in the program. Such operators are said to require "pass one evaluation" and are documented in "The 8080 Opcodes" (Page 4-10). All symbols must consist of legal HASL-8 characters.

The # Symbol

If the pound sign (#) is the first character in an expression, the expression is evaluated as a 16-bit expression. After the expression is evaluated, the resultant value is masked to an 8-bit equivalent. Once this is done, a 16-bit operand may be referenced in an instruction requiring 8 bits without causing an overflow (V) error. For example:

```
MVI H, ADDR/256
MVI L, #ADDR      (HL) = 16 bit address
```

In this example, the first line of code loads the H and L register pair (16-bit register) with the binary value associated with the label "ADDR" divided by 256. The second line of code immediately loads the L register (an 8-bit register) with the lower 8-bits of the binary value equated to the symbol ADDR in the symbol table. This process does not cause an overflow error, as the 16-bit binary equivalent of ADDR is masked to the least significant 8-bits before it is moved into the 8-bit L register.



CHARACTER STRING

A character string consisting of one or two legal characters may be used as a token in an HASL-8 expression. Such a character string is enclosed in a single quote (apostrophe). For example:

'A'	The character A (Value 101Q)
'GL'	The character string GL (Value 10n 114A)
" "	The character quotation mark (Value 042Q)

THE ORIGIN SYMBOL, ORG

The current value of the origin counter may be referenced with the special symbol asterisk (*). NOTE: The assembler decides from the expression context whether the asterisk (*) represents the origin counter or is the multiplication operator. For example, the program

```
ORG    10
A      EQU    ***
```

defines the symbol A to have the value 100. The first statement, ORG 10, sets the origin counter to the value 10. In the second statement, the label A is equated with the first asterisk, which the assembler presumes to be the symbol for the origin counter. This is multiplied by the third symbol, which the assembler also presumes to be the origin symbol. However, the middle asterisk is taken as the multiplication operator.

THE 8080 OPCODES*

Heath Assembly Language supports the standard 8080 machine opcodes. A review of the 8080 instruction set is presented on the following pages. Included in this review is a discussion of instruction and data formats, addressing modes, conditions flags, the symbols or abbreviations used in describing the 8080 instruction set, and the discussion of the format used to describe each instruction.

* Portions of this section are reprinted with the permission of Intel Corporation (Copyright, 1976).



The 8080 instruction set includes five different types of instructions:

- **Data Transfer Group** — move data between registers or between memory and registers.
- **Arithmetic Group** — add, subtract, increment, or decrement data in registers or in memory.
- **Logical Group** — AND, OR, EXCLUSIVE-OR, compare, rotate, or complement data in registers or in memory.
- **Branch Group** — conditional and unconditional jump instructions, subroutine call instructions, and return instructions.
- **Stack, I/O and Machine Control Group** — includes I/O instructions, as well as instructions for maintaining the stack and internal control flags.

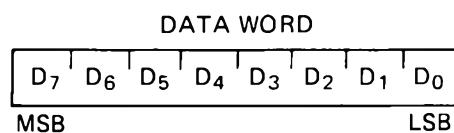
TERMS, SYMBOLS, & NOMENCLATURE

INSTRUCTION AND DATA FORMATS

Memory for the 8080 is organized into 8-bit quantities called bytes. Each byte has a unique 16-bit binary address corresponding to its sequential position in memory.

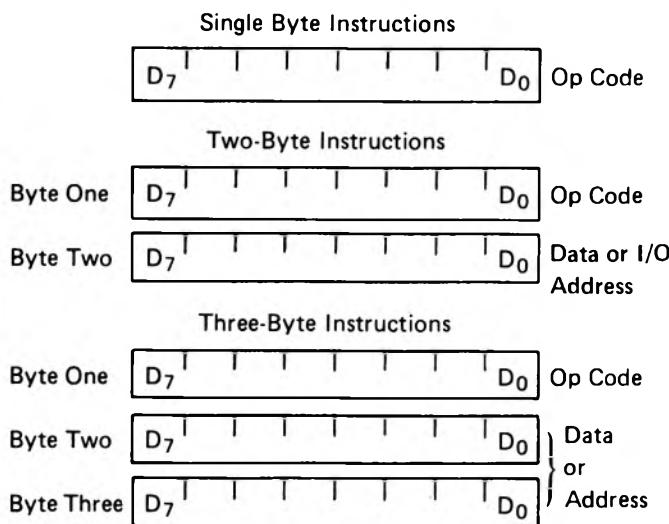
The 8080 can directly address up to 65,536 bytes of memory, which may consist of both read-only memory (ROM) elements and random-access memory (RAM) elements (read/write memory).

Data in the 8080 is stored in the form of 8-bit binary integers:



When a register or data word contains a binary number, it is necessary to establish the order in which the bits of the number are written. In the Intel 8080, BIT 0 is referred to as the **Least Significant Bit (LSB)**, and BIT 7 (of an 8-bit number) is referred to as the **Most Significant Bit (MSB)**.

The 8080 program instructions may be one, two, or three bytes in length. Multiple byte instructions must be stored in successive memory locations; the address of the first byte is always used as the address of the instructions. The exact instruction format will depend on the particular operation to be executed.



ADDRESSING MODES

Often, the data that is to be operated on is stored in memory. When multi-byte numeric data is used, the data, like instructions, is stored in successive memory locations with the least significant byte first, followed by increasingly significant bytes. The 8080 has four different modes for addressing data stored in memory or in registers:

- **Direct** — Bytes 2 and 3 of the instruction contain the exact memory address of the data item (the low-order bits of the address are in byte 2, the high-order bits in byte 3).
- **Register** — Specifies the register or register pair in which the data is located.
- **Register Indirect** — Specifies a register pair which contains the memory address where the data is located (the high-order bits of the address are in the first register of the pair, the low-order bits in the second).
- **Immediate** — Contains the data itself. This is either an 8-bit quantity or a 16-bit quantity (least significant byte first, most significant byte second).



Unless directed by an interrupt or branch instruction, the execution of instructions proceeds through consecutively increasing memory locations. A branch instruction can specify the address of the next instruction to be executed in one of two ways:

- **Direct** — The branch instruction contains the address of the next instruction to be executed. (Except for the “RST” instruction, byte 2 contains the low-order address and byte 3 the high-order address.)
- **Register Indirect** — The branch instruction indicates a register pair which contains the address of the next instruction to be executed. (The high-order bits of the address are in the first register of the pair, the low-order bits in the second.)

The RST instruction is a special 1-byte call instruction (usually used during interrupt sequences). RST includes a 3-bit field; program control is transferred to the instruction whose address is eight times the contents of this 3-bit field.

CONDITION FLAGS

There are five condition flags associated with the execution of instructions on the 8080. They are Zero, Sign, Parity, Carry, and Auxiliary Carry, and are each represented by a 1-bit register in the CPU. A flag is “set” by forcing the bit to 1; and “reset” by forcing the bit to 0.

Unless indicated otherwise, when an instruction affects a flag, it affects it in the following manner.

Zero: If the result of an instruction has the value 0, this flag is set. Otherwise it is reset.

Sign: If the most significant bit of the result of the operation has the value 1, this flag is set. Otherwise it is reset.

Parity If the modulo 2 sum of the bits of the result of the operation is 0 (i. e., if the result has even parity), this flag is set. Otherwise it is reset (i. e., if the result has odd parity).

Carry: If the instruction resulted in a carry (from addition), or a borrow (from subtraction or a comparison) out of the high-order bit, this flag is set. Otherwise it is reset.



Auxiliary Carry: If the instruction caused a carry out of bit 3 and into bit 4 of the resulting value, the auxiliary carry is set. Otherwise it is reset. This flag is affected by single precision additions, subtractions, increments, decrements, comparisons, and logical operations, but is principally used with additions and increments preceding a DAA (Decimal Adjust Accumulator) instruction.

Symbols and Abbreviations

The following symbols and abbreviations are used in the subsequent description of the 8080 instructions:

SYMBOLS	MEANING
accumulator	Register A
addr	16-bit address quantity
data	8-bit data quantity
data 16	16-bit data quantity
byte 2	The second byte of the instruction
byte 3	The third byte of the instruction
port	8-bit address of an I/O device
r, r1, r2	One of the registers A,B,C,D,E,H,L
DDD, SSS	The bit pattern designating one of the registers A, B, C, D, E, H, L (DDD = destination, SSS = source):

DDD or SSS	REGISTER NAME
111	A
000	B
001	C
010	D
011	E
100	H
101	L

111	A
000	B
001	C
010	D
011	E
100	H
101	L



rp One of the register pairs:

B represents the B, C pair with B as the high-order register and C as the low-order register;

D represents the D, E pair with D as the high-order register and E as the low-order register;

H represents the H, L pair with H as the high-order register and L as the low-order register;

SP represents the 16-bit stack pointer register.

RP The bit pattern designating one of the register pairs B, D, H, SP:

RP	REGISTER PAIR
00	B-C
01	D-E
10	H-L
11	SP

00	B-C
01	D-E
10	H-L
11	SP

rh The first (high-order) register of a designated register pair.

rl The second (low-order) register of a designated register pair.

PC 16-bit program counter register (PCH and PCL are used to refer to the high-order and low-order 8-bits respectively).

SP 16-bit stack pointer register (SPH and SPL are used to refer to the high-order and low-order 8-bits respectively).

rm Bit m of the register r (bits are numbered 7 through 0 from left to right).

Z, S, P, The condition flags:

Cy, AC

Zero,
Sign,
Parity,
Carry,
and Auxiliary Carry,
respectively.



NOTE: HASL-8 recognizes the E as well as the Z defining the zero bit. Therefore, JZ (jump zero) or JE (jump equal) are both valid op-codes.

() The contents of the memory location or registers enclosed in the parentheses.

\leftarrow “Is transferred to”

\wedge Logical AND

\vee Exclusive OR

\vee Inclusive OR

$+$ Addition

$-$ Two's complement subtraction

$*$ Multiplication

\leftrightarrow “Is exchanged with”

$-$ The one's complement (e. g., (A))

n The restart number 0 through 7

NNN The binary representation 000 through 111 for restart number 0 through 7 respectively.

Description Format

The following pages provide a detailed description of the instruction set of the 8080. Each instruction is described in the following manner:

1. The HASL-8 format, consisting of the opcode and operand fields, is printed in **BOLDFACE** on the left side of the first line.
2. The name of the instruction is enclosed in parentheses at the center of the first line.
3. The next line(s) contain a symbolic description of the operation of the instruction.

4. This is followed by a narrative description of the operation of the instruction.
5. The following line(s) contain the binary fields and patterns that comprise the machine instruction.
6. The last two lines contain incidental information about the execution of the instruction. The number of machine cycles and states required to execute the instruction are listed first. If the instruction has two possible execution times, as in a conditional jump, both times will be listed, separated by a slash. Next, any significant data addressing modes (see "Addressing Modes," Page 4-12) are listed. The last line lists any of the five Flags that are affected by the execution of the instruction.

Data Transfer Group

This group of instructions transfers data to and from registers and memory.
Condition flags are not affected by any instruction in this group.

MOV r1, r2 (Move Register)

$(r1) \leftarrow (r2)$

The content of register r2 is moved to register r1.

0		1		D		D		D		S		S		S
---	--	---	--	---	--	---	--	---	--	---	--	---	--	---

Cycles: 1

Addressing: register

States: 5

Flags: none

MOV r, M (Move from memory)

$(r) \leftarrow ((H) (L))$

The content of the memory location whose address is in registers H and L is moved to register r.

0		1		D		D		D		1		1		0
---	--	---	--	---	--	---	--	---	--	---	--	---	--	---

Cycles: 2

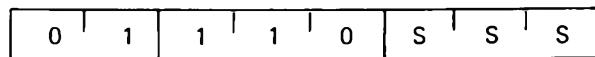
Addressing: reg. indirect

States: 7

Flags: none

**MOV M, r** (Move to memory) $((H) (L)) \leftarrow (r)$

The content of register r is moved to the memory location whose address is in registers H and L.



Cycles: 2

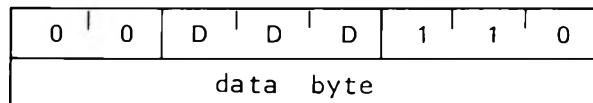
States: 7

Addressing: reg. indirect

Flags: none

MVI r, data (Move to register immediate) $(r) \leftarrow (\text{byte } 2)$

The content of byte 2 of the instruction is moved to register r.



Cycles: 2

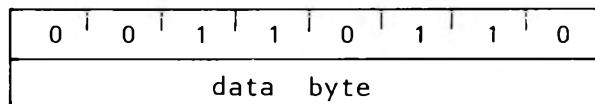
States: 7

Addressing: immediate

Flags: none

MVI M, data (Move to memory immediate) $((H) (L)) \leftarrow (\text{byte } 2)$

The content of byte 2 of the instruction is moved to the memory location whose address is in registers H and L.



Cycles: 3

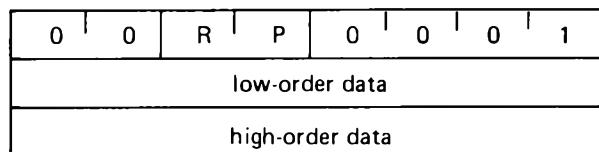
States: 10

Addressing: immed./reg.
indirect

Flags: none

**LXI rp, data 16** (Load register pair immediate) $(rh) \leftarrow (\text{byte 3}),$ $(rl) \leftarrow (\text{byte 2})$

Byte 3 of the instruction is moved into the high-order register (rh) of the register pair rp. Byte 2 of the instruction is moved into the low-order register (rl) of the register pair rp.



Cycles: 3

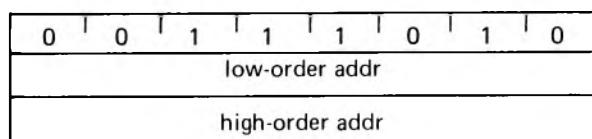
Addressing: immediate

States: 10

Flags: none

LDA addr (Load Accumulator direct) $(A) \leftarrow (\text{byte 3}) (\text{byte 2})$

The content of the memory location, whose address is specified in byte 2 and byte 3 of the instruction, is moved to register A.



Cycles: 4

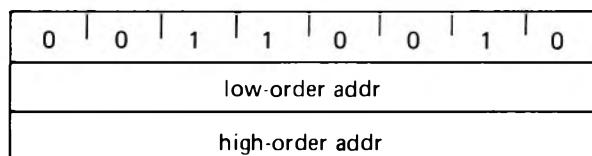
Addressing: direct

States: 13

Flags: none

STA addr (Store accumulator direct) $((\text{byte 3}) (\text{byte 2})) \leftarrow (A)$

The content of the accumulator is moved to the memory location whose address is specified in byte 2 and byte 3 of the instruction.



Cycles: 4

Addressing: direct

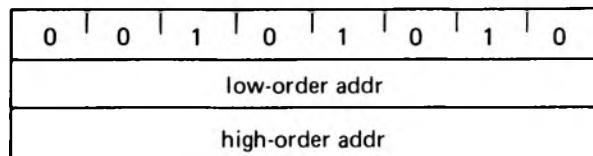
States: 13

Flags: none

**LHLD addr** (Load H and L direct)
$$(L) \leftarrow ((\text{byte 3}) (\text{byte 2}))$$

$$(H) \leftarrow ((\text{byte 3}) (\text{byte 2}) + 1)$$

The content of the memory location whose address is specified in byte 2 and byte 3 of the instruction is moved to register L. The content of the memory location at the succeeding address is moved to register H.



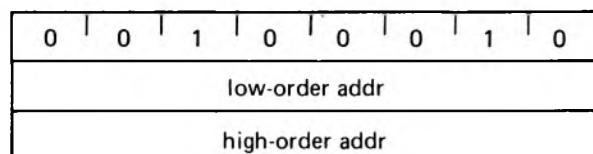
Cycles: 5
States: 16

Addressing: direct
Flags: none

SHLD addr (Store H and L direct)
$$((\text{byte 3}) (\text{byte 2})) \leftarrow (L)$$

$$((\text{byte 3}) (\text{byte 2}) + 1) \leftarrow (H)$$

The content of register L is moved to the memory location whose address is specified in byte 2 and byte 3. The content of register H is moved to the succeeding memory location.



Cycles: 5
States: 16

Addressing: direct
Flags: none

LDAX rp (Load accumulator indirect)
$$(A) \leftarrow ((rp))$$

The content of the memory location whose address is in the register pair rp is moved to register A. NOTE: Only register pairs rp = B (registers B and C) or rp = D (registers D and E) may be specified.



Cycles: 2
States: 7

Addressing: reg. indirect
Flags: none

**STAX rp** (Store accumulator indirect) $((rp)) \leftarrow (A)$

The content of register A is moved to the memory location whose address is in the register pair rp. NOTE: Only register pairs rp = B (registers B and C) or rp = D (registers D and E) may be specified.

0 0	R P	0 0 1 0
-------	-------	---------------

Cycles: 2

Addressing: reg. indirect

States: 7

Flags: none

XCHG (Exchange H and L with D and E) $(H) \longleftrightarrow (D)$ $(L) \longleftrightarrow (E)$

The contents of registers H and L are exchanged with the contents of registers D and E.

1 1 1 0 1 0 1 1

Cycles: 1

Addressing: register

States: 4

Flags: none

Arithmetic Group

This group of instructions performs arithmetic operations on data in registers and memory.

Unless indicated otherwise, all instructions in this group affect the Zero, Sign, Parity, Carry, and Auxiliary Carry flags according to the standard rules.

All subtraction operations are performed via two's complement arithmetic and set the carry flag to one to indicate a borrow and clear it to indicate no borrow.

ADD r (Add Register) $(A) \leftarrow (A) + (r)$

The content of register r is added to the content of the accumulator. The result is placed in the accumulator.

1 0 0 0 0 s s s

Cycles: 1

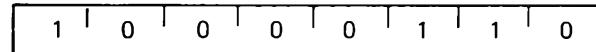
Addressing: register

States: 4

Flags: Z,S,P,CY,AC

**ADD M** (Add memory)
$$(A) \leftarrow (A) + ((H)(L))$$

The content of the memory location whose address is contained in the H and L registers is added to the content of the accumulator. The result is placed in the accumulator.



Cycles: 2

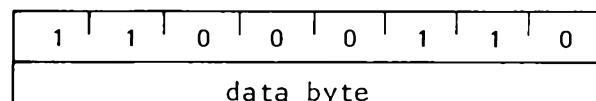
States: 7

Addressing: reg. indirect

Flags: Z,S,P,CY,AC

ADI DATA (add immediate)
$$(A) \leftarrow (A) + (\text{byte } 2)$$

The content of the second byte of the instruction is added to the content of the accumulator. The result is placed in the accumulator.



Cycles: 2

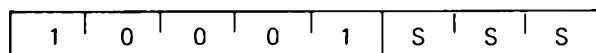
States: 7

Addressing: immediate

Flags: Z,S,P,CY,AC

ADC r (Add Register with carry)
$$(A) \leftarrow (A) + (r) + (\text{CY})$$

The content of register r and the content of the carry bit are added to the content of the accumulator. The result is placed in the accumulator.



Cycles: 1

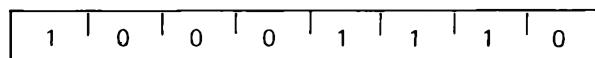
States: 4

Addressing: register

Flags: Z,S,P,CY,AC

ADC M (Add memory with carry)
$$(A) \leftarrow (A) + ((H) (L)) + (CY)$$

The content of the memory location whose address is contained in the H and L registers and the content of the CY flag are added to the accumulator. The result is placed in the accumulator.



Cycles: 2

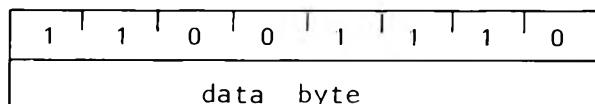
Addressing: reg. indirect

States: 7

Flags: Z,S,P,CY,AC

ACI data (Add immediate with carry)
$$(A) \leftarrow (A) + (\text{byte } 2) + (CY)$$

The content of the second byte of the instruction and the content of the CY flag are added to the contents of the accumulator. The result is placed in the accumulator.



Cycles: 2

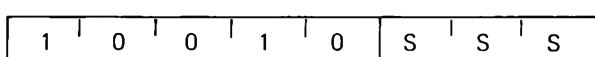
Addressing: immediate

States: 7

Flags: Z,S,P,CY,AC

SUB r (Subtract Register)
$$(A) \leftarrow (A) - (r)$$

The content of register r is subtracted from the content of the accumulator. The result is placed in the accumulator.



Cycles: 1

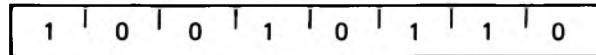
Addressing: register

States: 4

Flags: Z,S,P,CY,AC

**SUB M** (Subtract memory)
$$(A) \leftarrow (A) - ((H) (L))$$

The content of the memory location whose address is contained in the H and L registers is subtracted from the content of the accumulator. The result is placed in the accumulator.



Cycles: 2

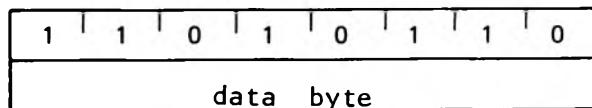
States: 7

Addressing: reg. indirect

Flags: Z,S,P,CY,AC

SUI DATA (Subtract immediate)
$$(A) \leftarrow (A) - (\text{byte } 2)$$

The content of the second byte of the instruction is subtracted from the content of the accumulator. The result is placed in the accumulator.



Cycles: 2

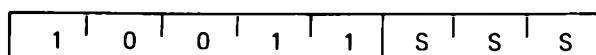
States: 7

Addressing: immediate

Flags: Z,S,P,CY,AC

SBB r (Subtract Register with borrow)
$$(A) \leftarrow (A) - (r) - (\text{CY})$$

The content of register r and the content of the CY flag are both subtracted from the accumulator. The result is placed in the accumulator.



Cycles: 1

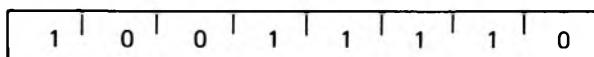
States: 4

Addressing: register

Flags: Z,S,P,CY,AC

**SBB M** (Subtract memory with borrow) $(A) \leftarrow (A) - ((H) (L)) - (CY)$

The content of the memory location whose address is contained in the H and I registers and the content of the CY flag are both subtracted from the accumulator. The result is placed in the accumulator.

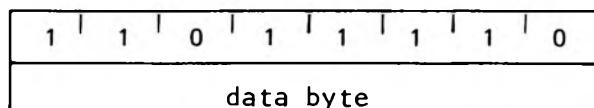


Cycles: 2
States: 7

Addressing: reg. indirect
Flags: Z,S,P,CY,AC

SBI data (Subtract immediate with borrow) $(A) \leftarrow (A) - (\text{byte } 2) - (CY)$

The contents of the second byte of the instruction and the contents of the CY flag are both subtracted from the accumulator. The result is placed in the accumulator.

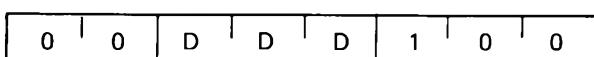


Cycles: 2
States: 7

Addressing: immediate
Flags: Z,S,P,CY,AC

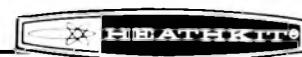
INR r (Increment Register) $(r) \leftarrow (r) + 1$

The content of register r is incremented by one. NOTE: All condition flags **except CY** are affected.

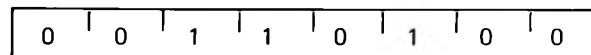


Cycles: 1
States: 5

Addressing: register
Flags: Z,S,P,AC

**INR M** (Increment memory)
 $((H) (L)) \leftarrow ((H) (L)) + 1$

The content of the memory location whose address is contained in the H and L registers is incremented by one. NOTE: All condition flags **except CY** are affected.

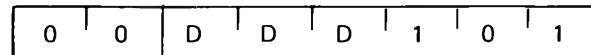


Cycles: 3
States: 10

Addressing: reg. indirect
Flags: Z,S,P,AC

DCR r (Decrement Register)
 $(r) \leftarrow (r) - 1$

The content of register r is decremented by one. NOTE: All condition flags **except CY** are affected.

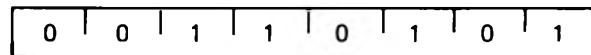


Cycles: 1
States: 5

Addressing: register
Flags: Z,S,P,AC

DCR M (Decrement memory)
 $((H) (L)) \leftarrow ((H) (L)) - 1$

The content of the memory location whose address is contained in the H and L registers is decremented by one. NOTE: All condition flags **except CY** are affected.



Cycles: 3
States: 10

Addressing: reg. indirect
Flags:



INX rp (Increment register pair) 0(0,2,4,6)3

$$(rh) (rl) \leftarrow (rh) (rl) + 1$$

The content of the register pair rp is incremented by one. NOTE: **No condition flags are affected.**

0		0		R		P		0		0		1		1
---	--	---	--	---	--	---	--	---	--	---	--	---	--	---

Cycles: 1

Addressing: register

States: 5

Flags: none

DCX rp (Decrement register pair)

$$(rh) (rl) \leftarrow (rh) (rl) - 1$$

The content of register pair rp is decremented by one. NOTE: **No condition flags are affected.**

0		0		R		P		1		0		1		1
---	--	---	--	---	--	---	--	---	--	---	--	---	--	---

Cycles: 1

Addressing: register

States: 5

Flags: none

DAD rp (Add register pair to H and L)

$$(H) (L) \leftarrow (H) (L) + (rh) (rl)$$

The content of register pair rp is added to the content of the register pair H and L. The result is placed in register pair H and L. NOTE: **Only the CY flag is affected.** It is set if there is a carry out of the double precision add; otherwise it is reset.

0		0		R		P		1		0		0		1
---	--	---	--	---	--	---	--	---	--	---	--	---	--	---

Cycles: 3

Addressing: register

States: 10

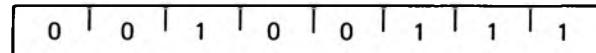
Flags: CY

**DAA**

(Decimal Adjust Accumulator)

The eight-bit number in the accumulator is adjusted to form two 4-bit Binary-Coded-Decimal digits by the following process:

1. If the value of the least significant 4 bits of the accumulator is greater than 9 **or** if the AC flag is set, 6 is added to the accumulator.
2. If the value of the most significant 4 bits of the accumulator is now greater than 9, **or** if the CY flag is set, 6 is added to the most significant 4 bits of the accumulator.



Cycles:	1
States:	4
Flags:	Z,S,P,CY,AC

Logical Group:

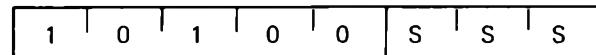
This group of instructions performs logical (Boolean) operations on data in registers and memory and on condition flags.

Unless indicated otherwise, all instructions in this group affect the Zero, Sign, Parity, Auxiliary Carry, and Carry flags according to the standard rules.

ANA r (AND Register)

$$(A) \leftarrow (A) \wedge (r)$$

The content of register r is logically anded with the content of the accumulator. The result is placed in the accumulator. **The CY flag is cleared.**

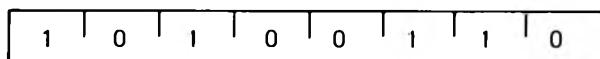


Cycles:	1
States:	4

Addressing:	register
Flags:	Z,S,P,CY,AC

**ANA M** (AND memory) $(A) \leftarrow (A) \wedge ((H)(L))$

The contents of the memory location whose address is contained in the H and L registers is logically anded with the content of the accumulator. The result is placed in the accumulator. **The CY flag is cleared.**



Cycles: 2

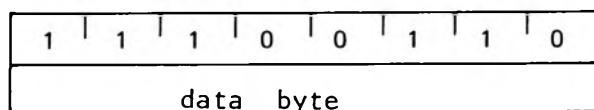
Addressing: reg. indirect

States: 7

Flags: Z,S,P,CY,AC

ANI data (AND immediate) $(A) \leftarrow (A) \wedge (\text{byte } 2)$

The content of the second byte of the instruction is logically anded with the contents of the accumulator. The result is placed in the accumulator. **The CY and AC flags are cleared.**



Cycles: 2

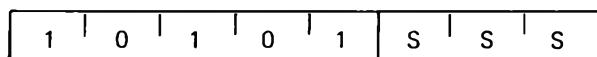
Addressing: immediate

States: 7

Flags: Z,S,P,CY,AC

XRA r (Exclusive OR Register) $(A) \leftarrow (A) \vee (r)$

The content of register r is exclusive-OR'd with the content of the accumulator. The result is placed in the accumulator. **The CY and AC flags are cleared.**



Cycles: 1

Addressing: register

States: 4

Flags: Z,S,P,CY,AC

**XRA M**

(Exclusive OR Memory)

$$(A) \leftarrow (A) \nabla ((H) (L))$$

The content of the memory location whose address is contained in the H and L registers is exclusive-OR'd with the content of the accumulator. The result is placed in the accumulator. **The CY and AC flags are cleared.**

1	0	1	0	1	1	1	0
---	---	---	---	---	---	---	---

Cycles: 2

States: 7

Addressing: reg. indirect

Flags: Z,S,P,CY,AC

XRI data

(Exclusive OR immediate)

$$(A) \leftarrow (A) \nabla (\text{byte } 2)$$

The content of the second byte of the instruction is exclusive-OR'd with the content of the accumulator. The result is placed in the accumulator. **The CY and AC flags are cleared.**

1	1	1	0	1	1	1	0
data byte							

Cycles: 2

States: 7

Addressing: immediate

Flags: Z,S,P,CY,AC

ORA r

(OR Register)

$$(A) \leftarrow (A) \vee (r)$$

The content of register r is inclusive-OR'd with the content of the accumulator. The result is placed in the accumulator. **The CY and AC flags are cleared.**

1	0	1	1	0	s	s	s
---	---	---	---	---	---	---	---

Cycles: 1

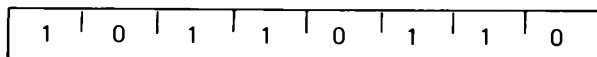
States: 4

Addressing: register

Flags: Z,S,P,CY,AC

**ORA M** (OR memory) $(A) \leftarrow (A) \vee ((H) (L))$

The content of the memory location whose address is contained in the H and L registers is inclusive-OR'd with the content of the accumulator. The result is placed in the accumulator. **The CY and AC flags are cleared.**



Cycles: 2

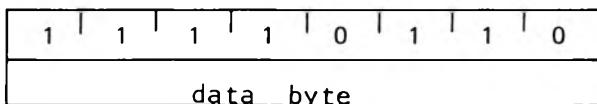
Addressing: reg. indirect

States: 7

Flags: Z,S,P,CY,AC

ORI data (OR Immediate) $(A) \leftarrow (A) \vee (\text{byte } 2)$

The content of the second byte of the instruction is inclusive-OR'd with the content of the accumulator. The result is placed in the accumulator. **The CY and AC flags are cleared.**



Cycles: 2

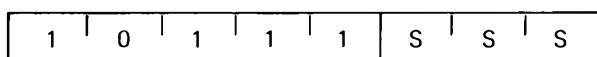
Addressing: immediate

States: 7

Flags: Z,S,P,CY,AC

CMP r (Compare Register) $(A) — (r)$

The content of register r is subtracted from the accumulator. The accumulator remains unchanged. The condition flags are set as a result of the subtraction. **The Z flag is set to 1 if $(A) = (r)$. The CY flag is set to 1 if $(A) < (r)$.**



Cycles: 1

Addressing: register

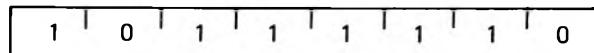
States: 4

Flags: Z,S,P,CY,AC


CMP M (Compare memory)

(A) — ((H) (L))

The content of the memory location whose address is contained in the H and L registers is subtracted from the accumulator. The accumulator remains unchanged. The condition flags are set as a result of the subtraction. The Z flag is set to 1 if (A) = ((H) (L)). The CY flag is set to 1 if (A) < ((H) (L)).



Cycles: 2

States: 7

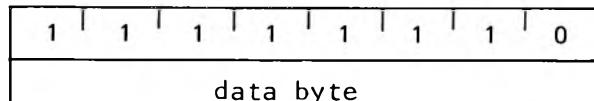
Addressing: reg. indirect

Flags: Z,S,P,CY,AC

CPI data (Compare immediate)

(A) — (byte 2)

The content of the second byte of the instruction is subtracted from the accumulator. The condition flags are set by the result of the subtraction. The Z flag is set to 1 if (A) = (byte 2). The CY flag is set to 1 if (A) < (byte 2).



Cycles: 2

States: 7

Addressing: immediate

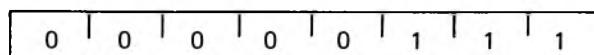
Flags: Z,S,P,CY,AC

RLC (Rotate left)

$(A_{n+1}) \leftarrow (A_n); (A_0) \leftarrow (A_7)$

$(CY) \leftarrow (A_7)$

The content of the accumulator is rotated left one position. The low order bit and the CY flag are both set to the value shifted out of the high order bit position. **Only the CY flag is affected.**



Cycles: 1

States: 4

Flags: CY

**RRC** (Rotate right) $(A_n) \leftarrow (A_{n-1}); (A_7) \leftarrow (A_0)$ $(CY) \leftarrow (A_0)$

The content of the accumulator is rotated right one position. The high order bit and the CY flag are both set to the value shifted out of the low order bit position. **Only the CY flag is affected.**

0		0		0		0		1		1		1		1
---	--	---	--	---	--	---	--	---	--	---	--	---	--	---

Cycles: 1

States: 4

Flags: CY

RAL (Rotate left through carry) $(A_{n+1}) \leftarrow (A_n); (CY) \leftarrow (A_7)$ $(A_0) \leftarrow (CY)$

The content of the accumulator is rotated left one position through the CY flag. The low order bit is set equal to the CY flag and the CY flag is set to the value shifted out of the high order bit. **Only the CY flag is affected.**

0		0		0		1		0		1		1		1
---	--	---	--	---	--	---	--	---	--	---	--	---	--	---

Cycles: 1

States: 4

Flags: CY

RAR (Rotate right through carry) $(A_n) \leftarrow (A_{n+1}); (CY) \leftarrow (A_0)$ $(A_7) \leftarrow (CY)$

The content of the accumulator is rotated right one position through the CY flag. The high order bit is set to the CY flag and the CY flag is set to the value shifted out of the low order bit. **Only the CY flag is affected.**

0		0		0		1		1		1		1		1
---	--	---	--	---	--	---	--	---	--	---	--	---	--	---

Cycles: 1

States: 4

Flags: CY


CMA (Complement accumulator)

 $(A) \leftarrow (A)$

The contents of the accumulator are complemented (zero bits become 1, one bits become 0). **No flags are affected.**

0		0		1		0		1		1		1		1
---	--	---	--	---	--	---	--	---	--	---	--	---	--	---

Cycles: 1

States: 4

Flags: none

CMC (Complement carry)

 $(CY) \leftarrow (\overline{CY})$

The CY flag is complemented. **No other flags are affected.**

0		0		1		1		1		1		1		1
---	--	---	--	---	--	---	--	---	--	---	--	---	--	---

Cycles: 1

States: 4

Flags: CY

STC (Set carry)

 $(CY) \leftarrow 1$

The CY flag is set to 1. **No other flags are affected.**

0		0		1		1		0		1		1		1
---	--	---	--	---	--	---	--	---	--	---	--	---	--	---

Cycles: 1

States: 4

Flags: CY

Branch Group

This group of instructions alter normal sequential program flow. **Condition flags are not affected** by any instruction in this group.

The two types of branch instructions are unconditional and conditional. Unconditional transfers simply perform the specified operation on register PC (the



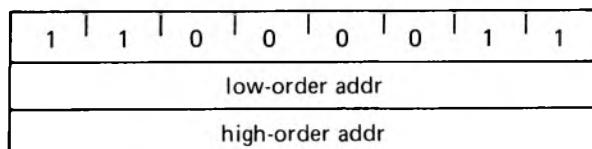
program counter). Conditional transfers examine the status of one of the four processor flags to determine if the specified branch is to be executed. The following conditions may be specified:

CONDITION	CCC	OCTAL
NE or NZ — not zero ($Z=0$)	000	0
E or Z — zero ($Z=1$)	001	1
NC — no carry ($CY = 0$)	010	2
C — carry ($CY = 1$)	011	3
PO — parity odd ($P = 0$)	100	4
PE — parity even ($P = 1$)	101	5
P — plus ($S = 0$)	110	6
M — minus ($S = 1$)	111	7

JMP addr (Jump)

$(PC) \leftarrow (\text{byte } 3) (\text{byte } 2)$

Control is transferred to the instruction whose address is specified in byte 3 and byte 2 of the current instruction.



Cycles: 3

Addressing: immediate

States: 10

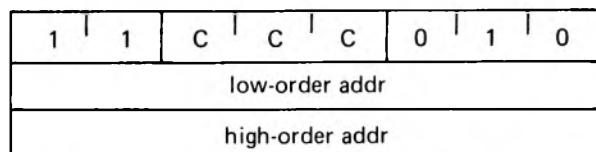
Flags: none

JNE JNC JPO JP (Condition jump) JE JC JPE JM

If (CCC),

$(PC) \leftarrow (\text{byte } 3) (\text{byte } 2)$

If the specified condition is true, control is transferred to the instruction whose address is specified in byte 3 and byte 2 of the current instruction. Otherwise, control continues sequentially.



Cycles: 3

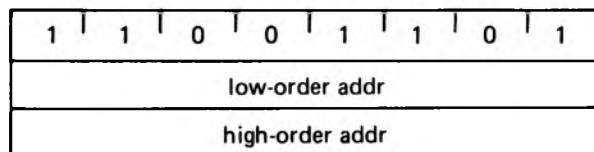
Addressing: immediate

States: 10

Flags: none

**CALL addr** (Call)
$$\begin{aligned} ((SP) - 1) &\leftarrow (PCH) \\ ((SP) - 2) &\leftarrow (PCL) \\ (SP) &\leftarrow (SP) - 2 \\ (PC) &\leftarrow (\text{byte 3}) \text{ (byte 2)} \end{aligned}$$

The high-order eight bits of the next instruction address are moved to the memory location whose address is one less than the content of register SP. The low-order eight bits of the next instruction address are moved to the memory location whose address is two less than the content of register SP. The content of register SP is decremented by 2. Control is transferred to the instruction whose address is specified in byte 3 and byte 2 of the current instruction.



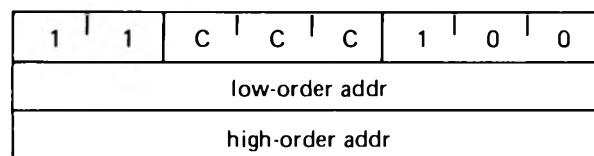
Cycles: 5 Addressing: immediate/reg.
 indirect
States: 17 Flags: none

CNE **CNC** **CPO** **CP** (Condition call)
CE **CC** **CPE** **CM**

If (CCC),

$$\begin{aligned} ((SP) - 1) &\leftarrow (PCH) \\ ((SP) - 2) &\leftarrow (PCL) \\ (SP) &\leftarrow (SP) - 2 \\ (PC) &\leftarrow (\text{byte 3}) \text{ (byte 2)} \end{aligned}$$

If the specified condition is true, the actions specified in the CALL instruction (see above) are performed; otherwise, control continues sequentially.



Cycles: 3/5 Addressing: immediate/reg.
 indirect
States: 11/17 Flags: none

**RET** (Return)

$(PCL) \leftarrow ((SP))$:
 $(PCH) \leftarrow ((SP)) + 1$;
 $(SP) \leftarrow (SP) + 2$;

The content of the memory location whose address is specified in register SP is moved to the low-order eight bits of register PC. The content of the memory location whose address is one more than the content of register SP is moved to the high-order eight bits of register PC. The content of register SP is incremented by 2.

1		1		0		0		1		0		0		1
---	--	---	--	---	--	---	--	---	--	---	--	---	--	---

Cycles: 3 Addressing: reg. indirect
States: 10 Flags: none

RNE RNC COP CP
RE RC CPE CM (Conditional return)

If (CCC).

$(PCL) \leftarrow ((SP))$
 $(PCH \leftarrow ((SP) + 1))$
 $(SP) \leftarrow (SP) + 2$

If the specified condition is true, the actions specified in the RET instruction (see above) are performed; otherwise, control continues sequentially.

1		1		c		c		c		0		0		0
---	--	---	--	---	--	---	--	---	--	---	--	---	--	---

Cycles: 1/3 Addressing: reg. indirect
States: 5/11 Flags: none

RST n (Restart)

$((SP) - 1) \leftarrow (PCH)$
 $((SP) - 2) \leftarrow (PCL)$
 $(SP) \leftarrow (SP) - 2$
 $(PC) \leftarrow 8 * (NNN)$

The high-order eight bits of the next instruction address are moved to the memory location whose address is one less than the content of register SP. The low-order eight bits of the next instruction address are moved to the memory location whose address is two less than the content of register SP.



The content of register SP is decremented by two. Control is transferred to the instruction whose address is eight times the content of NNN.

1		1		N		N		N		1		1		1
---	--	---	--	---	--	---	--	---	--	---	--	---	--	---

Cycles: 3
States: 11

Addressing: reg. indirect
Flags: none

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	N	N	N	0	0

Program Counter After Restart

PCHL (Jump H and L indirect — move H and L to PC)
 $(PCH) \leftarrow (H)$
 $(PCL) \leftarrow (L)$

The content of register H is moved to the high-order eight bits of register PC. The content of register L is moved to the low-order eight bits of register PC.

1		1		1		1		0		1		0		0		1
---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---

Cycles: 1
States: 5

Addressing: register
Flags: none

Stack, I/O, and Machine Control Group

This group of instructions performs I/O, manipulates the Stack, and alters internal control flags. Unless otherwise specified, **condition flags are not affected by any instructions in this group.**

PUSH rp (Push)

$((SP) - 1) \leftarrow (rh)$
 $((SP) - 2) \leftarrow (rl)$
 $(SP) \leftarrow (SP) - 2$



The content of the high-order register of register pair rp is moved to the memory location whose address is one less than the content of register SP. The content of the low-order register of register pair rp is moved to the memory location whose address is two less than the content of register SP. The content of register SP is decremented by 2. **NOTE: Register pair rp = SP may not be specified.**

1		1		R		P		0		1		0		1
---	--	---	--	---	--	---	--	---	--	---	--	---	--	---

Cycles: 3

Addressing: reg. indirect

States: 11

Flags: none

PUSH PSW (Push processor status word)

$((SP) - 1) \leftarrow (A)$
 $((SP) - 2)_0 \leftarrow (CY), ((SP) - 2)_1 \leftarrow 1$
 $((SP) - 2)_2 \leftarrow (P), ((SP) - 2)_3 \leftarrow 0$
 $((SP) - 2)_4 \leftarrow (AC), ((SP) - 2)_5 \leftarrow 0$
 $((SP) - 2)_6 \leftarrow (Z), ((SP) - 2)_7 \leftarrow (S)$
 $(SP) \leftarrow (SP) - 2$

The content of register A is moved to the memory location whose address is one less than register SP. The contents of the condition flags are assembled into a processor status word and the word is moved to the memory location whose address is two less than the content of register SP. The content of register SP is decremented by two.

1		1		1		1		0		1		0		1
---	--	---	--	---	--	---	--	---	--	---	--	---	--	---

Cycles: 3

Addressing: reg. indirect

States: 11

Flags: none

FLAG WORD

D₇ D₆ D₅ D₄ D₃ D₂ D₁ D₀

S	Z	0	AC	0	P	1	CY
---	---	---	----	---	---	---	----

**POP rp** (Pop)
$$\begin{aligned}(\rl) &\leftarrow ((\text{SP})) \\ (\rh) &\leftarrow ((\text{SP}) + 1) \\ (\text{SP}) &\leftarrow (\text{SP}) + 2\end{aligned}$$

The content of the memory location whose address is specified by the content of register SP is moved to the low-order register of register pair rp. The content of the memory location whose address is one more than the content of register SP is moved to the high-order register of register pair rp. The content of register SP is incremented by 2. **NOTE: Register pair rp = SP may not be specified.**

1		1		R		P		0		0		0		1
---	--	---	--	---	--	---	--	---	--	---	--	---	--	---

Cycles: 3
States: 10

Addressing: reg. indirect
Flags: none

POP PSW (Pop processor status word)
$$\begin{aligned}(\text{CY}) &\leftarrow ((\text{SP}))_0 \\ (\text{P}) &\leftarrow ((\text{SP}))_2 \\ (\text{AC}) &\leftarrow ((\text{SP}))_4 \\ (\text{Z}) &\leftarrow ((\text{SP}))_6 \\ (\text{S}) &\leftarrow ((\text{SP}))_7 \\ (\text{A}) &\leftarrow ((\text{SP}) + 1) \\ (\text{SP}) &\leftarrow (\text{SP}) + 2\end{aligned}$$

The content of the memory location whose address is specified by the content of register SP is used to restore the condition flags. The content of the memory location whose address is one more than the content of register SP is moved to register A. The content of register SP is incremented by 2.

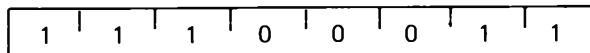
1		1		1		1		1		0		0		0		1
---	--	---	--	---	--	---	--	---	--	---	--	---	--	---	--	---

Cycles: 3
States: 10

Addressing: reg. indirect
Flags: Z,S,P,CY,AC

**XTHL** (Exchange stack top with H and L)
$$\begin{aligned}(L) &\longleftrightarrow ((SP)) \\ (H) &\longleftrightarrow ((SP) + 1)\end{aligned}$$

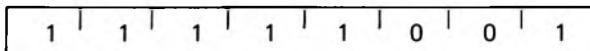
The content of the L register is exchanged with the content of the memory location whose address is specified by the content of register SP. The content of the H register is exchanged with the content of the memory location whose address is one more than the content of register SP.



Cycles: 5 Addressing: reg. indirect
States: 18 Flags: none

SPHL (Move HL to SP)
$$(SP) \leftarrow (H) (L)$$

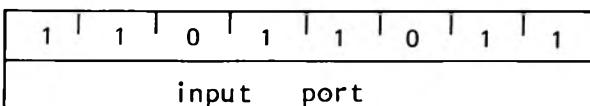
The contents of registers H and L (16 bits) are moved to register SP.



Cycles: 1 Addressing: register
States: 5 Flags: none

IN port (Input)
$$(A) \leftarrow (\text{data})$$

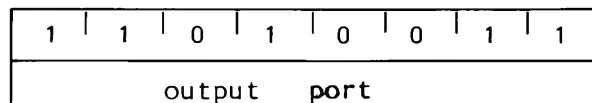
The data placed on the eight bit bidirectional data bus by the specified port is moved to register A.



Cycles: 3 Addressing: direct
States: 10 Flags: none

**OUT port** (Output)(data) \leftarrow (A)

The content of register A is placed on the eight bit bidirectional data bus for transmission to the specified port.



Cycles: 3

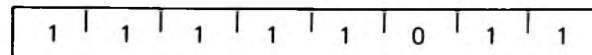
States: 10

Addressing: direct

Flags: none

EI (Enable interrupt)

The interrupt system is enabled **following the execution of the next instruction.**



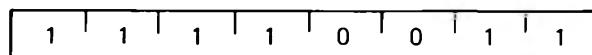
Cycles: 1

States: 4

Flags: none

DI (Disable interrupt)

The interrupt system is disabled **immediately following the execution of the DI instruction.**



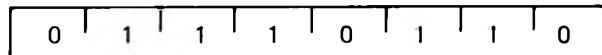
Cycles: 1

States: 4

Flags: none

**HLT** (Halt)

The processor is stopped. The registers and flags are unaffected.



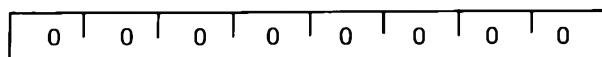
Cycles: 1

States: 7

Flags: none

NOP (No op)

No operation is performed. The registers and flags are unaffected.



Cycles: 1

States: 4

Flags: none

PSEUDO OPCODES/ASSEMBLER DIRECTIVES

The Heath Assembly Language supports 20 assembler directives or, as they are more commonly known, pseudo opcodes or simply pseudo ops. These opcodes are called "pseudo" because they are coded as machine operations. But as their alternate name (assembler directives) indicates, they represent commands to HASL-8 and are not translated as instructions for the H8. Some pseudo ops conditionally affect the operation of the assembler. Others cause the assembler to generate constants into the generated object code.



Define Byte, DB

The DB pseudo defines byte contents. The DB pseudo is of the form:

```
Label DB iexp1, . . . . ,iexpn
```

The integer expressions iexp1 through iexpn are expressions which evaluate to 8-bit values. For the DB pseudo, a long string can be substituted for an expression. The long string is a character string, delimited by single quotes ('), containing one or more characters. You can enclose a quote ('') within a string by coding it as two single quotes. Each of the expressions is converted into an 8-bit binary number and stored in sequential memory locations. A few examples of the DB pseudo are:

CR	EQU	15Q
LF	EQU	12Q
	DB	1
	DB	2,3,4
	DB	10,CR,LF,'H8 BASIC',0

In each case, the DB pseudo converts the expression into a single byte and stores it in the appropriate memory location. The DB pseudo recognized a character string as a series of expressions. Therefore, each character is converted into its ASCII binary equivalent and is stored in a sequential memory location.

Define Space, DS

The defined space pseudo (DS) reserves a block of memory during assembly.

The form of the DS pseudo is:

```
LABEL DS iexp COMMENT
```

This pseudo is used, for example, to set up a buffer area or to define any other storage area. The DS pseudo causes the assembler to reserve a number of bytes specified by the expression (iexp) in the operand. These bytes are not preset to any value. Therefore, you should not presume any special original contents. Programs using extensive buffer area should use the DS pseudo to declare this area. Using the DS pseudo significantly shortens the program load time. In the example

```
LINE DS 80 80 character input line buffer
```

an 80-character input buffer is reserved by a single statement.



Define Word, DW

The DW pseudo defines word constants. The form of the DW pseudo is:

```
LABEL    DW    iexp1, . . . . . , iexpn
```

The DW pseudo specifies one or more data words iexp through iexpn. Data words are **2-byte** values which are placed into memory space, low order byte first. NOTE: Strings greater than two characters long are not allowed when you are using the DW pseudo.

Conditional Assembly Pseudo Operators

Frequently, you may want to write a program with certain portions of it that can be turned on or turned off. That is to say, when they are turned on, these portions of the program are assembled. If they are turned off, they are not assembled during that particular assembly. HASL-8 contains three pseudos to aid in conditional assembly. They are:

IF ELSE and ENDIF

IF

The IF pseudo conditionally disables assembly of any statements following the IF pseudo operator. The form of the IF pseudo operator is:

```
IF    iexp
```

If the expression (iexp) evaluates to zero, the statements following the IF pseudo are assembled. If the expression does not evaluate to zero (either negative or positive), any statements in the assembly source code following this expression are skipped until one of the three following pseudos are encountered. The ELSE, ENDIF and END pseudos are not skipped regardless of the value of the expression "iexp".

ELSE

The ELSE pseudo toggles the state of the assembly conditions. The ELSE pseudo is of the form:

```
ELSE
```

If the conditional assembly flag is set to skip assembling source code, it is changed so source code is now assembled. If lines of source code prior to encountering the ELSE pseudo are being assembled, those following the ELSE pseudo are skipped until an ELSE, ENDIF, or END is encountered. NOTE: The ELSE segment must appear after an IF statement, but before the associated ENDIF statement.

**ENDIF**

The ENDIF statement indicates the end of a block of source code designated for conditional assembly. The form of the ENDIF pseudo is:

```
ENDIF
```

Assembly resumes regardless of the current assembly state (assembling or skipping) when the ENDIF conditional assembly pseudo occurs.

END

The END pseudo indicates the END of a program. The END pseudo takes the form:

```
END iexp
```

where iexp is the program entry point. The program entry point is the memory address where program execution begins. If the END statement is missing, the assembler generates one. If iexp is missing, the H8 does not receive a starting value for the program counter where the binary tape is loaded.

EQU

The Equate statement is used to assign an arbitrary value to a symbol. The form of the equate statement is:

```
LABEL EQU iexp
```

The equate statement is unique, as it must evaluate on pass one. For this reason, any symbols used within the expression "iexp" must be defined before the assembler encounters the EQU statements. The label is assigned the value of the integer expression "iexp". This label may not be redefined by subsequent use as a label in any other statement. For example,

```
.START EQU *
```

The label .START is set equal to the value of the memory location counter, or

```
.START EQU 100
```

The label .START is set equal to 100.

NOTE: If you omit the label, an error is generated.



ORG

The Origin statement (ORG) sets the initial value of the memory location counter. The form of the origin statement is:

```
LABEL ORG iexp
```

The expression iexp must evaluate on pass one. Therefore, any symbols used within this expression must be defined before the assembler encounters this statement. When the assembler encounters the ORG statement, the memory location counter is set to the expression value. All subsequent object code generated by the assembler is placed in sequential memory locations, starting at the address given by the expression. It is legal to establish a new origin, either before or after a previous origin. If a label is present, it is given the value iexp. For example:

BEGIN	ORG	40 100A	The program is started at location 040 100 (offset octal) and the label BEGIN is assigned the offset octal value 040 100.
BEGIN	ORG	.START+256	The memory location counter is set to the previously defined value of the label .START +256. The label BEGIN also assumes this value.

SET

The SET statement assigns an arbitrary value to a desired symbol. The form of the SET statement is:

```
LABEL SET iexp
```

The SET pseudo op differs from the EQU pseudo op in that any label defined in a SET statement can be redefined in a following SET statement as many times as desired in the course of the program. The expression "iexp" must evaluate during pass one. Therefore, any symbols used within the expression "iexp" must be previously defined.

Listing Control

HASL-8 provides a number of pseudo operators which affect the listing mode. They control paging, pagination, titles, and subtitles. The listing control pseudos are used to affect easily read documentation; they do not appear in the program listing.



TITLE

The pseudo operator TITLE causes a new page title to be used. The form of the title pseudo op is:

```
TITLE    'new title'
```

Unless the assembler is already at the top of a page, a new page of the assembly listing is generated. This page is given the title contained in the string 'new title'.

STL

The subtitle pseudo (STL) causes a new page subtitle to be set. The form of the subtitle pseudo is:

```
STL    'new subtitle'
```

The subtitle pseudo does not affect pagination. This is to say, it does not generate a new page but simply titles a subsection of the program. Subtitles are frequently used to indicate subroutines or major program modules.

EJECT

The EJECT pseudo causes a new page to be started. The form of the eject pseudo is:

```
EJECT
```

When HASL-8 processes an EJECT pseudo, the output device is instructed to move to the start of a new page during the listing.

SPACE

The space pseudo leaves blank lines in the program listing. The form of the space pseudo is:

```
SPACE    iexp1, iexp2
```

During the assembly listing, iexp1 blank lines are left. If the optional expression iexp2 is specified, the assembler checks during a listing to see if the number of lines remaining on the page is greater than or less than iexp2. If there are less than iexp2 lines remaining on the page, the spacing function is skipped and a new page is started, as if an EJECT pseudo was executed.



LON (Listing on)

The LON pseudo operator is used to turn-on listing options. The form of the LON pseudo is:

LON CCC

Each option is represented by a single character. The characters for the desired options are supplied as CCC. The options and their default modes (if they are not specified) are:

L Master listing

If this option is enabled, all program lines are listed. If it is disabled, only lines containing errors are listed.

DEFAULT MODE: All program lines are listed (normally enabled; disable using LOF).

I Lists the IF-skipped lines. When this option is enabled, all lines skipped due to IF statements are listed (although they are not assembled).

DEFAULT MODE: The skip lines are not contained in the listing.

G Lists all generated bytes. When this option is enabled, all generated bytes appear on the listing. If more than three bytes are generated by a statement, new lines are generated in the listing to display these bytes. NOTE: Define byte pseudo can produce many bytes when you are encoding a string. These are not normally listed.

DEFAULT MODE: Lists a maximum of the 3-bytes generated in each statement.

LOF (Listing off)

The LOF pseudo is identical to the LON pseudo except that the selected options are disabled. The form of the LOF pseudo is:

LOF CCC

See LON, above, for a description of the control character CCC.

ERRxx

HASL-8 contains four conditional error pseudo operators. These are of the form:

ERRZR	iexp
ERRNZ	iexp
ERRPL	iexp
ERRMI	iexp



For each of these pseudo operators, the assembler tests the indicated expression. If the expression matches the expressed error condition, an error code is flagged in the listing. The errors associated with each of the conditional error pseudos are:

ERRZR	tests for zero expression
ERRNZ	tests for non-zero expression
ERRPL	tests for positive expression
ERRMI	tests for negative expression

These pseudo error tests are particularly useful when you make assumptions about the configuration of various program elements or expressions. You can encode these assumptions into ERRxx pseudos. So any change which causes the code to fail generates an error, flagging the programmer during the listing. For example,

```

LXI H, AREA1
MOV B, M           (B) = (AREA1)
INX H
ERRNZ AREA2-AREA1-1 Assume area 2 follows area 1
MOV C, M           (C) = (AREA2)

```

If, when the program is assembled, AREA 1 and AREA 2 have been defined differently, an error flag would warn of this mistake.

USING THE ASSEMBLER

Before the Heath Assembly Language is used, the source program must be prepared using a Text Editor such as TED-8. Once the source program is prepared and stored on tape, Heath Assembly Language can be loaded in the H8. The loading procedure is outlined in "Appendix A" (Page 4-57). Once a configured version of HASL-8 is loaded and started, a series of questions must be answered. First the assembler asks

PAGE SIZE?

The normal page ($8\frac{1}{2} \times 11$) has 66 lines. You can specify longer or shorter page sizes by counting the total number of lines required to fill the page top to bottom. The assembler then asks

INTER-PAGE GAP SIZE?



The normal page allows the last six lines for the inter-page gap. However, you may specify any desired gap. The PAGE SIZE and INTER-PAGE GAP inputs are used when the title, subtitle, space and eject pseudo-ops are executed. The assembler then asks

LISTING PORT:

The normal reply to listing port is a carriage-return. A carriage-return with no port number specified indicates the listing port is to be the console terminal. If you specify, a port number, HASL-8 returns control to PAM-8 so you may configure the port as desired. Once the port is configured, HASL-8 is restarted at location 040 100. Once listing port is specified, the assembler asks

BINARY (Y/N)?

If you do not want a binary (N), the output tape transport is not used and no binary image of the assembled program is placed in memory. Often, no binary is specified until you are sure the program will assemble. If a yes (Y) is given in reply to the question, the assembler then asks

BINARY TAPE (Y/N)?

A no (N) reply to this question directs HASL-8 to place the binary generated from the assembly into memory at the proper location. If NO BINARY TAPE is specified, you should set the HIGH MEMORY limit below the point used by the object program. NOTE: To do this may require reconfiguration of the assembler. See "Appendix A," (Page 4-57).

A yes (Y) reply to this question directs HASL-8 to place the binary generated from the assembly of the source code onto tape at the dump port. This tape is in the memory image format and contains the starting and ending addresses, and the entry point address of the desired program.

Once the assembler determines whether a binary is to be generated or not, and if it is to be placed into memory or dumped onto tape, it then asks

INPUT



The response to this is the character string used to identify the source file when it was created by the Text Editor. Do not include any string delimiters to specify the file name when outputting from Text Editor. For example:

```
NEWOUT "TEST"  
FLUSH  
SURE?
```

dumps a file named TEST using TED-8. It is loaded by the assembler by

```
INPUT?TEST  
FOUND TEST
```

The file name does not have to be complete and can be a null, which allows HASL-8 to load the next file on the tape. (Enter a null by typing a range return.)

Once the file is found, HASL-8 begins the assembly process. The entire file is read for the first pass. Once the first pass is complete, HASL-8 issues the instruction

```
REWIND SOURCE TAPE TYPE CR WHEN DONE.
```

The tape drive is not turned off, so the source tape may be easily returned at its starting point. Once the tape is at its starting point, type a carriage return (CR) and start the tape transport. HASL-8 then issues the instruction

```
FOUND TEST  
POSITION PAPER. TYPE CR:
```

The paper in any printer on the H8 system should be in place and the dump tape transport should be made ready at this time. Position the paper at the bottom of a form. HASL-8 starts by spacing an interpage gap. It then prints the title and subtitles.

Once you type the carriage return, HASL-8 begins the second pass, generating the listing and creating the binary tape. The listing may require reading several records of the input tape and the output binary dump may come in a number of records. Once the listing and the binary dump are complete, HASL-8 terminates its operation by outputting

```
STATEMENTS = #####  
FREE BYTES - #####  
NO ERRORS DETECTED.      or
```

```
STATEMENTS = #####  
FREE BYTES - #####
```

```
ERRORS - #####
```



This first version of this terminating statement indicates that you have successfully completed an assembly of your source program, and if a binary output is specified it is generated. The second version of this terminating statement indicates that you have completed assembly of your source program, but there are errors which the assembler is able to detect. These errors exist in any binary output which may have been specified. Up to three errors per statement line will be shown on the listing output. The errors are shown as single letters in the left hand three columns of the listing. A typical output listing format is shown below.

```
HEATH/WINTEK H8 ASSEMBLER
ISSUE # 4.01.00.
COPYRIGHT WINTEK CORP., 01/77
```

```
.PAD = 4/0
.CONSOLE LENGTH = 00080/72
.HIGH MEMORY = 24575/
.LOWER CASE (Y/N)?  
.
```

```
HEATH H8ASM ISSUE #4.01.00.
```

```
PAGE SIZE? 60
INTER-PAGE GAP SIZE? 6
LISTING PORT:
BINARY (Y/N)?Y
BINARY TAPE (Y/N)?Y
INPUT?USR PROGRAM FOR BASIC #1.0
FOUND USR PROGRAM FOR BASIC #1.0
REWIND SOURCE TAPE. TYPE CR WHEN DONE.
FOUND USR PROGRAM FOR BASIC #1.0
POSITION PAPER. TYPE CR:
```

```
HASL #04.01.00
```

```
PAGE 1
```

Errors	Addresses	Object Code	Labeler	Opcodes	Operands	Comments
	117.220			ORG	120000A-160Q	
	063.207		FPNRM	EQU	063207A	
	117.220	003	START	INX	B	INC UP
	117.221	003		INX	B	TO
	117.222	003		INX	B	EXONENT
	117.223	012		LDAX	B	(A) = ACCX EXP
	117.224	247		ANA	A	SET CONDX CODE
	117.225	310		RZ		
	117.226	075		DCR	A	/2
	117.227	312 233 077		JZ	USR1	IF UNDER FLOW
	117.232	075		DCR	A	/2 AGAIN (/4)
	117.233	002	USR1	STAX	B	RET TO ACCX
	117.234	315 207 063		CALL	FPNRM	NORMALIZE
	117.237	311		RET		IN CASE 0
	117.240			END	START	

```
STATEMENTS = 00016
```

```
FREE BYTES - 10331
```

```
NO ERRORS DETECTED.
```



Errors

All errors detected by the Heath Assembly Language are flagged directly on the listing in the first three columns. One character is flagged for each error detected. If more than one error is detected, the second error character is placed in column 2 and the third error character is placed in column 3.

<u>CHARACTER</u>	<u>ERROR</u>
U	An undefined symbol. The symbol name does not match any symbol in the symbol assignment table. Check for spelling errors or for a completely undefined symbol.
R	Illegal register specified. Two different errors can cause this message. A non-8080 register may have been specified, or the instruction was not meaningful for the register. For example, a register pair instruction which refers to a single register.
D	Label is doubly defined. The symbolic label has been defined twice in the source program.
A	Operand syntax error. The operand expression is improper. For example, it may evaluate to a number >65535, be a divide by zero, or be nonexistent.
V	Value exceeds eight bits. The result of an expression is greater than 255. This error is not flagged if the op-code called for a 16-bit operand such as an LXI instruciton.
F	Format error. A pseudo-op requires a label that is not present in the source code. For example, an EQU pseudo-op requires a label. Too many characters in a label.
O	Unrecognized op-code. The op-code in this statement does not belong to the 8080 instruction set, nor does it belong to the HASL-8 pseudo-op instruction code set. Check for spelling errors or for op-codes used from other microprocessor instruction sets.



<u>CHARACTER</u>	<u>ERROR</u>
------------------	--------------

P	Error generated by ERRxx pseudo or reference to a doubly defined label. Note the ERRxx pseudos are generated to flag the user when a test expression does not evaluate satisfactorily.
---	--

NOTE: If an assembly generates a great number of errors, it is best to return to the Text Editor, correct as many errors as possible, and reassemble. The reassembly will frequently flag additional errors which are then obvious on the second assembly. If the errors are few, you may load the program and debug it using BUG-8 or PAM-8. However, this **does not** result in a correct listing.

During an Input, one of two error messages may be generated. They are:

SEQ ERR and
CHKSUM ERR.

A sequence error (SEQ ERR) is generated if the file records are not in the proper sequence. For example, if two consecutive label records are read, an error is generated, as a TED-8 Source file consists of a label record followed by text records. The form of the sequence error is

SEQ ERR

Typing a CNTRL-C after the SEQ ERR message generates a tape error message

TRY AGAIN?

Reply Y to TRY AGAIN? if you wish to try once more to read the tape. Rewind the tape until you are sure it is before the bad/missing record. HASL-8 will discard all records until the bad/missing one is located. Watch the record numbers on the H8 front panel LED's to make sure you don't miss the record again.

Reply N to TRY AGAIN? if you wish to restart the assembly completely.

A checksum error (CHKSUM ERR) is generated if the actual computed CRC for the record in question does not match the CRC recorded at the start of the record. The form of the checksum error message is

CHKSUM ERR IGNORE?

A Y in response to the question ignore aborts the error message and the next consecutive record is read. NOTE: Do not ignore the checksum error unless there is no other way to recover the data. If a checksum error is flagged, the chances are very good that the data in the designated record is faulty.



Control Characters

CONTROL-C

CONTROL-C is a general-purpose cancel key. Typing CNTRL-C causes HASL-8 to start over at the beginning.

RESTARTS

The H8 front panel keyboard can be used to restart the assembler if control has been returned to PAM-8. HASL-8 can be restarted in two places. They are:

NEW PASS #1 040 100 and;
NEW PASS #2 040 103.

OUTPUT SUSPENSION and RESTORATION, CNTRL-S and CNTRL-Q

Typing CONTROL-S during an output suspends the output to the terminal and suspends program execution. This command is particularly useful when you use a video terminal, since you can use the CONTROL-S or suspend feature each time a screen is nearly filled and information at the top is about to be lost due to scrolling.

Typing a CONTROL-Q permits HASL-8 to resume execution and outputting information to the terminal. The CONTROL-Q cancels the CONTROL-S function.

The DISCARD FLAG, CNTRL-O and CNTRL-B

Typing the CONTROL-O toggles the DISCARD FLAG. This stops output on the terminal but does not halt program execution until the program terminates. Typing a CONTROL-P (or retyping CONTROL-O) clears the discard flag. CONTROL-O is often used to discard the remainder of long listings and other similar outputs.



APPENDIX A

Loading Procedures

Loading From the Software Distribution Tape

1. Load the tape in the reader.
2. Ready the tape transport.
3. Press LOAD on the H8 front panel.
4. A single beep indicates a successful load.
5. Press GO on the H8 front panel.
6. The console terminal will respond with:

```
HEATH/WINTEK H8 ASSEMBLER
ISSUE #4.01.00
COPYRIGHT WINTEK CORP., 01/76
```

7. Configure HASL-8 as desired, answering the following questions. Prompt each question by typing its first character on the console terminal keyboard.

```
•AUTO NEW-LINE (Y/N)?
•BKSP = 00008/
•CONSOLE LENGTH = 00080/
•HIGH MEMORY = 16383/
•LOWER CASE (Y/N)?
•PAD = 4/
•RUBOUT = 00127/
•SAVE?
•
```

8. Before executing SAVE, have the tape transport ready at the DUMP port.
9. To use HASL-8 directly from the distribution tape, type the return key at any time rather than a question prompt key. HASL-8 responds

```
HEATH HASL-8 ISSUE #4.01.00.
```

```
*
```

Loading From a Configured Tape

1. Load the tape in the tape transport.
2. Ready the tape transport.
3. Press LOAD on the H8 front panel.
4. A medium beep indicates a successful load.
5. Press GO key on the H8 front panel.
6. The console terminal responds with:

```
HEATH HASL-8 ISSUE #4.01.00
BINARY (Y/N)←?Y
```

HASL-8 is ready to use in the configured form. Proceed to answer the question directing the desired assembly procedure.

INDEX

- Addressing Modes, 4-12
Arithmetic Instructions, 4-21 ff,
Assembler Directives, 4-43
Assembler Operations, 4-50
- Branch Instructions, 4-34 ff,
- Character Set, 4-4
Character Strings, 4-10
Comment Field, 4-4, 4-6
Condition Flags, 4-13
Conditional Assembly, 4-45
Control Characters, 4-56
- Data Transfer Instructions, 4-17 ff,
Define Byte (DB), 4-44
Define Word (DW), 4-45
Defined Space (DS), 4-44
Direct, 4-12
Dollar Sign (\$), 4-4
Doubly Defined Label, 4-54
- ERRxx, 4-49
EQU, 4-46
EJECT, 4-48
ELSE, 4-45
END, 4-46
ENDIF, 4-46
Errors, 4-54
Expressions, 4-7
- Format Control, 4-7
- I/O Instructions, 4-38 ff,
IF, 4-45
Illegal Register, 4-54
Immediate, 4-12
Integers, 4-8
- LOF, 4-49
LON, 4-49
Label Field, 4-4 ff,
Least Significant Bit (LSB), 4-11
Letters, 4-4
Listing Control, 4-47
- Logical Instructions, 4-28 ff,
Machine Control Instructions, 4-38 ff,
Most Significant Bit (MSB), 4-11
- Numerals, 4-4
- Opcode Field, 4-4 ff,
OPCODES (8080), 4-10 ff,
 Arithmetic Group, 4-21 ff,
 Branch Group, 4-35 ff,
 Data Transfer Group, 4-17 ff,
 Logical Group, 4-29 ff,
 Machine Group, 4-38 ff,
- Operating the Assembler, 4-50
- Operand Field, 4-4, 4-6
Operator Precedence, 4-8
Operators, 4-6
ORG, 4-47
Origin Symbol (ORG), 4-10, 4-47
Overflow Error, 4-9
- Period, (.), 4-4
Pound symbol, (#), 4-9
Pseudo Opcodes, 4-43
- Register, 4-12
Register Indirect, 4-12
- Set, 4-47
Space, 4-48
Stack Instructions, 4-38 ff,
Statements, 4-4
STL, 4-48
Strings, 4-10
Symbolic Programs, 4-3
Symbols, 4-9
Syntax Error, 4-54
- Text Editor, 4-3
Title, 4-48
Tokens, 4-8
TED-8, 4-3, 4-7
- Undefined Symbol, 4-54
Unrecognized Op-Code, 4-54

Section 5

BENTON HARBOR BASIC

And

**EXTENDED BENTON HARBOR
BASIC**



TABLE OF CONTENTS

INTRODUCTION

Manual Scope	5-6
Hardware Requirements	5-6
Loading and Running BASIC	5-7
Benton Harbor BASIC and Extended Benton Harbor BASIC	5-7
Command Completion	5-7

HEATH/WINTEK BASIC ARITHMETIC

Data Types	5-9
Variables	5-11
Subscripted Variables	5-12
Expressions	5-14
Arithmetic Operations	5-14
Relational Operators	5-18
Boolean Operators	5-19

STRING MANIPULATION

String Variables	5-21
String Operators	5-22

THE COMMAND MODE

Using The Command Mode For Statement Execution	5-23
--	------

BASIC STATEMENTS

Line Numbers	5-25
Statement Types	5-25
Command Mode Statements	5-27
Statements Valid In the Command or Program Mode	5-33
Program Mode Statements	5-58

PREDEFINED FUNCTIONS

Introduction	5-61
Arithmetic and Special Feature Functions	5-61
STRING Functions (Extended BASIC only)	5-68

EDITING COMMANDS

Control-C, CNTRL-C	5-71
Inputting Control	5-71
Outputting Control	5-72
Command Completion	5-72
Enforced Lexical Rules	5-73
General Text Rules	5-73



ERRORS

Error Message	5-75
Recovering from Errors	5-75

BASIC ERROR TABLE

BASIC ERROR TABLE	5-77
-------------------------	------

APPENDIX A

Loading From the Software Distribution Tape	5-81
Loading From a Configured Tape	5-82

APPENDIX B

Numeric Data	5-83
Boolean Data	5-83
String Data (Extended Basic Only)	5-83
Variables	5-83
Subscripted Variables	5-84
Arithmetic Operators	5-84
Relational Operators	5-84
Boolean Operators	5-85
String Variables	5-85
String Operators	5-85
Line Numbers	5-85
The Command Mode	5-85
Multiple Statements on One Line	5-85
Command Mode Statements	5-86
Command and Program Mode Statements	5-87
Program Mode Statements	5-90
Predefined Functions	5-91
Editing Commands	5-93

APPENDIX C

BASIC Utility Routines	5-95
------------------------------	------

APPENDIX D

Entry Points to Utility Routines	5-107
--	-------

APPENDIX E

An Example of USR	5-111
-------------------------	-------

INDEX

5-113



**TAB GUIDE**

BASIC ARITHMETIC	
STRING MANIPULATION	
THE COMMAND MODE	
BASIC STATEMENTS	
PREDEFINED FUNCTIONS	
EDITING COMMANDS	
ERRORS	
BASIC ERROR TABLE	
APPENDIX A	
APPENDIX B	
APPENDIX C	
APPENDIX D	
APPENDIX E	



INTRODUCTION

BENTON HARBOR BASIC is a conversational programming language which is an adaptation of Dartmouth BASIC*. (BASIC is an acronym for Beginners' All Purpose Symbolic Instruction Code.) It uses simple English statements and familiar algebraic equations to perform an operation or a series of operations to solve a problem. BENTON HARBOR BASIC is an interpretive language, compact enough to run in a Heath H8 computer with minimal memory, yet powerful enough to satisfy most problem-solving requirements. The interpretive structure of BASIC affords excellent facilities for the detection and correction of programming errors. It uses advanced techniques to perform intricate manipulations and to express problems more efficiently.

Two versions of BENTON HARBOR BASIC are available. Extended BENTON HARBOR BASIC (EX. B.H. BASIC) with strings provides character string manipulation and advanced functions. BENTON HARBOR BASIC (B.H. BASIC) does not have strings and some advanced functions, and so uses less memory. The user may operate B.H. BASIC in an H8 computer with 8K of memory.

Manual Scope

This Manual is written for the user who is already familiar with the language BASIC. It also describes the extended implementation of Dartmouth BASIC and, in so doing, provides a brief summary of the language. However, this manual is not intended as an instruction Manual for the language BASIC. If you are not familiar with BASIC, we suggest that you obtain the Heathkit Continuing Education course entitled "Basic Programming," Model EC-1100, before attempting to use this Manual.

Hardware Requirements

Extended BENTON HARBOR BASIC (with strings) runs on an H8 computer with a minimum of 12K bytes of random access memory. BENTON HARBOR BASIC (without strings) runs on an H8 with a minimum of 8K memory. Both versions require a console terminal, its appropriate interface card, and a mass storage device such as a cassette or paper tape reader/punch.

Both BASICs automatically measure the maximum amount of unbroken memory above the starting point at 8K (40,100 offset octal). They use all available memory unless the high memory limit is configured otherwise during the system configuration procedure (see "Product Installation" on Page 0-19 in the "Introduction" to this Software Reference Manual).

*BASIC is a registered trademark of the Trustees of Dartmouth College.



Benton Harbor BASIC and Extended Benton Harbor BASIC

This Manual covers both BENTON HARBOR BASIC (B.H. BASIC) and Extended BENTON HARBOR BASIC (EX. B.H. BASIC). Information that applies only to Extended BASIC is printed in a different type face, as shown below. For example:

Strings are only used in EX. B.H. BASIC, except in PRINT statements.

Anything not marked in this different type face applies to BASIC. References to "BASIC" apply to both BENTON HARBOR BASIC and to Extended BENTON HARBOR BASIC. BASIC is summarized in "Appendix B."

Loading and Running BASIC

BASIC is distributed in binary load format on cassette tapes or paper tape. It is loaded in accordance with the software configuration guide, outlined in "Product Installation" on Page 0-19. A condensed version of this loading procedure is given in "Appendix A" of this Manual. Once a system BASIC tape is configured, you can load the configured tape, using the internal PAM-8 loader, and start it by pressing the GO key. EX. B.H. BASIC and B.H. BASIC use the asterisk (*) as a prompt character.

Command Completion

Both the B.H. BASIC and EX. B.H. BASIC employ command completion. BASIC examines each character as you type it on the console keyboard, and when sufficient information is received to uniquely identify one particular command, BASIC finishes typing the command for you. For example, once the letters PR have been typed, the command PRINT is uniquely defined. Therefore, BASIC supplies letters INT and the required blank following the T.

BASIC also watches your spelling. As commands are being typed, letter combinations leading to non-existent commands are not accepted and the console terminal bell is rung.





BASIC ARITHMETIC

Data Types

BASIC supports three different data types:

1. Numeric data.
2. Boolean data.
3. String data.

NUMERIC DATA

BASIC accepts real and integer numbers. A real number contains a decimal point. BASIC assumes a decimal point **after** integer data. Any number can be used in mathematical expression without regard to its type. Real numbers must be in the approximate range of 10^{-38} to 10^{+37} . Integer numbers must lie in the range of 0 to 65535. All numbers used in BASIC are internally represented in floating point, which allows approximately 6.9 digits of accuracy. Numbers may be either negative or positive.

In addition to integer and real numbers, BASIC recognizes a third format. This format, called exponential notation, expresses a number as a decimal number raised to a power of 10. The exponential form is

$XXE(\pm)NN$

where E represents the algebraic statement “times ten to the power of,” XX represents up to a six digit integer or real number, and NN represents an integer from 0 to 38. Thus, the number is read as “XX times 10 to the \pm power of NN.”

Numeric data in all three forms may be used in the immediate mode, program mode in data statements, or in response to READ and INPUT statements.

Unless otherwise specified, all the numbers including exponents are presumed to be positive.



The results of BASIC computations are printed as decimal numbers if they lie in the range of 0.1 to 999999*. If the results do not fall in this range, the exponential format is used. BASIC automatically suppresses all leading and trailing zeros in real and integer numbers. When the output is in exponential format, it is in the form

(±) X.XXXXXE (±) NN

The following are examples of typical inputs and the corresponding output. Note the dropping of leading and trailing zeros, truncation to six places of accuracy, conversion to exponential notation when necessary, and conversion to decimal notation where permitted.

<u>INPUT NUMBER</u>	<u>OUTPUT NUMBER</u>	<u>COMMENTS</u>
0.1	.1	(leading zero dropped)
.0079	7.90000E-03	(<.1 converts to exponential)
0022	22	(leading zeros dropped)
22.0200	22.02	(trailing zeros dropped)
999999	999999	(format maintained)
1000000	1.00000E+06	(converted to exponential)
100000007	1.00000E+08	(truncated to 6 places)
-10.1E+2	-1010	(converted to decimal format)

BOOLEAN VALUES

Boolean values are a subclass of numeric values. Values representing the positive integers from 0-65,535 (2^{16-1}) may be used as Boolean data. When using numeric data as Boolean values, the numeric data represents the equivalent 16-bit binary numbers. Fractional parts of numeric data used with Boolean operators are discarded. If the numeric value with the fractional part does not fall into the range of 0-65,535, an illegal number error is generated.

STRING DATA (Extended BASIC Only)

Extended BASIC handles data in a character string format. Data elements of this type are made up of a string of ASCII characters up to 255 characters in length. Extended BASIC provides operators and functions to manipulate string data. String values in either programmed text or data must always be enclosed by quotation marks (""). Any printable ASCII character (with the exception of the quotation mark itself) may appear in an Extended BASIC string. In addition to the printable ASCII characters, the line feed and bell characters are also permitted. A string may not be typed on more than one line. A carriage return is rejected as an illegal string character.

*NOTE: This may be changed in EX. B.H. BASIC. See "CNTRL 1," Page 5-35.



Variables

A BASIC variable is an algebraic symbol representing a number. Variable naming adheres to the Dartmouth specification. That is, variable names consist of one alphabetic character which may be followed by one digit (zero to nine). The following is a list of acceptable and unacceptable variables, and the reason why the variable is unacceptable.

<u>ACCEPTABLE VARIABLES</u>	<u>UNACCEPTABLE VARIABLES</u>	<u>REASON FOR UNACCEPTABILITY</u>
C	2C	A digit cannot begin a variable.
A5	AF	A second character in a variable must be a number (0-9).
D	3	A single number is not an acceptable variable.
L2	\$2	The first character of a variable must be a letter (A-Z).

Subscripted variables, string variables, and subscripted string variables are permitted. See "Subscripted Variables," Page 5-12, and "String Manipulation" on Page 5-21.

A value is assigned to a variable when you indicate the value in a LET, READ, or INPUT statement. These operations are discussed in "LET" (Page 5-46), "PRINT" (5-50), and "INPUT AND LINE INPUT" (Page 5-59).

The value assigned to a variable changes each time a statement equates the variable to a new value. The RUN command sets all variables to zero (0). Therefore, it is only necessary to assign an exact value to a variable when an initial value other than zero is required.



Subscripted Variables

In addition to the variables described above, BASIC permits subscripted variables. Subscripted variables are of the form:

$A_n (N_1, \dots, N_8)$,

where A is the variable letter, n is a number (optional) 0-9, and N_1 thru N_8 are the integer dimensions of the variable. Subscripted variables provide you with the ability to manipulate lists, tables, matrices, or any set of variables. Variables are allowed one to eight subscripts.

The use of subscripts permits you to create multi-dimensional arrays of numeric and string variables. It is important to note that a dimensioned variable is distinguished from a scalar value of the same name. For example, all four of the following are distinct variables:

A , $A(N)$, $A\* , $A\$(N)^*$

When you are referencing a subscripted variable, each element in the subscript list may consist of an arbitrarily complex expression so long as it evaluates to a numeric value within the allowable range for the indicated dimension. Thus, the subscripted variable $A(5,5)$, would be dimensioned as:

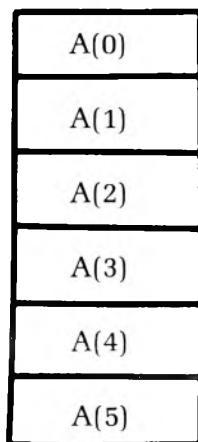
$X = A(2,3)$	is legal
$X = A(2\^{2}, VAL("4.0"))$	is legal as it is equivalent to $A(4,4)$
$X = A(2,"4.0")$	is not legal as ("4.0" is a string)

*NOTE: The \$ indicates a string variable valid only in EX. B.H. BASIC. See Section 3.

The following are graphic illustrations of simple subscripted variables. In these particular examples, a simple variable (A) is followed by one or two integer expressions in parentheses. For example,

A(I)

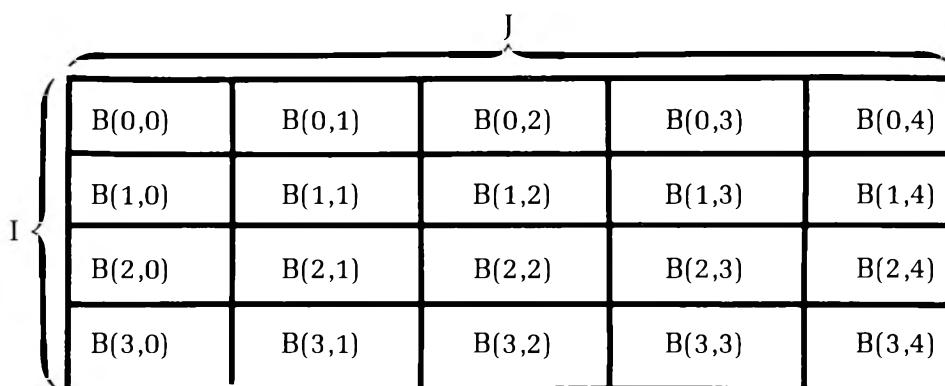
where I may assume the values of 0 to 5, allows reference to each of the six elements A(0), A(1), A(2), A(3), A(4), and A(5). A graphic representation of this 6-element, single-dimension array is shown below. Each box represents a memory location reserved for the value of the variable of the indicated name. Often, the entire array is referred to as A().



NOTE: Subscripted variables begin at zero. Therefore, the previous example 0 to 5 defines six elements.

A two dimensional array B(I, J) allows you to refer to each of the 20 elements B(0,0), B(0,1), B(0,2), . . . , B(0,J), , B(I,J).

This is graphically illustrated as follows, for B(3,4).



A 4x5 grid of 20 cells, labeled B(i,j) where i ranges from 0 to 3 and j ranges from 0 to 4. Brackets on the left and top indicate the row and column indices respectively. The grid is enclosed in a large bracket on the left labeled 'I' and a large bracket on the top labeled 'J'.

B(0,0)	B(0,1)	B(0,2)	B(0,3)	B(0,4)
B(1,0)	B(1,1)	B(1,2)	B(1,3)	B(1,4)
B(2,0)	B(2,1)	B(2,2)	B(2,3)	B(2,4)
B(3,0)	B(3,1)	B(3,2)	B(3,3)	B(3,4)

NOTE: A variable cannot be dimensioned twice in the same program. In EX. B.H. BASIC, a clear may be used to destroy an array, allowing you to reuse it.



BASIC does not presume any dimension. Therefore, the DIMension (DIM) statement must be used to define the maximum number of elements in any array. It is described in "DIM (DIMENSION)" on Page 5-36.

Expressions

An expression is a group of symbols to be evaluated by BASIC. Expressions are composed of numeric data, Boolean data, string data, variables, or functions. In an expression, these are alone or combined by arithmetic, relational, or Boolean operators.

The following examples show some expressions BASIC recognizes.

<u>ARITHMETIC EXPRESSIONS</u>	<u>BOOLEAN EXPRESSIONS</u>	<u>STRING EXPRESSIONS</u>	<u>DESCRIPTION</u>
1.02	255	"YES"	Data
1.02 + 16	255 OR 003	"YES" + "NO"	Combined
A < B		"YES" < "NO"	Relational

A major feature of BASIC is its extensive use of expressions in situations when many other BASICs only permit variables or numbers. This feature permits you to perform very sophisticated operations within a particular command or function. It is important to note that not all expressions can be used in all statements. The explanations describing the individual statements detail any limitations.

Arithmetic Operators

BASIC performs *exponentiation* (*in EX. B.H. BASIC only*), multiplication, division, addition, and subtraction. BASIC also supports two unary operators (– and NOT). The asterisk (*) is used to signify multiplication and the slash (/) is used to indicate division. *Exponentiation is indicated by the up arrow (^).*

THE PRIORITY OF ARITHMETIC OPERATIONS

When multiple operations are to be performed in a single expression, an order of priority is observed. The following list shows the arithmetic operators in order of descending precedence. Operators appearing on the same line are of equal precedence.

–(Unary)	(negation)
↑	(<i>exponentiation</i>)
* /	(multiplication division)
+ –	(addition subtraction)



Parentheses are used to change the precedence of any arithmetic operations, as they are in common algebra. Parentheses receive top priority. Any expression within parentheses is evaluated before an expression without parentheses. The innermost leftmost parenthetical expression has the greatest priority.

UNARY OPERATORS

BASIC supports two unary operators: – and NOT. These operators are referred to as unary because they require only one operand. For example:

```
A = -2  
C = NOT D
```

The unary operator (–) performs arithmetic negation. The NOT operator performs Boolean negation. See Page 5-19.

EXPONENTIATION (EXTENDED BASIC ONLY)

Exponentiation (\uparrow) is used to raise numeric or variable data to a power. For example:

*A = B \uparrow 2 is equivalent to A = B * B.*

NOTE: The operand must not be negative. The exponent may be negative. A negative operand generates a syntax error. For greatest efficiency, B \uparrow 2 should be written as B B and B \uparrow 3 should be written as B * B * B. All other powers should use the \uparrow .*

MULTIPLICATION AND DIVISION

BASIC uses the asterisk (*) and the slash (/) as symbols to perform the algebraic operations of multiplication and division respectively. Both multiplication and division require numeric data as operands.

The following examples use the multiplication and division operators:

```
*PRINT 2*6  
12  
  
*PRINT 6/3  
2  
  
*PRINT 6/3*2  
4  
  
*
```

*NOTE: This last expression evaluates to 4, not 1; as * and / have equal precedence and therefore the leftmost operator is evaluated first.*



ADDITION AND SUBTRACTION

The plus sign (+) and the minus sign (-) perform arithmetic addition and subtraction. *In addition, the plus operator (+) performs string concatenation if both operands are string data. Concatenation is restricted to Extended BASIC.* The following examples use the plus and minus operators:

```
*PRINT 3
3
*PRINT 3+5
8

*PRINT 10-3
7

*PRINT "HEATH" + " " + "H8"
HEATH H8
*                                         } extended BASIC only
```

SUMMARY

In any given expression, BASIC performs arithmetic operations in the following order:

1. Parentheses have top priority. Any expression in parentheses is evaluated prior to a nonparenthetical expression.
2. Without parentheses, the order of priority is:
 - a. Unary minus and NOT (equal priority).
 - b. *Exponentiation (proceeds from left to right).*
 - c. Multiplication and division (equal priority, proceeds from left to right).
 - d. Addition and subtraction (equal priority, proceeds from left to right).
3. If the rules in either 1 or 2 do not clearly designate the order of priority, the evaluation of expression proceeds from left to right.

The following examples illustrate these principles. *The expression $2 \uparrow 3 \uparrow 2$ is evaluated from left to right:*

1. $2 \uparrow 3 = 8$ (*left-most exponentiation has highest priority*).
2. $8 \uparrow 2 = 64$ (*answer*).



The expression $12/6*4$ is evaluated from left to right since multiplication and division are of equal priority:

1. $12/6 = 2$ (division is the left-most operator).
2. $2*4 = 8$ (answer).

The expression $6+4*3^2$ evaluates as:

1. $3^2 = 9$ (*exponentiation has highest priority*).
2. $9*4 = 36$ (multiplication has second priority).
3. $36+6 = 42$ (addition has lowest priority; answer).

Parentheses may be nested, (enclosed by additional sets of parentheses). The expression in the innermost set of parentheses is evaluated first. The next innermost left justified is second, and so on, until all parenthetical expressions are evaluated. For example:

$6 * ((2^3+4)/3)$

Evaluates as:

1. $2^3 = 8$ (*exponentiation in parentheses has highest priority*).
2. $8+4 = 12$ (addition in parentheses has next highest priority).
3. $12/3 = 4$ (next innermost parentheses are evaluated).
4. $4*6 = 24$ (multiplication outside of parentheses is lowest priority).

Parentheses prevent confusion or doubt when you are evaluating the expression. For example, the two expressions

$D^E^2/4+E/C^A^2$
 $((D^((E^2))/4)+((E/C)*(A^2)))$

are executed identically. However, the second is much easier to understand.

Blanks should be used in a similar manner, as BASIC ignores blanks (except when they are part of a string enclosed in quotation marks). The two statements:

```
10 LET B = 3 * 2 + 1  
10 LET B=3*2+1
```

are identical. The blanks in the first statement make it easier to read.



Relational Operators

Relational operators compare two variables or expressions. They are generally used with an IF THEN statement. The result of a comparison by the relational operators is either a true or a false. A false is represented by zero, and true is represented by 65535 ($2^{16}-1$). NOTE: These values are chosen so when they are used as Boolean values, false is all zeros and true is all ones.

The following table lists relational operators as used in BASIC.

ALGEBRAIC SYMBOL	BASIC SYMBOL	EXAMPLE	MEANING
=	=	A=B	A is equal to B.
<	<	A<B	A is less than B.
\leq	\leq	A \leq B	A is less than or equal to B.
>	>	A>B	A is greater than B.
\geq	\geq	A \geq B	A is greater than or equal to B.
\neq	\neq	A \neq B	A is not equal to B.

The symbols $=<$, $=>$, $><$ are not accepted and BASIC generates a syntax error if they are used.

The following examples show the results of using relational operators.

```
*PRINT 3<4      (true)
65535
```

```
*PRINT 4<3      (false)
0
```

EX. B.H. BASIC and B.H. BASIC differ from most other versions in the use of the relational operator. When you are using BASIC, you may use the relational operators in any expression. When the expression is evaluated, the appropriate numeric answer (0 or 65535) will be used as the answer to that expression.



Boolean Operators

OR

The operator OR performs a Boolean OR on the two integer operands. The integer operands (which must lie in the range of 0 to 65535) are converted to 16-bit binary numbers. The Boolean (logical) 16-bit OR is applied and the result is returned to the equivalent integer representation. NOTE: As the Boolean value chosen to represent true (65535) and false (0), the OR operator implements a standard truth table OR function. For example:

*PRINT 132 OR 255	00000000 10000100	132
255	00000000 11111111	255
	<hr/>	
	00000000 11111111	255

and

```
*PRINT (3>2) OR (4>9)  
65535
```

AND

The AND operator performs a Boolean (logical) AND on the two integer operands. These integer operands must lie in the range of 0 to 65535. The integer operands are converted into 16-bit binary numbers and the logical AND is performed. The result is returned to the equivalent integer representation. NOTE: The AND operator implements a standard AND truth table on the values true (65535) AND false (0). For example:

*PRINT 132 AND 255	00000000 10000100	132
132	00000000 11111111	255
*	<hr/>	
	00000000 10000100	132

and

```
*PRINT (3>2) AND (9>7)  
65535
```

NOT

The NOT operator Boolean negation. That is, the numeric value of the variable is converted into a 16-bit Boolean data value; each bit is inverted, and the 16-bit binary number is restored to numeric data. For example:

```
*PRINT NOT 0 0 = 00000000 00000000 and  
65535 65535 = 11111111 11111111  
*
```





STRING MANIPULATION

Extended BENTON HARBOR BASIC is capable of manipulating string information. A string is a sequence of characters treated as a single unit of an expression. It can be composed of alphanumeric and other printing characters. An alphanumeric string contains letters, numbers, blanks, or any combination of these characters. A character string may not exceed 255 characters. The blank, bell, and line feed are considered to be printing characters.

String Variables

The dollar sign (\$) following a variable name indicates a string variable. For example:

B\$
and
L6\$

are string variables. A string variable (B\$) is used in the following example.

```
*B$ = "HI": PRINT B$
```

HI

NOTE: The string variable B\$ is separate and distinct from the variable B.

Any array name followed by the \$ character notes that the dimensioned variable is a string. For example:

L\$(n) A2\$(n) (single-dimensioned string variables).
D\$(m,n) H1\$(m,n) (multiple-dimensioned string variables).

The numbers in parentheses indicate the location within the array. See "Subscripted Variables," Page 5-12.

The same variable can be used as a numeric variable and as a string variable in one program. For example, each of the following is a different variable:

B B(n)
B\$ B\$(m,n)

The following are illegal, as they are double declarations of the same variable.

A\$(n) A\$(n,m)

String arrays are defined with a dimension (DIM) statement in the same way numerical arrays are defined.



String Operators

Extended BASIC provides you with the ability to manipulate strings. The string manipulation operators are: plus (+), for concatenation, and the relational operators.

CONCATENATION

Concatenation connects one string to another without any intervening characters. This is specified by using the plus (+) symbol and only works with strings. The maximum range of a concatenated string is 255 characters. For example:

```
*PRINT "THE HEATH" + " H8 COMPUTER"
THE HEATH H8 COMPUTER
```

RELATIONAL OPERATORS FOR STRINGS

Relational operators, when applied to strings, indicate alphabetic sequence. The relational comparison is done on the basis of the ASCII value associated with each character, on a character-by-character basis, using the ASCII collating sequence. A null character (indicating that the string is exhausted) is considered to head the collating sequence. For example:

```
*PRINT "ABC" < "DEF"
65536      (The relation shown is true)
*PRINT "ABC">"ABCD"
0          (The relation is false. "ABC" is less than "ABCD.")
```

NOTE: In any string comparison, trailing blanks are not ignored. For example:

```
*PRINT "CDE" = "CDE "
0      (The equality is false.)
```

The following table indicates how relational operators are used with string variables in Extended BASIC.

OPERATOR	EXAMPLE	MEANING
=	A\$ = B\$	String A\$ and B\$ are alphabetically equal.
<	A\$ < B\$	String A\$ is alphabetically less than B\$.
>	A\$ > B\$	String A\$ is alphabetically greater than B\$.
< =	A\$ < = B\$	String A\$ is equal to or less than B\$.
> =	A\$ > = B\$	String A\$ is equal to or greater than B\$.
< >	A\$ < > B\$	String A\$ and B\$ are not alphabetically equal.



THE COMMAND MODE

Using The Command Mode For Statement Execution

You may solve a problem in BASIC by using a complete program or by use of the **command** mode. **Command** mode makes BASIC an extremely powerful calculator.

Lines of program material entered for later execution are identified by line numbers. BASIC identifies those lines entered for immediate execution by the absence of the line number. That is to say, statements that begin with line numbers are stored, and statements without line numbers are executed immediately when a carriage return is received. For example^{*}:

```
10 PRINT "THIS IS AN H8 COMPUTER"
```

is not executed when it is entered at the console terminal. However, the statement:

```
*PRINT "THIS IS THE HEATH H8 COMPUTER"
```

when the RETURN key is typed, is immediately executed as:

```
THIS IS THE HEATH H8 COMPUTER
```

The **command** mode of operation is useful in program de-bugging and performing simple calculations which do not justify the writing of a complete program.

For example, in order to facilitate program de-bugging, you may place STOP statements liberally throughout a program. Once BASIC encounters a STOP statement, the program halts. You can examine and change data values using the **command** mode. The statement

```
CONTINUE
```

is used to continue execution of the program. You can also use the GOSUB and IF commands. Values assigned to variables remain intact using this technique. A SCRATCH, CLEAR, or another RUN command resets these values.

^{*}NOTE: Strings may be used in B.H. BASIC PRINT statement. However, these strings cannot be manipulated in B.H. BASIC.



The ability to place multiple statements on a single line is an advantage in the **command** mode. For example:

```
*B = 2:PRINT B:PRINT B + 1  
2  
3  
*
```

Program loops are allowed in the **command** mode. For example, a table of squares can be produced as follows:

```
*FOR A = 1 TO 10:PRINT A,A * A:NEXT A  
1 1  
2 4  
3 9  
4 16  
5 25  
6 36  
7 49  
8 64  
9 81  
10 100  
*
```

Some statements cannot be used in the **command** mode. The INPUT statement, for example, is not available in the **command** mode, and its use results in the USE error message. There are certain command functions in the **command** mode which make no sense when used in the **command** mode. Statements available in the **command** mode are covered in “Command Mode Statements” on Page 5-27 and “Statements Valid in the Command or Program Mode” on Page 5-33.



BASIC STATEMENTS

A program is composed of one or more lines or "statements" instructing BASIC to solve a problem. Each program line begins with a line number identifying the line and its statement. The line number indicates the desired order of statement execution. Each statement starts with an English word specifying the operation to be performed. Single statements are terminated with the return key. Multiple statements are separated by a colon (:) with the last statement terminated by a return (a non-printing character). A DATA statement cannot share a line with other statements. (See Page 5-54.)

Line Numbers

An integer number begins each line in a BASIC program. BASIC executes the program statements in numerical sequence, regardless of the input order. Statement numbers must lie in the range of 1 to 65,535. It is good programming practice to number lines in increments of 5 or 10 to allow insertion of forgotten or additional statements when de-bugging the program.

The length of a BASIC statement must not exceed one line. There is no method to continue a statement to a following line. However, multiple statements may be written on a single line. In this situation each statement is separated by a colon. For example:

```
10 PRINT "VALUES",A,A+1 is a single line print statement, whereas  
10 LET A=12: PRINT A,A+1,A+2 is a line containing two statements, LET and PRINT.
```

Virtually all statements can be used anywhere in a multiple statement line. There are, however, a few exceptions. They are noted in the discussion of each statement. NOTE: Only the first statement on a line can have a line number. Program control cannot be transferred to a statement **within** a line, but only to the beginning of a line.

Statement Types

BENTON HARBOR BASIC supports three different types of statements. First, there are statements valid only in the command mode. These statements are used for loading programs, erasing memory, and other such functions directing BASIC's activities. Second, there are statements valid as both commands or within a program. Third, there are statements valid only within a program. These statements may not be used in the command mode. Most statements fall into the second category. This means they can appear within a program or be typed directly in the command mode and immediately executed.



As noted earlier, some statements valid in both modes may not be meaningful in both modes.

BASIC is designed to allow maximum versatility in its structure. Thus, almost everywhere that BASIC requires a number or a string, BASIC allows a numeric or a string **expression**. For example, you can construct a computed GOTO by simply computing a value for a variable, X. The statement

```
GOTO X
```

then redirects the program to the computed line number.

The following three sections are organized as command mode statements, command and program mode statements, and program mode statements. They can be found, respectively in: "Command Mode Statements" (below), "Statements Valid in the Command or Program Mode" (Page 5-33), and "Program Only Statements" (Page 5-58).

To simplify some practical descriptions in these sections and those following, the notation below is used to describe allowed expressions:

1. "iexp" indicates an integer expression, an expression lying in the range of 0 to 65535. The fractional part of any integer expression is discarded when the integer is formed.
2. "nexp" indicates a numeric expression. This may be an integer, decimal, or exponential expression with up to 6 decimal places.
3. "*sexp*" indicates a string expression. *String expressions are limited to a maximum of 255 printing ASCII characters. String expressions are limited to Extended BASIC.*
4. "sep" indicates a separator. Separators such as the comma and the semi-colon are used to delineate certain portions of BASIC statements.
5. "[]" brackets indicate optional portions of a statement, depending on the exact function desired.
6. "var" indicates a variable. This may be a numeric or string variable, depending upon the example.
7. "name" indicates a string used to identify a date, a program, or a language record.



Command Mode Statements

The command mode statements cannot be used within a program. For example, the RUN statement cannot be used within a program to make it self-starting. Any attempt to incorporate one of these statements within a program generates a USE error message.

BUILD

This statement is used to insert or replace many program lines. The form of the BUILD statement is:

```
BUILD iexp1, iexp2
```

When BUILD is executed, the initial line number iexp1 is displayed on the terminal. Any text entered after the new line number is displayed becomes the new line, replacing any pre-existing line. Once the line is completed by a carriage return, the next line number is displayed. NOTE: If a null entry is given (a carriage return typed directly after the line number is displayed), the line whose number is displayed is eliminated if it existed.

BUILD is illustrated in the following example. CONTROL-C terminates BUILD.

```
*BUILD 100,10
 100 PRINT "LINE 100"
 110 PRINT "LINE 110"
 120 PRINT "LINE 120"
 130 < CNTRL-C >      (Control-C typed here)
*LIST
 100 PRINT "LINE 100"
 110 PRINT "LINE 110"
 120 PRINT "LINE 120"
*
```

CONTINUE

CONTINUE begins or resumes the execution of a BASIC program. CONTINUE has the unique feature of not affecting any existing variable values, nor does it affect the GOSUB or FOR stack. CONTINUE is normally used to resume execution after an error the program or after a CONTROL-C stops the program. CONTINUE may be used to enter a program or a specific line (in conjunction with a GOTO). CONTINUE is unlike RUN, which resets all variables, stacks, etc.. The form of the CONTINUE statement is:

```
CONTINUE
```



In the following example, CONTINUE starts the program at a specific line number.

```
*GOTO 100
*CONTINUE      (start execution at line 100)
```

CONTINUE is also useful for entering a program with a variable or variables set at particular values. For example:

```
*A = 23.5      (Program continues execution at Line 230
*GOTO 230      with variable A set to the value 23.5,
*CONTINUE      regardless of previous program effects on A.)
```

DELETE

The **DELETE** statement is used to remove several lines from the BASIC source program. The form of the **DELETE** statement is:

```
DELETE iexp1, iexp2,
```

The lines between and including *iexp1* and *iexp2* are deleted.

A syntax error is flagged if "*iexp1*" is greater than "*iexp2*." Normally, **DELETE** is used to eliminate a number of lines of text. The **SCRATCH** command is used to eliminate all text. A **RETURN** typed directly after a line number eliminates that line. This technique is used to eliminate a single line.

DUMP

The **DUMP** statement saves the current program text on the mass storage media connected to the load/dump port. This is usually paper tape or cassette. The current program is saved; however, no variables are saved. The specific program name is written with the data so the user may reload the program by the specified name. The form of the **DUMP** statement is:

```
*DUMP "name"
```

Make the tape drive ready before entering the **DUMP** statement. BASIC starts the drive, writes the data, and stops the drive. The **CONTROL-C** can be used to abort the **DUMP** while in progress. However, if a **DUMP** is aborted, an incomplete file exists on the tape.



The string "name" may be up to 80 ASCII characters. The normal string ASCII characters are permitted. An example of a DUMP is:

```
*DUMP "STARTREK VER 1.0 03/11/77"
```

This statement dumps the program Startrek version 1.0 dated the 11th of March 1977.

LOAD

The LOAD statement discards the current program in memory. A specified program is loaded from the mass storage device connected to the load/dump port. The form of the LOAD statement is:

```
LOAD "name"
```

The string "name" may consist of up to 80 ASCII characters. The normal string ASCII characters are permitted. BASIC scans the mass storage device until it finds a program whose name matches the specified string. Before destroying the stored information, the user is asked "SURE?." A "Y" reply causes LOAD to proceed. Any other response cancels LOAD.

NOTE: If the name on the mass storage device is longer than the specified name, a match on the supplied characters in the string "name" is valid. Thus, a program may be dumped with extra information entered in the name such as program version number and data. The program can then be loaded without it. For example:

```
*DUMP "STARTREK VER 1.0 03/11/77"
```

This program, Startrek version 1.0 dated 11 March 1977, may be loaded by

```
*LOAD "STARTREK"  
SURE? Y
```

A match is found between the first eight characters of the DUMP string "STARTREK" and the eight characters of the load command "STARTREK." If a null is used as the load string, the next program on the tape is loaded. Therefore, the statement

```
*LOAD ""
```

loads the next BASIC program appearing on the mass storage.



Mass storage media should be made ready before LOAD is executed. BASIC starts and stops the mass storage device. CONTROL-C may be used to abort the load part way through. Use a SCRATCH command to clear the results of an aborted load.

During a load, either one of the following two error messages may be generated:

SEQ ERR and
CHKSUM ERR.

A sequence error (SEQ ERR) is generated if the file records are not in sequence. For example, if two consecutive label records are read an error is generated, as a BASIC file consists of a label file followed by a data file. The form of the sequence error is

SEQ ERR

Type a blank after the SEQ ERR message. This will clear the error. The entire file must be reread.

A checksum error (CHKSUM ERR) is generated if the computed CRC for the record in question does not match the CRC included in the record. The form of the checksum error message is

CHKSUM ERR IGNORE?

A Y in response to the question "ignore" aborts the error message and the next consecutive record is read. Do not ignore the checksum error unless there is no other way to recover the data. If a checksum error is flagged, the chances are very good that the data in the designated record is faulty. If you attempt to use such bad data, it may cause BASIC to crash.

RUN

A prepared program may be executed using the RUN statement. The program is executed starting at the lowest numbered statement. All variables and stacks are cleared (set to zero) before program execution starts.

The form of the RUN statement is:

*RUN



After program completion, BASIC prompts the user with an asterisk (*) in the left margin, indicating that it is ready for additional command statements. If, the program should contain errors, an error message is printed that indicates the error and the line number containing the error, and program execution is terminated. Again, a prompt is given. The program must now be edited to correct the error and rerun. This process is continued until the program runs properly without producing any error messages. See "Errors" (Page 5-75) for a discussion of error messages.

Occasionally a program contains an error that causes it to enter an unending loop. In this case, the program never terminates. The user may regain control of the program by typing CONTROL-C (CNTRL-C). This aborts the program and returns control to the user. Storage is not altered in this process. CONTINUE resumes program execution. RUN clears the storage and restarts program execution.

SCRATCH

SCRATCH clears all current storage areas used by BASIC. This deletes any commands, programs, data, strings, or symbols currently stored by BASIC.

SCRATCH should be used for entering a new program from the terminal keyboard to ensure that old program lines are not mixed with new program lines. It also assures a clear symbol table. The form of the SCRATCH statement is:

*SCRATCH

Before destroying stored information, the user is asked "SURE?" A "Y" reply causes SCRATCH to proceed. Any other response cancels SCRATCH. For example:

*SCRATCH	(Scratch statement entered.)
SURE? Y	(Are you sure, answer Y (YES,)
*	(BASIC is ready for a new entry.)

VERIFY

The VERIFY statement permits you to check a file placed on mass storage without affecting the current program. The VERIFY command responds by indicating the name of the file found, and if the file is correct. A form of the VERIFY command is:

*VERIFY "name"



The string "name" can be the name of the record the user desires to verify or it may be a null (""); in which case, BASIC verifies the first record encountered. For example,

```
*VERIFY "STARTREK"  
FOUND STARTREK VER 1.0 03/11/77  
FILE OK  
*
```

In the above example, the file containing the Startrek dump is verified. Note, that the name of the file is printed immediately as soon as the label record is encountered. The FILE OK message is printed after the data record is read and verified. VERIFY performs a checksum on the contents of all data in the file. Using the VERIFY command does not destroy any program data in memory.

During a VERIFY, one of two error messages may be generated. They are:

SEQ ERR and
CHKSUM ERR.

A sequence error (SEQ ERR) is generated if the file records are not in sequence. For example, if two consecutive label records are read an error is generated, as a BASIC file consists of a label file followed by a data file. The form of the sequence error is

SEQ ERR

Type a blank after the SEQ ERR message. This will clear the error. The entire file must be reread.

A checksum error (CHKSUM ERR) is generated if the computed CRC for the record in question does not match the checksum included in the record. The form of the CRC error message is

CHECKSUM ERR - IGNORE?

A Y in response to the question ignore aborts the error message and the record is considered valid. Do not ignore the checksum error unless there is no other way to recover the data. If a checksum error is flagged, the chances are very good that the data in the designated record is faulty.

The command VERIFY is not available if BASIC is patched to use an ASR console terminal as the load dump device. See "Product Installation" on Page 0-19 of the "Introduction" to this Software Reference Manual.



Statements Valid in the Command or Program Mode

The statements in this section may be used in either the command or the program mode. A few of them have only subtle uses in one mode or the other. Because they may be used in both modes, they are listed in this section.

CLEAR

CLEAR sets the contents of all variables, arrays, string buffers, and stacks to zero. The program itself is not affected. The command is generally used before a program is rerun to insure a fresh start if the program is started with a command other than RUN. The form of the CLEAR statement is:

*10 CLEAR	(BASIC)
10 CLEAR varname	(EXTENDED BASIC)

All variables, arrays, string buffers, etc., are cleared before program is executed by RUN. Therefore, a clear statement is not required. However, a program terminated prior to execution (by a STOP command or an error) does not set these variables, etc., to zero. They are left with the last value assigned. *If the variable name (varname) is specified, the CLEAR command clears the named variable, array, or DEF FN (user defined function).*

Note that the memory space used by string variables and arrays is not freed when CLEAR varname is used. String values should be set to null (for example, A \$ = "") before clearing so the string space can be recovered.

For example:

CLEAR A	Clears variable A
CLEAR A\$	Clears the string variable A\$
CLEAR A(Clears the dimensioned variable A(

If a section of the program is to be rerun after appropriate editing, the variables, arrays, dimensions, etc., should be reinitialized. You can accomplish this by using the CLEAR statement in the command mode.



CNTRL (CONTROL)

CONTROL is a multi-purpose command used to set various options and flags. The form of the CONTROL statement is:

```
CNTRL iexp1, iexp2
```

The various CNTRL options are:

iexp1	iexp2
-------	-------

CNTRL	0,	nnn
CNTRL	1,	n
CNTRL	2,	n
CNTRL	3,	n
CNTRL	4,	n

CNTRL 0

The CNTRL 0, nnn command sets up a GOSUB routine to process CONTROL-B characters. The line number of the routine is specified as "iexp2." When a CONTROL-B is entered from the terminal program, control is passed to the specified statement (beginning at the line iexp2) via a GOSUB linkage, after the statement being executed is completed. For example:

```

10 CNTRL 0,500
20 FOR A=1 TO 9
30 PRINT A,A*A,A*A*A
40 NEXT A
50 END
500 PRINT "THAT TICKLES"
510 RETURN

```

*RUN

1	1	1
2	4	8
3	9	27 (CONTROL-B typed)

THAT TICKLES		
4	16	64 (CONTROL-B typed)

THAT TICKLES		
5	25	125
6	36	216
7	49	343
8	64	512
9	81	729

END AT LINE 50

*



During the execution of the program containing these three statements, a CONTROL-B from the keyboard momentarily interrupts the regular execution of the program. The program completes the line in progress and then enters the subroutine at line 500 printing the string

THAT TICKLES

It then moves to the next statement, a RETURN. This causes the program to continue with normal program execution. NOTE: The CNTRL 0, nnn must be executed before it is operational.

CNTRL 1

The CNTRL 1, n command sets the number of digits permitted before the exponential notation is used. Normal mode M = 6. For example:

CNTRL 1,2 (Numbers ≥ 100 are to be in exponential format.)

```
*PRINT 101  
1.01000E+02
```

CNTRL 2

The CNTRL 2, n command controls the H8 front panel LED display mode. The control functions are:

CNTRL 2,0 Turn display off (Normal mode).

CNTRL 2,1 Turn display on without update. (For writing into a display. See the example under "The SEG Function, SEG (NARG)" on Page 5-65.)

CNTRL 2,2 Turn display on with update (to monitor a register or memory location).

CNTRL 3

The CNTRL 3, n command controls the size of a print zone. This is normally 14. However, CNTRL 3, n can change the number of spaces in a print zone.

```
*
```

```
*CNTRL 3,5  
*PRINT 1,2,3,4,3,2,1,0  
1    2    3    2    3    4    3    2    1    0
```



CNTRL 4

The CNTRL 4,n command turns the hardware clock on and off. CNTRL 4,0 turns the clock off and CNTRL 4,1 turns it on. Once the clock has been turned off, clock dependent functions such as PAUSE iexp, and PAD(cannot be used. Turning the clock off increases execution speed approximately 11%.

NOTE: The CNTRL 1 through CNTRL 4 commands permanently reconfigure loaded BASIC. A new program does not clear them to the original state. You can reset them to their original state by using the appropriate form of the Control statement in the Command mode.

DIM (DIMENSION)

The DIMENSION statement explicitly defines the maximum dimensions of array variables. A single dimension array is often called a vector. The form of the DIMENSION statement is:

```
*DIM varname (iexp1[ . . . . . , iexpn]) [.varname2 ( . . . . )]
```

The expressions “iexp1” through “iexpn” are integer expressions specifying the bounds of each dimension. Dimensions are 0 to “expn.” So, for example, the statement:

```
DIM A(5,5)
```

reserves an array 6×6 or 36 values. If the dimensioned variable is numeric, the values are preset to zero. If the dimensioned variable is a string, all the values are preset to a null string.

You may declare several variables in one DIMENSION statement by separating them with commas. For example:

```
*DIM A6(3,2), B(5,5), C3(10,10)
```

dimensions the following arrays

<u>VARIABLE</u>	<u>SIZE</u>	
A6	4 by 3	12 elements
B	6 by 6	30 elements
C3	11 by 11	121 elements



You can place a DIMENSION statement anywhere in a multiple statement line and it can appear anywhere in the program. However, an array can only be dimensioned once in a program unless it is cleared. DIMENSION statements must be executed before the first reference to the array, although good programming practices place all DIMENSION statements in a group among the first statements of a program. This allows them to be easily identified and changed if alterations are required later. The following example demonstrates the use of the DIMENSION statement with subscripted variables and a two-level FOR statement.

```
*LIST
10 REM DIMENSION DEMO PROGRAM
20 DIM A(5,10)
30 FOR B=0 TO 5
40 LET A(B,0)=B
50 FOR C=0 TO 10
60 LET A(0,C)=C
70 PRINT A(B,C):
80 NEXT C:PRINT :NEXT B
90 END
```

```
*RUN
 0 1 2 3 4 5 6 7 8 9 10
 1 0 0 0 0 0 0 0 0 0 0
 2 0 0 0 0 0 0 0 0 0 0
 3 0 0 0 0 0 0 0 0 0 0
 4 0 0 0 0 0 0 0 0 0 0
 5 0 0 0 0 0 0 0 0 0 0
```

END AT LINE 90

*

FOR AND NEXT

FOR and NEXT statements define the beginning and end of a program loop. A program loop is a set of repeated instructions. Each time they are repeated they modify a variable in some way until a predetermined condition is reached, causing the program to exit from the loop. The FOR NEXT statement is of the form:

```
FOR var = nexp1 to nexp2 [STEP nexp3]
NEXT VAR
```



When BASIC encounters the FOR statement, the expressions nexp1, nexp2 and nexp3 (if present) are evaluated. The variable "var" may be a scalar numeric variable, or it may be an element of a numeric array. It is assigned a value of "nexp1." For example:

```
*FOR A=2 TO 20 STEP 2:PRINT A::NEXT A  
2 4 6 8 10 12 14 16 18 20
```

causes the program to execute as long as A is less than or equal to 20. Each time the program passes through the loop, the variable A is incremented by 2 (the STEP number). Therefore, this loop is executed a total of 10 times. When incremented to 22, program control passes to the line following the associated NEXT statement. It is important to note that the initial value used for the variable is the value assigned to the variable expression when it entered the FOR-NEXT loop. For example:

```
*A=10:FOR A=2 TO 20 STEP 2:PRINT A::NEXT A  
2 4 6 8 10 12 14 16 18 20  
*
```

Prior to execution, the variable A is assigned the value 10. The program passes through the loop 10 times. A is reset to 2 and then increments from 2 to 20.

If "nexp2" ≥ 0 , and the initial value of var \geq "nexp2," the loop terminates. For example, the program:

```
*LIST  
10 FOR J=2 TO 18 STEP 4  
20 J=18  
30 PRINT J::NEXT J  
40 END  
  
*RUN  
18  
END AT LINE 40  
*
```

is only executed once, since the value of J = 18 is reached on the first pass, satisfying the termination condition.



A loop created by the statement:

```
*FOR A=20 TO 2 STEP 2:PRINT A::NEXT A  
20  
*
```

is executed only once, as the initial value exceeds the terminal value. However, if this example is modified to read:

```
*FOR A=20 TO 2 STEP -2:PRINT A::NEXT A  
20 18 16 14 12 10 8 6 4 2  
*
```

the negative step allows normal operation.

In summary, for positive STEP values, the loop is executed until the variable (var) is greater than the final assigned value (nexp2). For negative STEP values, the loop is executed until the variable (var) is less than the final assigned value (nexp2).

If the loop does not terminate, execution is transferred to the statement following the FOR statement. Therefore, a series of statements may be executed using the incremented value of the variable. If the loop does terminate, execution is transferred to the statement following NEXT.

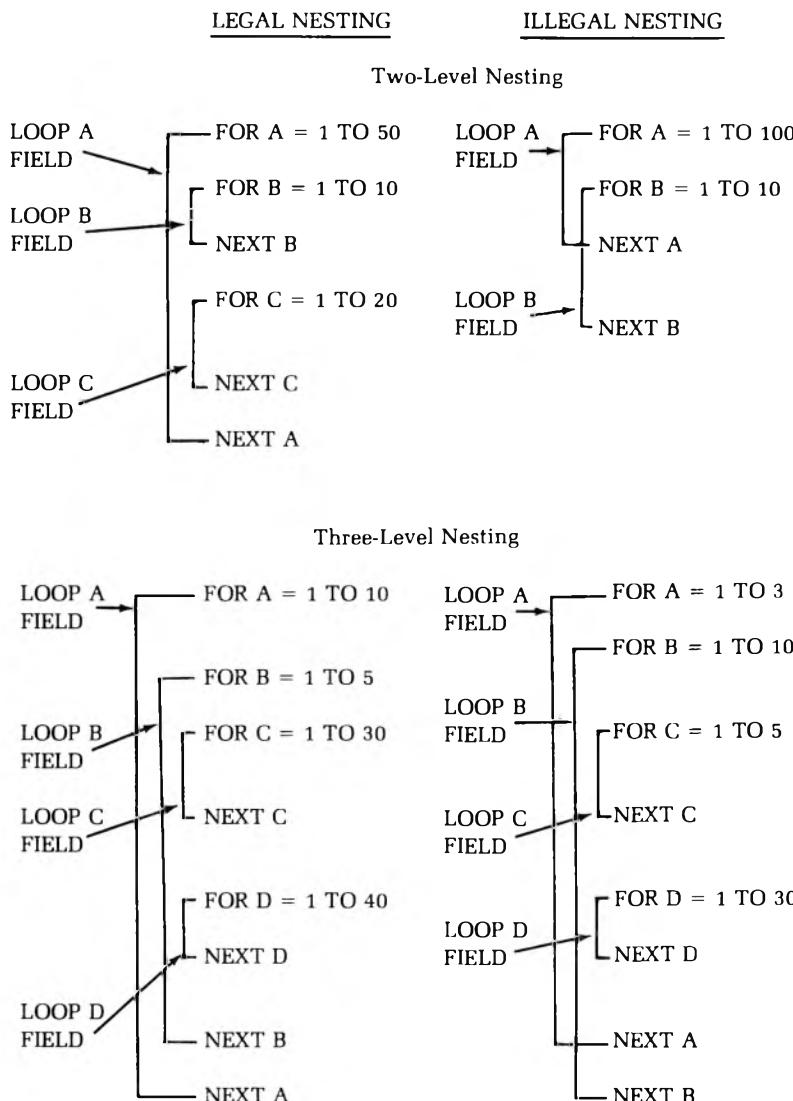
The expressions in the FOR statement can be any acceptable BASIC numeric expressions.

If the STEP expression and the word STEP are omitted from the FOR statement, a step of +1 is the default value. Since +1 is an extremely common step value, the STEP portion of the statement is frequently omitted. For example:

```
*FOR A=2 TO 10:PRINT A::NEXT A  
2 3 4 5 6 7 8 9 10  
*
```



Nesting is a technique frequently used in programming. It consists of placing one or more loops completely inside another loop. The field or operating range of the loop (the lines from the FOR statement to the corresponding NEXT statement) must not cross the field of another loop. The following two examples show legal and illegal nesting of FOR NEXT loops.



Note that both columns of nesting illustrations are shown in two-level and three-level forms. However, right-hand columns are not truly nesting but a crossover of FOR and NEXT loops (fields), and therefore are illegal. Also note that each of these examples uses the implied STEP value of 1.



The depth of nesting depends upon the amount of memory space available in Extended BASIC. BASIC limits FOR loops to 5 levels. Exceeding 5 levels generates an overflow error.

It is possible to exit from a FOR NEXT loop without reaching the variable termination value. This can be done using a conditional transfer such as an IF statement within the loop. However, control can only be transferred into a loop if the loop is left during prior program execution without being completed. This ensures the assignment of values to the termination and step variables.

Both FOR and NEXT statements can appear anywhere on a multiple statement line.

The NEXT statement does not require the variable. If the variable is not given, BASIC will NEXT the innermost FOR loop.

FREE (EXTENDED BASIC ONLY)

The FREE statement displays the amount of memory used by EX. B.H. BASIC and any program material. It also displays the total amount of free space left, which is dependant on the amount of memory in the computer and the program size. This command is particularly valuable when you are gauging the size of the program's data structure and establishing limits on a DIMENSION command. The FREE command also indicates the cause of memory overflow errors. The form of the FREE statement is:

*FREE

The form of the printout is:

TEXT = nnnn	(Bytes used by program text.)
SYMB = nnnn	(Bytes used by variables and arrays.)
FORL = nnnn	(Bytes used by FOR loops.)
GSUB = nnnn	(Bytes used by GOSUBs.)
STRN = nnnn	(Bytes used by STRING.)
WORK = nnnn	(Bytes used by expression and function evaluation.)
FREE = nnnn	(Total number of free bytes.)



For example, running the program

```
*10 GOSUB 10
```

BASIC soon returns a memory overflow error. Executing FREE shows the user a very large GOSUB table. This, and the statement provided in the error message, enables one to determine the program is in a GOSUB loop.

```
*FREE
TEXT = 9
SYMB = 0
FORL = 0
GSUB = 0
WORK = 0
STRN = 0
FREE = 7248
*10GOSUB 10
*RUN

! ERROR - MEM OVR AT LINE 10
*FREE
TEXT = 9
SYMB = 0
FORL = 0
GSUB = 7232
WORK = 0
STRN = 0
FREE = 16
*
```

GOSUB AND RETURN

A subroutine is a section of program performing some operation required one or more times during program execution. Complicated operations on a volume of data, mathematical evaluations too complex for user defined functions, or a previously written routine are all examples of processes best performed by a subroutine.

More than one subroutine is allowed in a single program. Good programming practices dictate that subroutines should be placed one after another at the end of the program in line number sequence. A useful practice is to assign distinctive line number groups to subroutines.

For example, a main program uses line numbers through 300. The 400 block is assigned to subroutine #1 and the 500 block is assigned to subroutine #2. Thus, any errors, program modifications, etc., involving the subroutine are easily identified.



Subroutines are normally placed at the end of a program, but before data statements if there are any.

Program execution begins and continues until a GOSUB statement is encountered. The form of the GOSUB statement is:

```
*GOSUB iexp
```

where iexp is the first line in the subroutine. Once GOSUB is executed, program control transfers to the first line of the subroutine and the subroutine is executed. For example:

```
60 GOSUB 500
```

in this example, control (the sequence of program execution) is transferred to line 500 in the program after line 60 is executed. The first line in the subroutine may often be a remark to identify the subroutine, or it may be any executable statement.

Once program control is transferred to a subroutine, program execution continues in the normal line-by-line manner until a RETURN statement is encountered. The RETURN statement is of the form:

```
RETURN
```

RETURN causes the program control to return to the statement **following** the original GOSUB statement. A subroutine must always be terminated by a RETURN.

Before BASIC transfers control to a subroutine, the next sequential statement to be processed after the GOSUB statement is internally recorded. The RETURN statement draws on this stored information to restart normal program execution. Using this technique, BASIC always knows where to transfer control, no matter how many times subroutines are called.

Subroutines can be nested in the same manner that FOR NEXT statements can be nested. That is, one subroutine can call another subroutine, and if necessary, that subroutine may call a third subroutine, etc. If, during execution of the subroutine a RETURN is encountered, control is returned to the line following the GOSUB calling the subroutine. Therefore, a subroutine can call another subroutine, even itself. Subroutines can be entered at any point and can have more than one RETURN. Multiple RETURN statements are often necessary when a subroutine contains conditional statements imbedded in it, which cause different subroutine completions dependent on the program data.



It is possible to transfer to the beginning or to any part of the subroutine. Multiple entry points and returns make the GOSUB statement an extremely versatile tool.

Up to 10 levels of GOSUB nesting are permitted in BASIC. *Extended BASIC permits unlimited GOSUB nesting. However, nesting uses memory and excessive nesting depth will cause an overflow.*

GOTO

The GOTO statement provides unconditional transfer of program execution to another line in the program. The GOTO statement is of the form:

```
*GOTO iexp
```

When this statement is executed, program control transfers to the line number specified by the integer expression "iexp." For example:

```
10 LET A=1
20 GOTO 40
30 LET A=2
40 PRINT A
50 END

*RUN
1

END AT LINE 50
*
```

Program lines in this example are executed in the following order:

10, 20, 40, 50

Line 30 is never executed because the GOTO statement in line 20 unconditionally transfers control to line 40. After the unconditional transfer takes place, normal sequential execution resumes at line 40.



IF THEN (IF GOTO)

The IF THEN (IF GOTO) conditionally transfers program execution from the normal consecutive order of program lines, depending on the results of a relation test. The forms of the IF statement are:

IF expression { THEN } iexp or
 { GOTO }
IF expression THEN statement

The expression frequently consists of two variables combined by the relational operators described in "Relational Operators" (Page 5-18). In the first form, if the result of the expression is true, control passes to the specified line number (iexp). In the second form, control passes to the statement following THEN on the remainder of the line. If the result of the expression is false, control passes to the next line or to a statement separated from the IF THEN statement by a colon (:). The following examples show use of the IF THEN statement.

```
10 READ A
20 B=10
30 IF A=B THEN 50
40 PRINT "A< >B",A:END
50 PRINT "A=B",A
60 DATA 10,5,20
70 END

*RUN
A=B    10

END AT LINE 70
*CONTINUE
A< >B  5

END AT LINE 40
*CONTINUE
A< >B

END AT LINE 40
*
```

NOTE: The expression can be an arbitrarily complex expression. For example:

```
IF (A<3) AND NOT (B>C) THEN
```



LET

The LET statement assigns a value to a specific variable. The form of the LET statement is:

```
LET var = nexp,          or  
LET var$ = sexp
```

The variable "var" may be a numeric variable *or a string variable "var\$."* The expression may be either an arithmetic "nexp" *or a string expression "sexp"* (*Extended BASIC*). However, all items in a statement must be either numeric or string, they cannot be mixed. If they are mixed, a type conflict error is flagged. NOTE: Unlike standard BASIC, multiple assignments are not permitted. For example,

```
LET A=B=3
```

causes A to be set to 65,535 (true) if B is equal to 3, or it causes A to be set to 0 (false) if B is not equal to 3. It does not cause both A and B to be set to 3.

You may omit the key word LET if you prefer. For example, the following two statements produce identical results.

```
10 LET A = 6  
AND  
10 A = 6
```

The LET statement is often referred to as an assignment statement. In this context, the meaning of the equal (=) symbol should be understood as it is used in BASIC. In ordinary algebra, the formula $Y = Y + 1$ is meaningless. However, in BASIC the equal sign denotes replacement rather than equality. Thus, the formula $Y = Y + 1$ is translated as add 1 to the current value of Y and store the new result at the location indicated by the variable Y.

Any values previously assigned to Y are combined with 1. An expression such as $D = B + C$ instructs BASIC to add the values assigned to the variables B and C and store the resultant value at the location indicated by the variable D. The variable D is not evaluated in terms of previously assigned values, but only in terms of B and C. Therefore, if previous assignments gave D a different value, the prior value^A is lost when this statement is executed.



LIST

This command lists the program on the console terminal for reviewing, editing, etc. The form of the list command is:

LIST [iexp]	B.H. BASIC
LIST [iexp1], [iexp2]	EX. B.H. BASIC

Line numbers are indicated by the optional integer expressions. If no line numbers are specified, the entire program is listed. *If a single line number ("iexp1") is specified, EX. B.H. BASIC lists that single line.* BASIC lists the indicated line and the balance of the program lines. You can use a CONTROL-O or CONTROL-C to abort the listing. *If both of the optional line numbers are specified, separated by a comma (,), all lines within the range of iexp1 to iexp2 are listed.* You can abort a listing by using the control characters. Refer to "Editing Commands" (Page 5-71) or to "Appendix B" (Page 5-83) for a complete explanation of these functions.

The following are examples of the LIST command.

```
10 LET A=5:LET B=6
20 PRINT A,B,A+B,
30 LET C=A/B
40 PRINT C
50 END
*RUN
      6      11      .833333
```

```
END AT LINE 50
*LIST
```

```
10 LET A=5:LET B=6
20 PRINT A,B,A+B,
30 LET C=A/B
40 PRINT C
50 END
*LIST 20
```

```
20 PRINT A,B,A+B.
*LIST 20,40
```

```
20 PRINT A,B,A+B,
30 LET C=A/B
40 PRINT C
*
```

}

EX. B.H. BASIC only



ON . . . GOSUB

The ON . . . GOSUB statement allows you to program a computed GOSUB. When you use the ON . . . GOSUB statement, use a RETURN at the end of the subroutine to return program control to the statement **following** the ON . . . GOSUB statement. The form of the ON . . . GOSUB statement is:

```
ON iexp1 GOSUB iexp2, . . . , iexpn
```

When it is processing an ON . . . GOSUB statement, BASIC evaluates the expression "iexp1" and uses the result as an index to the list of statement numbers iexp2 thru iexpn. If the expression "iexp1" evaluates to 1, for example, control is passed to the line number given by the expression "iexp2." If the expression "iexp1" evaluates to 3, for example, control is passed to line number given by the expression "iexp4." If the expression "iexp1" evaluates to 0, or to an index greater than the number of statement numbers listed, control is passed to the next program statement.

ON . . . GOTO

The ON . . . GOTO statement allows you to perform a computed GOTO. The form of the ON . . . GOTO statement is:

```
ON iexp1 GOTO iexp2, . . . , iexpn
```

When it is processing an ON . . . GOTO statement, BASIC evaluates the expression "iexp1" and uses the result as an index to the list of statement numbers iexp2 thru iexpn. If the expression "iexp1" evaluates to 1, for example, control is passed to the line number given by the expression "iexp2." If the expression "iexp1" evaluates to 3, for example, control is passed to line number given by the expression "iexp4." If the expression "iexp1" evaluates to 0, or to an index greater than the number of statement numbers listed, control is passed to the next program statement.

OUT

The OUT statement is used to output binary numbers to an output port. The form of the OUT statement is:

```
OUT iexp1, iexp2
```



The expression "iexp1" is used as the port address, and "iexp2" is the value to be placed at that port. Both iexp1 and iexp2 are decimal numbers. The low-order 8-bits generated by the decimal numbers in iexp1 or iexp2 are used. If you wish to write iexp1 and iexp2 in octal notation for ease in conversion to the actual binary values, write a subroutine or function to perform octal to decimal conversion.

PAUSE

The PAUSE statement causes BASIC to delay before executing the next statement. There are two forms of PAUSE. In BASIC the form of the PAUSE statement is:

```
PAUSE
```

Once the PAUSE statement is executed, no further statements are executed until you type a console terminal character. You can terminate PAUSE by typing any key, and this will not cause the character, to be echoed, but it is good practice to consistently use one character such as space to terminate PAUSE. *In Extended BASIC the form of the PAUSE statement is:*

```
PAUSE [iexp]
```

You can also terminate PAUSE by specifying an optional time duration with iexp. If iexp is specified, PAUSE waits 2 times iexp milliseconds. After 2 times iexp milliseconds pass, normal program execution continues.

The PAUSE statement is particularly useful when you are viewing long outputs on a CRT display. You can insert a PAUSE at appropriate points in the program, allowing you to view the information on the CRT before continuing execution.

POKE

The POKE statement is used to write values into an assigned H8 memory location. The form of the POKE statement is:

```
POKE iexp1, iexp2
```

The low-order 8-bits of iexp2 are inserted into memory location iexp1. NOTE: iexp1 and iexp2 must be given as decimal numbers. If you wish to use octal numbers for ease in referencing to binary notation, you must use a separate octal to decimal subroutine or function to generate these numbers.



CAUTION: You can damage BASIC when using the POKE statement, causing a failure which could result in loss of program material and/or require reloading BASIC itself. The POKE statement should be confined to areas of memory, not used by the BASIC interpreter.

PORt

The PORT statement is used to direct the result of a print statement to an I/O port other than the console terminal port (372_8 or 250_{10}). You can also use it to direct the console terminal operations to another port. One of the primary uses of the PORT statement is directing data printing to a printer. The form of the PORT statement is:

```
PORT iexp
```

The expression iexp is the desired port number (in decimal). If iexp is positive, the print function is transferred to the indicated port number while the statement (in the command mode) or the program (in the program mode) is being executed. After execution is complete, the printing function returns to the console terminal. If iexp is negative, the keyboard functions of the console terminal are transferred to the desired port, along with the printing functions.

The console terminal is then permanently transferred to the selected port and an additional PORT instruction must be issued at this new console terminal to return operation to port 250. Recovery from accidental assignment to a nonexistent port is accomplished by an RST0 followed by a warm start (PC = 040 103).

PRINT

The PRINT statement is used to output **data** to the console terminal. The form of the PRINT statement is:

```
PRINT [nexp1 sep1 . . . nexpn(sepn)]
```

The expressions and separators contained within the brackets are optional. When used without these optional expressions and separators, the simple PRINT statement outputs a blank line followed by a carriage-return line feed.

Printing Variables

The PRINT statement can be used to evaluate expressions and to simultaneously print their results, or to simply print the results of a previously evaluated



expression or evaluations. Any expression contained in the PRINT statement is evaluated before the result is printed. For example:

```
10 A=4:B=6:C=5+A  
20 PRINT  
30 PRINT A+B+C  
40 END  
*RUN
```

19

```
END AT LINE 40  
*
```

All numbers are printed with a preceding and following blank. You can use PRINT statements anywhere in a multiple statement line. NOTE: The terminal performs a carriage-return line feed at the end of each PRINT statement unless you use the separators described in "Use of the , and ;" (Page 5-52). Thus, in the previous example, the first PRINT statement outputs a carriage-return line feed and the second print statement outputs the number 19 followed by a carriage-return line feed.

Printing Strings

The PRINT statement can be used to print a message (a string of characters). The string may be alone or it may be used together with the evaluation and printing of a numeric value. Characters to be printed are designated by enclosing them in quotation marks (""). For example:

```
10 PRINT "THIS IS A HEATH H8"  
*RUN  
THIS IS A HEATH H8  
  
END AT LINE 65535  
*
```

The string contained in a PRINT statement may be used to document the variable being printed. For example:

```
10 LET A=5:LET B=10  
20 PRINT "A + B",A+B  
30 END  
*RUN  
A + B      15  
  
END AT LINE 30  
*
```



When a character string is printed, only the characters between the quotes appear. No leading or trailing blanks are added as they are when a numeric value is printed. Leading and trailing blanks can be added within the quotation marks.

Use of the , And ;

The console terminal is normally initialized with 80 columns divided into five zones. (See CNTRL 3, n for exception.) Each zone, therefore, consists of 14 spaces. When an expression in the PRINT statement is followed by a comma (,) the next value to be printed appears in the next available print zone. For example:

```

10 A=5.55555:B=2
20 PRINT A,B,A+B,A*B,A-B,B-A
30 END
*RUN
5.55554      2          7.55554      11.1111      3.55554
-3.55554

END AT LINE 30
*
```

NOTE: The sixth element in the PRINT list is the first entry on a new line, as the five print zones of a 72-character line were used.

Using two commas together in a PRINT statement causes a print zone to be skipped. For example:

```

10 A=5.55555:B=2
20 PRINT @,B,A+B,,A*B,A-B,B-A
30 END
*RUN
5.55554      2          7.55554      11.1111
2.55554      -3.55554

END AT LINE 30
*
```



If the last expression in a PRINT statement is followed by a comma, no carriage-return line feed is given when the last variable is printed. The next value printed (by a later PRINT statement) appears in the next available print zone. For example:

```
10 LET A=1:LET B=2:LET C=3
20 PRINT A,
30 PRINT B
40 PRINT C
50 END
*RUN
1           2
3

END AT LINE 50
*
```

At certain times it is desirable to use more than the designated five print zones. If such tighter packing of the numeric values is desired, a semi-colon (;) is inserted in place of the comma. A semi-colon does not move the next output to the next PRINT zone, but simply prints the next variable, including its leading and trailing blank. For example:

```
10 LET A=1:LET B=2:LET C=3
20 PRINT A;B;C
30 PRINT A+1;B+1
40 PRINT C+1
50 END
*RUN
1   2   3
2   3
4

END AT LINE 50
*
```

NOTE: If either a comma or a semicolon is the final character in a PRINT statement, no final carriage-return line feed is printed.



READ AND DATA

The READ and DATA statements are used in conjunction with each other to enter data into an executing program. One statement is never used without the other. The form of the statements are:

```
READ var1, . . . , varn
DATA exp1, . . . , expn
```

The READ statement assigns the values listed in the DATA statement to the specified variables var1 through varn. The items in the variable list may be simple variable names, arrays, or *string variable names*. Each one is separated by a comma. For example:

```
5 DIM A (2,3)
10 READ C,B$,A (1,2)
20 DATA 12,"THIS IS SIX",56
30 PRINT C,B$,A (1,2)
*RUN
12      THIS IS SIX  56

END AT LINE 65535
*
```

Because data must be read before it can be used in the program, READ statements generally occur in the beginning of a program. You may, however, place a READ statement anywhere in a multiple statement line. The type of expression in the DATA statement must match the type of corresponding variable in the READ statement. When the DATA statement is exhausted, BASIC finds the next sequential DATA statement in the program. NOTE: BASIC does not automatically go to the next DATA statement for every READ statement. Therefore, one DATA statement may supply values for several READ statements if DATA statement contains more expressions than the READ statement has variables.

DATA statements may contain arbitrarily complex expressions to represent the data values. Each value expression is separated from other value expressions by a comma. A field in the DATA statement may be left null by means of two adjacent commas. This causes the associated variable to retain its old value. For example:

```
10 A=1:B=1:C=1
20 READ A,B,C
30 PRINT A,B,C
40 DATA 3,,4
50 END
*RUN
3      1      4

END AT LINE 50
*
```



If a DATA statement appears on a line, it must be the only statement on the line. DATA statements may not follow any other statement on the line. Other statements should not follow DATA statements.

DATA statements do not have to be executed to be used. That is, they may be the last statement in a program, and be used by a READ statement executed earlier in the program. However, if DATA statements appear in a program in such a place that they are executed (there are executable statements beyond the DATA statement), the executed DATA statement has no effect. Therefore, location of DATA statements is arbitrary as long as the expressions contained within the DATA statements appear in the correct order. However, good programming practice dictates all DATA statements occur near the end of the program. This makes it easy for the programmer to modify the DATA statements when necessary.

If an expression contained in a DATA statement is bad, the illegal character error message is printed. All subsequent READ statements also cause the message. If there is no data available in the data table for the READ statement to use, the no data error message is printed.

If the number of expressions in the data list exceed those required by the program READ statements, they are ignored, and thus not used.

REM (REMARK)

The REMARK statement lets you insert notes, messages, and other useful information within your program in such a form that it is not executed. The contents of the REMARK statement may give such information as the name and purpose of the program, how the program may be used, how certain portions of the program work, etc.. Although the REMARK statement inserts comments into the program without affecting execution, they do use memory which may be needed in exceptionally long programs.

REMARK statements must be preceded by a line number when used in the program. They may be used anywhere in a multiple statement line. The message itself can contain any printing character on the keyboard and can include blanks. BASIC ignores anything on a line following the letters REM.



RESTORE

The RESTORE statement causes the program to reuse data starting at the first DATA statement. It resets the DATA statement pointer to the beginning of the program. The RESTORE statement is of the form:

```
RESTORE
```

For example:

```
10 READ A,B,C
20 PRINT A,B,C
30 RESTORE
40 READ D,E,F
50 PRINT D,E,F
60 DATA 1,2,3,4,5,6,7,8
70 END
*RUN
1          2          3
1          2          3
END AT LINE 70
*
```

This program does not utilize the last five elements of the DATA statement. The RESTORE command resets the DATA statement pointer and the READ D,E,F, statement uses the first three data elements, as does the initial READ statement.

The CLEAR command includes the RESTORE function.



STEP

The STEP command permits you to step through a program a single line or a few lines at a time. The form of the step command is:

```
STEP iexp
```

where the integer expression iexp indicates the number of lines to be executed before stopping. Execution of the desired lines is indicated by the prompt NXT = nnnn, where nnnn is the next line number to be executed. A STEP 2 is required to execute the first program line. All future single-line executions require a STEP or STEP 1. For example:

```
10 READ A,B,C  
20 PRINT A,B,C  
30 RESTORE  
40 READ D,E,F  
50 PRINT D,E,F  
60 DATA 1,2,3,4,5,6,7,8  
70 END
```

```
*CLEAR
```

```
*STEP 3  
1          2          3  
NXT= 30  
*STEP  
NXT= 40  
*STEP  
NXT= 50  
*STEP  
1          2          3  
NXT= 60  
*STEP 2
```

```
END AT LINE 70
```

```
*
```



Program Mode Statements

PROGRAM MODE statements are valid only when utilized within a program. If they are entered in the command mode, an illegal use error is flagged.

DATA

The DATA statement discussed in “Read and Data” (Page 5-54) is a program only statement, although it is used in conjunction with a READ statement, which may be used in either the command or program modes.

DEF FN

The DEF FN statement defines single line program functions created by the user. The form of the DEF FN statement is:

```
DEF FN varname (arg1 [,arg2, . . . ,argn] ) = expr
```

The variable name (varname) must be a legal string or numeric variable name and cannot be previously dimensioned. However, it may be previously defined. The latest definition takes precedence. The argument list “(arg1 [arg2, . . . ,arg3])” must be supplied to indicate a function. NOTE: The arguments are real, not dummy variables, and do change as evaluation procedes.

```
10 REM DEFINE A SQUARE FUNCTION
20 DEF FN S1(I) = I * I
30 PRINT FN S1(3),I,FN S1(5),I
40 END

*RUN
      9          3          25          5

END AT LINE 40
*
```

END

The END statement causes control to return to the command mode. An END statement message is typed, giving the line number of the END statement. END also causes the next statement pointer to be set to the beginning of the program so a CONTINUE resumes execution at the beginning of the program.



An END statement may appear anywhere in the program, as many times as desired. If a program does not contain an END statement, it "runs off the end." In this case, BASIC generates a pseudo end statement at line 65,535.

INPUT AND LINE INPUT

The INPUT statement is used when data is to be READ from the terminal keyboard or from a mass storage device **working through the console terminal**. The form of the INPUT statement is:

```
INPUT prompt;var1, . . . , varn
```

If the first element in the list following the INPUT statement is a string, INPUT assumes it is a PROMPT and types the string in place of a question mark (?). *If no prompt string is desired but the first variable is a string variable, a leading semi-colon is inserted. For example:*

```
INPUT ;S3$(2)
```

This statement tells BASIC that the data to come from the console terminal is to be placed in a dimensioned string named S3.

Data input from the console terminal has a format identical to the DATA statement.

NOTE: Responses to string inputs must be enclosed in quotes.

Expressions may be supplied and null fields cause the variable to retain its previous value. If the user response does not supply sufficient data to complete the INPUT statement, another "?" prompt is issued, requesting more data input at the terminal. **CAUTION:** If you supply too much data, it will be ignored. The next INPUT statement issues a fresh READ to the terminal.

The response to the LINE INPUT statement cannot be continued on another line, as they are terminated by the return key.

When there are several values to be entered via the input statement, it is helpful to print a message explaining the data needed, using the prompt string. For example:

```
10 INPUT "THE TIME IS";T
```

When this line of the program is executed, BASIC prints

```
THE TIME IS
```

and then waits for a response.



The LINE INPUT statement is used to input one line of string data from the console terminal and assign it to a string variable. Its form is identical to the INPUT form, but the string should not be enclosed in quotes.

STOP

The STOP statement causes BASIC to enter the command mode. The message stating the line number of the STOP is printed. The next line pointer is left after the STOP statement, so a CONTINUE statement causes execution to resume on the line immediately after the STOP statement. The STOP statement is of the form:

```
STOP
```

The STOP statement can occur several times throughout a single program with conditional jumps determining the actual end of the program. The following example uses the STOP statement to examine a variable during execution.

```
10 A=1:B=2:C=3
20 PRINT A,B,C
30 END

*RUN
1           2           3

END AT LINE 30
*15STOP

*RUN

STOP AT LINE 15
*PRINT A
1
*15           Stop deleted

*RUN
1           2           3

END AT LINE 30
*
```



PREDEFINED FUNCTIONS

Introduction

There are 26 predefined functions in EX. B.H. BASIC and 16 predefined functions in B.H. BASIC. These functions perform standard mathematical operations such as square roots, logarithms, string manipulation, and special features. Each function has an abbreviated three- or four-letter name, followed by an argument in parentheses. As these functions are predefined, they may be used throughout a program when required. Predefined functions use numeric expressions (nexp), integer expressions (iexp), and in EX. B.H. BASIC, string expressions (sexp). Function key words are automatically followed by an open parenthesis “(”.

The abbreviation (narg) is used to indicate a numeric argument, a decimal number lying in the approximate range of 10^{-38} to 10^{+37} . Certain functions do not permit the argument to assume this wide range, as indicated in the function description.

The predefined functions may be used in either the command or program mode.

Arithmetic and Special Feature Functions

The Arithmetic and Special Feature Functions supported by BASIC are identical for EX. B.H. BASIC and B.H. BASIC with the exception of the functions Maximum, Minimum, Space, Tab, and Tangent. These are discussed later in this section. (See Page 5-67.)

THE ABSOLUTE VALUE FUNCTION, ABS (nexp)

The ABSOLUTE VALUE Function gives the absolute value of the argument. The absolute value is the positive portion of the numeric expression. For example:

```
*PRINT ABS(-5.5)
      5.5          or,
*PRINT ABS(SIN(3.5))
      .350783
      *
```

NOTE: The sine of 3.5 radians is -.350783.



THE ARC TANGENT FUNCTION, ATN (nexp)

The ARC TANGENT Function returns the arc tangent of the argument. For example:

```
*PRINT ATN(1/1)*57.296;"DEGREES"
45.0001 DEGREES
*PRINT 4*ATN(1)
3.14159
*
```

NOTE: $\pi = 3.14159$

THE COSINE FUNCTION, COS (nexp)

The COSINE function returns the COSINE of the argument (nexp) expressed in radians. For example:

```
*PRINT COS(60/57.296)
.500003
*
```

THE EXPONENTIAL FUNCTION EXP (NEXP)

The EXPONENTIAL function returns the value e^{nexp} . If “nexp” exceeds 88, an overflow error is flagged, as the result exceeds 10^{38} . If “nexp” is less than -88, an overflow error occurs. An example of the exponential function is:

```
*PRINT EXP(1),EXP(2),EXP(COS(60/57.296))
2.71828          7.38905          1.64873
*
```

THE INTEGER FUNCTION, INT (narg)

The INTEGER function returns the value of the greatest integer value, not greater than “narg.” If the argument is a negative number, the INTEGER function returns the negative number with the same or smaller absolute value. For example:

```
*PRINT INT (38.55)
38

*PRINT INT (-3.3)
-3
```

THE LOGARITHM FUNCTION, LOG (nexp)

The LOGARITHM function returns the natural logarithm (LOG to the base e) of the argument. You can find the Logarithms of a number N in any other base by using the formula:



$$\log_a N = \log_e N / \log_e a$$

where "a" represents the desired base. Most frequently, "a" is 10 when you are converting to common logarithms. For example:

```
*PRINT "A POWER RATIO OF 2 IS";10*(LOG(2)/LOG(10));"DECIBELS"  
A POWER RATIO OF 2 IS 3.0103 DECIBELS  
*
```

THE PAD FUNCTION, PAD (0)

The PAD function returns the value of the keypad pressed on the H8 front panel. For example:

```
*PRINT PAD(0)  
6
```

The #6 key was pressed.

The PAD function uses all the front panel debounce and repeat software contained in PAM-8. (See "The Segment Function," Page 5-65, for an additional example.)

NOTE: The PAD function must be completely executed before any other function will respond. Therefore, CONTROL-C, etc., will not work until you press an H8 front panel key.

THE PEEK FUNCTION, PEEK (iexp)

The PEEK function returns the numeric value of the byte at memory location iexp.

THE PIN FUNCTION, PIN (iexp)

The PIN function returns the value input from port "iexp" where iexp is a decimal expression ranging from 0-255. For example:

```
*A=PIN(38)
```

Where "A" now contains the data that was at port #38 (46 octal).

THE POSITION FUNCTION, POS (0)

The POSITION function returns the current terminal printhead (cursor) position. Although the numeric argument (0) is ignored, it must be present to complete the function. The value returned is a decimal number indicating the column number of the printhead (cursor) position. For example:

```
*PRINT POS(0), POS(0), POS(0); POS(0); POS(0)  
1 15 29 33 37  
*
```

THE RANDOM FUNCTION, RND (narg)

The RANDOM number function returns the next element in a pseudo random series. The RANDOM number generator is not truly random, and may be manipulated by controlling the argument. If narg>0, the random number generator



returns the next random number in the series. If narg = 0, the random number generator returns the previously returned random number. If narg<0, the value "narg" is used as a new seed for a random number, thus starting an entire new series. Using these three inputs to the random number series, the programmer may continuously return the same number while de-bugging the program, determine what the series of numbers will be when the program is run, or start a series of new random numbers each time BASIC is loaded. For example:

```
10 FOR A=0 TO 2
20 PRINT RND(1)
30 NEXT
40 END
```

```
*RUN
.93677
.566681
.53128
```

```
END AT LINE 40
*RUN
.564484
.787262
.332306
```

```
END AT LINE 40
*20PRINT RND(0)
```

```
*RUN
.332306
.332306
.332306
```

```
END AT LINE 40
*20PRINT RND(-1)
```

```
*RUN
6.25305E-02
6.25305E-02
6.25305E-02
```

```
END AT LINE 40
*20PRINT RND(-5)
```

```
*RUN
.460968
.460968
.460968
```

```
END AT LINE 40
*
```



THE SEGMENT FUNCTION, SEG (narg)

The SEG function returns a numeric value which is the correct 8-bit binary number to display the digit on the H8 front panel LED's. The argument must be an integer between 0 and 9. The following program demonstrates the use of PAD, POKE, and SEG in EX. B.H. BASIC. See the second example for a B.H. BASIC program.

```
10 REM A PROGRAM TO USE THE FRONT PANEL LEDS. CNTRL 2,1 TURNS
20 REM ON THE LEDS WITHOUT UPDATE. THE KEYPAD NOW DRIVES THE
30 REM DISPLAY THRU BASIC. 8203 IS THE FIRST LED MEM LOCATION.
40 CNTRL 2,1
50 A=8203
60 FOR I=A TO A+8
70 POKE I,SEG(PAD(0))
80 NEXT I
90 GOTO 60
*RUN
```

When this program is executed, the H8 front panel LEDs respond to the H8 keypad numeric entries. To escape from the program you would type CONTROL-C and then press a key on the H8 front panel. NOTE: If BASIC is to be used, the EX. B.H. BASIC command CNTRL cannot be used to turn on the displays and turn off the update. Therefore, the program is modified to use a POKE command at the first line which stops display update. The program escape routine is still the same. When you are using BASIC, the program is:

```
40 POKE 8200,2
50 A=8203
60 FOR I=A TO A+8
70 POKE I,SEG (PAD(0))
80 NEXT I
90 GOTO 60
*RUN
*
```



THE SIGN FUNCTION, SGN (narg)

The SGN function returns the value +1 if “narg” is a positive value, 0 if “narg” is 0, and -1 if “narg” is negative. For example:

```
*PRINT SGN(5.6)
1

*PRINT SGN(-500)
-1

*PRINT SGN(12-12)
0

*
```

THE SINE FUNCTION SIN (nexp)

The SIN function returns the sine of the argument (nexp) expressed in radians. For example:

```
*PRINT SIN(30/57.296)
.499999
*
```

SQUARE ROOT FUNCTION, SQR (narg)

The SQUARE ROOT function returns the square root of “narg.” The argument “narg” must be greater than or equal to 0 (for example, positive).

```
*FOR A=0 TO 5:PRINT A,SQR(A),A*A:NEXT
0      0      0
1      1      1
2      1.41421 4
3      1.73205 9
4      2      16
5      2.23607 25

*
```

USER DEFINED FUNCTION, USR (narg)

This function calls a user-supplied machine language function. The user-supplied function should be stored in high memory above BASICs user set high memory limit. You should place the starting address of your machine language function at location USRFCN. (See “Appendix C.”) A RET instruction exits from the user-defined function. An example of the USR function is given in “Appendix E,” Page 5-111.



THE FREE SPACE FUNCTION, FRE (0) [B.H. BASIC ONLY]

The FREE function returns the number of free bytes of program and variable storage area available. You must supply the dummy argument 0 to complete the function. This function is not available in EX. B.H. BASIC. Most frequently, this function is used in the command mode in conjunction with a PRINT statement. For example:

```
*PRINT FRE (0)
1224
*
```

THE MAXIMUM FUNCTION, MAX (nexp1, . . . ,nexpn)

The MAXIMUM function returns the maximum value of all the expressions which are arguments of the function. For example:

```
10 LET A=1
20 PRINT MAX(COS(A),SIN(A)/COS(A))
30 END
*RUN
1.55741
END AT LINE 30
*
```

The expression containing the maximum value is the expression for the tangent of 1 radian, (1.55741).

THE MINIMUM FUNCTION, MIN (nexp1, . . . ,nexpn)

The MINIMUM function returns the lowest value of all the expressions contained in the argument. For example:

```
*PRINT MIN(1,2,3,4,.5)
.5
*
```

THE TANGENT FUNCTION, TAN (nexp)

The TANGENT Function returns the TANGENT of the argument "nexp" expressed in radians. For example:

```
*PRINT TAN (45/57.296)
.999996
*
```



THE SPACE FUNCTION, SPC (iexp)

The SPACE function spaces the printhead (cursor) iexp spaces to the right of its present position. For example:

```
*PRINT 12,14,SPC(20);600
      12           14           600
      *
      *
```

THE TAB FUNCTION TAB (iexp)

The TAB function moves the printhead (cursor) to the iexp th column. NOTE: If the printhead is at or past the iexp th column, the function is ignored. For example:

```
*PRINT TAB(20);60,70
      *           60           70
      *
```

String Functions (Extended BASIC Only)

Extended BASIC contains various functions for processing character strings in addition to the functions used for mathematical operations. These functions allow the program to concatenate two strings, access a part of string, generate a character string corresponding to a given number, or generate a number for a given string.

THE CHARACTER FUNCTION, CHR\$ (iexp)

The CHARACTER function returns a string that consists of a single character. The character generated has the ASCII code "iexp." NOTE: "iexp" is a decimal number and must be converted to octal for comparison with most ASCII character tables. See "Appendix D" on Page 0-52 of the "Introduction." For example:

```
*PRINT CHR#(65)
      A
*PRINT CHR$(70)
      F
      *
```

NOTE: If iexp = 0, the generated string is null.



THE STRING FUNCTIONS, STR\$ (narg)

The STRING function encodes the argument (narg) into ASCII in the same format used by the PRINT statement for numbers. These characters are returned as a string, with leading and trailing blanks. For example:

```
*PRINT STR$(100)      } STR$ function
100
*PRINT "100"          } Normal string printing
100
*
```

THE ASCII FUNCTION, ASC (sexp)

The ASCII function returns the ASCII code for the first character in the string expression (sexp). If the string is a null, the ASCII function returns a zero. The return is a decimal number and must be converted to octal for comparison to most ASCII tables. See "Appendix D" on Page 0-52 of the "Introduction." For example:

```
*PRINT ASC("ABC")
65
*PRINT CHR$(65)
A
*
```

THE LEFT STRING FUNCTION, LEFT\$ (sexp, iexp)

The LEFT STRING function returns the "iexp" left-most characters of the string expression (sexp). If "iexp" equals 0, the null string is returned. For example:

```
*PRINT LEFT$("HELLO, THIS IS A TEST",10)
HELLO, THI
*
```

THE RIGHT STRING FUNCTION, RIGHT\$ (sexp, iexp)

The RIGHT STRING function returns the right-most "iexp" characters of the string expression (sexp). If "iexp" equals 0, the null string is returned. For example:

```
*PRINT RIGHT$("HELLO, THIS IS A TEST",10)
IS A TEST
*
```



THE MIDDLE STRING FUNCTION, MID\$ (sexp, iexp [, iexp2])

The MIDDLE STRING function returns the right-hand substring of the string expression "sexp" starting with the "iexp1" th character from the left-hand side (the first character is 1). The return continues for "iexp2" characters or to the end of the string if the optional terminating expression "iexp2" is omitted. For example:

```
*PRINT MID$("HELLO, THIS IS A TEST",10,10)
IS IS A TE
*
```

THE NUMERIC VALUE FUNCTION, VAL (sexp)

The NUMERIC VALUE function returns the numeric value of the number encoded in the string expression (sexp). For example:

```
*PRINT VAL (" .003E-1")
3.00000E-04
*
```

THE LEN FUNCTION, LEN (sexp)

The LEN function returns the length of the string expression "SEXP." For example:

```
*PRINT LEN("HOW LONG IS THE STRING?")
23
```



EDITING COMMANDS

BENTON HARBOR BASIC provides several commands used to halt program execution, erase characters, delete lines, add lines, and provide other editing functions. A great number of these editing commands are common to all the Heath H8 Software packages. Their operation is covered in detail in the "Console Driver" section (Page 0-36) of the "Introduction" to this Software Reference Manual.

Control-C, CNTRL-C

CONTROL-C is a general-purpose cancel key. It can be used to stop a mass storage input or output operation, stop program execution, stop a listing, and to stop a program during an input statement. Using CONTROL-C results in the CC error message (CNTRL-C in EXTENDED BASIC). NOTE: A CONTROL-C causes the program to terminate at the end of a current statement.

You can continue program execution by using the CONTINUE statement.

Inputting Control

The following control characters take effect when you are inputting information from the control terminal.

BACKSPACE, BKSP/CNTRL-H

The BACKSPACE key (or a CONTROL-H) causes a one-character backspace. The backspace code is echoed to the terminal so devices with backspace capability physically backspace. Attempting to backspace into column zero is illegal and causes a terminal bell code to be echoed. NOTE: Backspace can be changed at configuration. See "Product Installation" on Page 0-19 of the "Introduction" to this Software Reference Manual.

RUBOUT

The RUBOUT key causes BASIC to discard the current line being inputted. A carriage-return feed line is sent to the console terminal, and the user may now re-enter the entire line. NOTE: Rubout can be changed at configuration. See "Product Installation" on Page 0-19 of the "Introduction" to this Software Reference Manual.



Outputting Control

The following control characters take effect only when you are outputting information to the console terminal. They should not be used when you are inputting information via the console terminal, as they affect characters being echoed to the console.

OUTPUT SUSPENSION AND RESTORATION, CNTRL-S AND CNTRL-Q

Type CONTROL-S to suspend the output and to suspend program execution. This command is particularly useful when you are using a video terminal; you can use the CONTROL-S (suspend) feature each time a screen is nearly filled and information at the top will be lost due to scrolling.

By typing CONTROL-Q, you permit BASIC to continue execution and outputting information to the terminal. CONTROL-Q cancels the CONTROL-S function.

The DISCARD FLAG, CNTRL-O and CNTRL-P

Type a CONTROL-O to toggle the DISCARD FLAG. Setting the DISCARD FLAG stops output on the terminal but does not halt program execution until you retype CONTROL-O or until you type a CONTROL-P to clear the DISCARD FLAG. CONTROL-O is often used to discard the remainder of long listings and other similar outputs. BASIC clears the discard flag when it returns to the command mode or when an INPUT statement is executed, so that the prompt will appear.

Command Completion

When you are inputting information from the console terminal in the command mode, or in response to an INPUT command, BASIC checks the incoming characters for the initial characters of a keyword. As soon as enough characters of a keyword are entered to uniquely identify it, and it is distinguished from a variable name, BASIC completes the keyword into the terminal. For example, to enter the command SCRATCH, type SC. Since SC uniquely determines SCRATCH and SC is not a legal variable name, BASIC types the characters RATCH immediately following the SC. Striking the backspace key backspaces over the entire word SCRATCH.

Function keywords are automatically followed by an open parenthesis “(”. Other keywords are immediately followed by a blank.



Enforced lexical Rules

BASIC enforces two lexical rules during input.

1. Two adjacent alphabetical characters must start a keyword. For example, XX is illegal as no keyword starts with "XX" and "XX" is an illegal variable name. This rule is excepted when following a REMARK statement or when the characters are contained within quotes (indicating a string).
2. A quoted string must be closed; every quote character must have a mate on the same line.

Should a character be typed which is in violation of these rules, a bell code is echoed and the character is ignored.

General Text Rules

BLANKS

BASIC programs are generally "free format." That is, blanks (spaces) may be included freely with the following restrictions.

1. Variable names, keywords, and numeric constants may not contain imbedded blanks.
2. Blanks may not appear before a statement number.

LINE INSERTION

You can insert lines into a BASIC program by simply typing an appropriate line number followed by the desired line of text. This is done in response to the command mode prompt (an asterisk). Except when running a program, BASIC remains in the command mode, showing a single asterisk (*) as a prompt. NOTE: The text should immediately follow the last digit of the line number. Although intervening blanks are allowable, they waste memory. BASIC automatically inserts a blank when listing the text. For example:

```
*100PRINT "HEATH BASIC"  
LIST  
100 PRINT "HEATH BASIC"
```



LINE LENGTH

A line in BENTON HARBOR BASIC is restricted to 80 characters, and lines in Extended BENTON HARBOR BASIC are restricted to 100 characters. This restriction on line length is completely independent of the console width, which was established by the software configuration procedure (see Page 0-19 of the "Introduction" to this Software Reference Manual). NOTE: If the console terminal, for example, was set at a width of 34 characters, the console will display three complete lines for one line of BASIC.

LINE REPLACEMENT

Replace existing program lines by simply typing the line number and the new text. This is the same process you use to insert a new line. The old line is completely lost once the new line is entered.

LINE DELETION

Delete lines by typing the line number immediately followed by a carriage-return. You can leave blank lines by typing the single space before typing the carriage-return.



ERRORS

BASIC detects many different error conditions. When an error is detected, a message of the form:

```
! ERROR-(ERROR MESSAGE) [at line NNNNN]
```

is typed. BASIC returns to the command mode (if it is not already in the command mode), ringing the console terminal bell. If BASIC is in the command mode, the "at line NNNN" portion of the error message is omitted. For example:

```
*PRINT 1/0  
! ERROR - /0  
*10PRINT 1/0  
*RUN  
  
! ERROR - /0 AT LINE 10  
*
```

NOTE: If a line of BASIC contains an error, you can correct it by retyping the entire line. Once the line number is typed, the contents of the old line are lost. To delete a line, type the line number and follow it with a carriage-return.

Error Messages

The following "Basic Error Table" describes the error messages generated by BENTON HARBOR BASIC and Extended BENTON HARBOR BASIC. Two columns of error messages are given, as BENTON HARBOR BASIC provides a shortened form of the error messages. An explanation is given after each error message in the Comments column.

Recovering from Errors

When an error is detected, BASIC enters the command mode with the variables and stacks as they were at the time of the error. Thus, the user can use PRINT and LET statements to examine and alter variable contents. Likewise, a GOTO statement can be used to set the next statement pointer to any desired statement number. Often, a combination of these techniques allows the user to continue a program with the error corrected.

NOTE: If program text in an EX. B.H. BASIC program is modified in any way, the GOSUB and FOR stacks are purged. If an error occurred in a GOSUB routine, or a FOR LOOP, the entire program must be restarted. If you modify text in a B.H. BASIC program, B.H. BASIC CLEARs all variables.





BASIC ERROR TABLE

BASIC	EXTENDED BASIC	COMMENTS
AC	<i>ARG CT</i>	Argument count. Incorrect number of arguments supplied to a DEF defined function.
CC	<i>CNTRL-C</i>	CONTROL-C. Program execution or other operation aborted by a CONTROL-C typed at the console terminal.
DE	<i>NO DAT</i>	Data exhausted. A READ called for more data than is available in the DATA statement.
/0	/0	Divide by zero. An attempt to divide by zero. NOTE: Either dividing by the number zero or dividing by an expression which evaluates to zero generates this error.
IN	<i>NUM</i>	Illegal number. The line number referenced by a command or program statement is not used by BASIC.
IU	<i>USE</i>	Illegal usage. A command or statement is used in the improper context.
NX	<i>NXT</i>	Next variable missing. No FOR statement matching the accompanying NEXT statement.
OV	<i>OVRLF</i>	Overflow. Memory space is filled by program text.
RE	<i>RTN</i>	Return error. A RETURN is encountered without a calling statement.
S#	<i>STAT#</i>	Statement number. The referenced statement number does not exist in the program.



BASIC	EXTENDED BASIC	COMMENTS
SY	<i>SYN</i>	Syntax error. Command, statement, or function uses incorrect separators, functions, etc..
MO	<i>MEM OV</i>	Table overflow. One of the internal tables has grown too large for memory. Check GOSUBs, FOR loops, and DIM.
SR	<i>SUBS RANG</i>	Subscript range. The subscript size of a dimensioned variable exceeded the size defined by the DIM statement.
SC	<i>SUBS CNT</i>	Subscript count. The number of subscripts assigned to a variable exceeds the number defined in the DIM statement.
ND	<i>DIM?</i>	Not dimensioned. The subscripted variable has not been dimensioned in a DIM statement.
IC	<i>ILL CHA</i>	Illegal character. An improper character is assigned to a command or function NOTE: In B.H. BASIC, an attempt to assign a string to a numeric variable results in an illegal character message.
FU	<i>FN ?</i>	Function error. No single line function defined by a DEF statement was found when the FN function was encountered. NOTE: The DEF FN must be executed prior to executing the FN function.
TE	<i>TAPE</i>	Tape error. An error in handling the mass storage device at the load dump port.



BASIC	EXTENDED BASIC	COMMENTS
-------	-------------------	----------

	<i>CNTRL-B</i>	<i>CONTROL-B error. A CONTROL-B entered from the console terminal, but no CONTROL-B processing in the program.</i>
--	----------------	--

	<i>STR LE</i>	<i>String length error. The length of a string exceeded 255 characters.</i>
--	---------------	---

	<i>TYP CNFLC</i>	<i>Type conflict. String data supplied for a numeric variable or numeric data supplied for a string variable.</i>
--	------------------	---





APPENDIX A

Loading Procedures

Loading From the Software Distribution Tape

1. Load the tape in the reader.
2. Ready the tape transport.
3. Press LOAD on the H8 front panel.
4. A single beep indicates a successful load.
5. Press GO on the H8 front panel.
6. The console terminal will respond with:

```
HEATH/WINTEK H8 BASIC
BENTON HARBOR BASIC ISSUE # 05.01.00
COPYRIGHT 01/77 BY WINTEK CORP.
```

7. Configure B.H. BASIC or EX. B.H. BASIC as desired, answering the following questions. Prompt each question by typing its first character on the console terminal keyboard.

```
•AUTO NEW-LINE (Y/N)?
•BKSP = 00008/
•CONSOLE LENGTH = 00080/
•HIGH MEMORY = 16383/
•LOWER CASE (Y/N)?
•PAD = 4/
•RUBOUT = 00127/
•SAVE?
*
```

8. Before executing SAVE, have the tape transport ready at the DUMP port.
9. To use BASIC directly from the distribution tape, type the return key at any time rather than a question prompt key. The Console Terminal will display:

```
B.H. BASIC # 05.01.00
*
```

BASIC is ready to use.



Loading From a Configured Tape

1. Load the tape in the tape transport.
2. Ready the tape transport.
3. Press LOAD on the H8 front panel.
4. A single beep indicates a successful load.
5. Press GO on the H8 front panel.
6. The console terminal responds with:

B.H. BASIC # 05.00.00

*

BASIC is ready to use in the configured form.



APPENDIX B

A Summary of BASIC

For additional details, refer to the page number that is given with each of the following topics.

See Page

Numeric Data 5-9

Numbers may be real or integer with the following characteristics:

- Range 10^{-38} to 10^{+37}
- Accuracy 6.9 digits.
- Decimal range 0.1 to 999999.
- Exponential format $(\pm) X.XXXXXE (\pm) NN$.

Boolean Data 5-10

Integer numbers from 0 to 65535 represent two byte binary data from 00000000 00000000 to 11111111 11111111. Fractional parts of numbers between 0 and 65535 are discarded.

String Data (Extended BASIC Only) 5-10

Data is all printed in ASCII characters plus the BELL, BLANK and LINE FEED, with the following characteristics:

- Maximum string length 255 characters.
- Enclosure Quotation marks ("") on both ends.
- Multiple lines Not allowed for a single string.

Variables 5-11

Variables are named by a single letter (A through Z), or a single letter followed by a single number (0 through 9). For example: A or A6.



See Page

Subscripted Variables

5-12

Subscripted variables are named like variables, but are followed by dimensions in parentheses. Subscripted variables are of the form:

$A_n(N_1, N_2, \dots, N_r)$ For example: $A(1, 2, 7)$ or $A6(1, 5)$.

You must use a DIMENSION statement to define the range and number of allowable subscripts for a variable.

Arithmetic Operators

5-14

Listed in order of priority. Operators on the same line have equal precedence. Parenthetical operations are performed first. Precedence is left to right if all other factors are equal.

<u>SYMBOL</u>	<u>EXPLANATION</u>
-	Unary negation logical compliment
\uparrow	Exponentiation. Ex BASIC only
* /	Multiplication division
+ -	Addition subtraction

Relational Operators

5-18

<u>SYMBOL</u>	<u>EXPLANATION</u>
=	Equal to
<	Less than
\leq	Less than or equal to
>	Greater than
\geq	Greater than or equal to
\neq	Not equal to



See Page

Boolean Operators

5-19

Boolean operators perform the Boolean (logical) operations on two integer operands. The operands must evaluate to integers in the range of 0 to 65535. The operators are:

NOT	Logical complement, bit by bit
OR	Logical OR, bit by bit
AND	Logical AND, bit by bit

String Variables

5-21

String variables may be either subscripted or nonsubscripted. They take the same form as numeric or Boolean variables but are followed by a dollar sign (\$) to indicate a string variable. For example: A\$ A6\$ A\$(1,2,7) or A6\$(1,5).

String Operators

5-22

String expressions may be operated on by the relational operators as well as the plus (+) symbol. The plus symbol is used to perform string concatenation.

Line Numbers

5-25

When it is used in the program mode, BASIC requires that each line be preceded by an integer line number in the range 1 to 65535.

The Command Mode

5-23

The command mode does not use line numbers. Statements are executed when a carriage-return is typed.

Multiple Statements on One Line

5-25*

BASIC permits multiple statements on one line. Each statement is separated from the others by a colon (:). DATA statements may not appear on lines with other statements.

*See "Basic Statements."



Command Mode Statements

<u>COMMAND</u>	<u>FORM</u>	<u>DESCRIPTION</u>	<u>SEE Pg.</u>
BUILD	BUILD iexp1, iexp2	<i>Automatically generates program line numbers starting at iexp1 in steps of iexp2.</i>	5-27
CONTINUE	CONTINUE	Resumes program execution.	5-27
DELETE	DELETE [iexp1, iexp2]	<i>Deletes program lines between iexp1 and iexp2.</i>	5-28
DUMP	DUMP "name"	Saves current program "name" on mass storage media at load dump port; "name" is up to 80 ASCII characters.	5-28
LOAD	LOAD "name"	Loads program "name" from mass storage media at load dump port; "name" is up to 80 ASCII characters. Current program is destroyed.	5-29
RUN	RUN	Start execution of current program. Preclears all variables, stacks, etc..	5-30
SCRATCH	SCRATCH SURE?Y	Clears all program and data storage area. Any response to SURE but Y cancels SCRATCH.	5-31
VERIFY	VERIFY "name"	Performs a checksum on the mass storage record titled "name." No response if record is bad.	5-31



Command and Program Mode Statements

<u>COMMAND</u>	<u>FORM</u>	<u>DESCRIPTION</u>	<u>SEE Pg.</u>
CLEAR	CLEAR [varname]	Clears all variables, arrays, string buffers, etc. <i>Optionally clears named variable (varname).</i> <i>Specifies functions and arrays as V.</i>	5-33
CONTROL	CNTRL iexp1, iexp2	<i>CNTRL 0 sets a GOSUB to line iexp2 when a CONTROL-B is typed.</i>	5-34
		<i>CNTRL 1 sets iexp2 digits before exponential format is used.</i>	5-35
		<i>CNTRL 2 controls the H8 front panel. If iexp2: = 0, display off; = 1, display on without update; =2, display on with update.</i>	5-35
		<i>CNTRL 3 sets the width of a print zone to iexp2 columns.</i>	5-35
		<i>CNTRL 4 controls the H8 hardware clock. iexp2 = 0, clock off. iexp2 = 1, clock on.</i>	5-36
DIMENSION	DIMvarname(iexp1 [, . . . ,iexpn]) [,varname2(. . .)]		
		Defines the maximum size of variable arrays.	5-36
FOR/NEXT	FOR var = nexp1 TO nexp2 [STEP nexp3]		
	NEXT var	Defines a program loop. Var is initially set to nexp1. Loop cycles until NEXT is executed; then var is incremented by nexp3 (default is +1). Looping continues until var > nexp2 (or less than nexp2 if STEP is negative). The statement after NEXT is then executed.	5-37



<u>COMMAND</u>	<u>FORM</u>	<u>DESCRIPTION</u>	<u>SEE Pg.</u>
FREE	FREE	<i>Displays the amount of memory assigned to tables and text</i>	5-41
GOSUB/ RETURN	GOSUB iexp RETURN	Transfers execution sequence of program to line iexp (the beginning of a subroutine). RETURN returns execution sequence to the statement following the calling GOSUB.	5-42
GOTO	GOTO iexp	Unconditionally transfers the program execution sequence to the line iexp.	5-44
IF/THEN	IF expression THEN iexp IF expression THEN statement	If the expression is true, control passes to iexp line or to "statement." If the relation is false, control passes to the next independent statement.	5-45
LET	LET var = nexp <i>LET var\$ = sexp</i>	Assigns the value nexp (<i>or sexp in the case of strings</i>) to the variable var (<i>or var\$</i>). LET keyword is optional.	5-46
LIST	LIST[iexp1] [,iexp2]	Lists the entire program on the console terminal. Lists the line iexp1 or the range of lines iexp1 to iexp2.	5-47
ON/GOSUB	ON iexp1 GOSUB iexp2, . . . ,iexpn.	Permits a computed GOSUB. Iexp1 is evaluated and acts as an index to line numbers iexp2 thru iexpn, each pointing to a different subroutine.	5-48



<u>COMMAND</u>	<u>FORM</u>	<u>DESCRIPTION</u>	<u>SEE Pg.</u>
ON/GOTO	ON iexp1 GOTO iexp2, . . . ,iexpn	Permits a computed GOTO. Iexp1 is evaluated and acts as an index to line numbers iexp2 thru iexpn.	5-48
OUT	OUT iexp1, iexp2	Outputs a number iexp2 to output port iexp1.	5-48
PAUSE	PAUSE (<i>iexp</i>)	Ceases program execution until a console terminal key is typed. <i>Ceases program execution for 2 X iexp mS.</i>	5-49
POKE	POKE iexp1, iexp2	Writes a number iexp2 into memory location iexp1.	5-49
PORT	PORT <i>iexp</i>	<i>Assigns the print function to port iexp. Assigns the Console Terminal function to the device at port iexp if iexp is negative.</i>	5-50
PRINT	PRINT(<i>nexp. sep1 . . . nexpn(sepn)</i>)	<i>Prints the value of the expression (s) exp with a leading and trailing space. Expressions may be numeric or string. If the separator is a comma, the next print zone is used. If the separator is a semicolon, no print zones are used. No separator prints each expression value on a new line.</i>	5-50
READ & DATA	READ var1, . . . ,varn DATA exp1 . . . ,expn	The READ statement assigns the values exp1 thru expn in the data to the variables var1 thru varn.	5-54



<u>COMMAND</u>	<u>FORM</u>	<u>DESCRIPTION</u>	<u>SEE Pg.</u>
REMARK	REM	Text following the REM is not executed and is used for commentary only.	5-55
RESTORE	RESTORE	Causes the program to reset the DATA pointer, thus reusing data at the first DATA statement.	5-56
STEP	STEP iexp	Executes iexp lines of the program. Then returns BASIC to the command mode.	5-57

Program Mode Statements

DEF	DEF FN varname (arg list) = exp	
		Defines a single-line program function created by the user.
END	END	Causes control to return to the command mode.
INPUT	INPUT prompt; var1, . . . , varn	Reads data from the console terminal. <i>String data must be enclosed in quotes.</i>
LINE INPUT	LINE INPUT prompt; var1, . . . , varn	<i>Reads string data from the terminal. String data for LINE INPUT is not enclosed in quotes.</i>
STOP	STOP	Causes BASIC to enter the command mode when the statement containing STOP is executed.



Predefined Functions

<u>FUNCTION</u>	<u>DEFINITION</u>	<u>SEE Pg.</u>
ABS (nexp)	Returns the absolute value of nexp.	5-61
ATN (nexp)	Returns the arctangent of nexp (radians).	5-62
COS (nexp)	Returns the cosine of nexp (radians).	5-62
EXP (nexp)	Returns e^{nexp} .	5-62
INT (narg)	Returns the integer value of narg.	5-62
LOG (nexp)	Returns the natural logarithm of nexp.	5-62
PAD (0)	Returns the value of the H8 front panel key pressed. Includes key debounce.	5-63
PEEK (iexp)	Returns the numeric value at memory location iexp.	5-63
PIN (iexp)	Returns the data input from port iexp.	5-63
POS (0)	Returns the current terminal printhead (cursor) position (by column number).	5-63
RND (narg)	Returns a random number. If narg >0, RND is next in the series. If narg = 0, RND is the previous random number. If narg<0, RND algorithm uses narg as a new seed.	5-63
SEG (narg)	Returns the correct eight-bit number to display narg (0-9) on the H8 LEDs.	5-65
SGN (narg)	Returns +1 if narg is positive. Returns -1 if narg is negative. Returns 0 if narg is zero.	5-66
SIN (nexp)	Returns the sine of nexp (radians).	5-66
SPC (iexp)	Positions printhead (cursor) iexp columns to the right.	5-68
SQR (narg)	Returns the square root of narg.	5-66



<u>FUNCTION</u>	<u>DEFINITION</u>	<u>SEE Pg.</u>
TAB (iexp)	Position printhead (cursor) to the iexp th column.	5-68
USR (narg)	Calls a user-written machine language function to evaluate narg.	5-66
FRE (0)	Returns the amount of free memory in B.H. BASIC.	5-67
MAX (nexp1, . . . ,nexpn)	<i>Returns the maximum value of expressions nexp1 thru nexpn.</i>	5-67
MIN (nexp1, . . . ,nexpn)	<i>Returns the minimum value of expressions nexp1 thru nexpn.</i>	5-67
TAN (nexp)	<i>Returns the tangent of nexp (radians).</i>	5-67
CHR\$ (iexp)	<i>Returns the ASCII character iexp.</i>	5-68
STR\$ (narg)	<i>Returns narg encoded into ASCII with leading and trailing blanks as in the print statement.</i>	5-69
ASC (sexp)	<i>Returns the ASCII code for the first character in the string sexp.</i>	5-69
LEFT\$ (sexp, iexp)	<i>Returns the left iexp characters of the string sexp.</i>	5-69
RIGHT\$ (sexp, iexp)	<i>Returns the right iexp characters of the string sexp.</i>	5-69
MID\$ (sexp, iexp1) [,iexp2]	<i>Returns the substring of the string sexp starting with the iexp1 th character and ending with the iexp2 th character if iexp2 is specified. If not specified, returns iexp1 th character to the end.</i>	5-70
VAL (sexp)	<i>Returns the numeric value of the number encoded in the string.</i>	5-70
LEN (sexp)	<i>Returns the length of sexp.</i>	5-70



Editing Commands

<u>COMMAND</u>	<u>FUNCTION</u>	<u>SEE Pg.</u>
CONTROL-C	General-purpose cancel. Returns BASIC to monitor mode from any operation or program execution.	5-71
CONTROL-S	Suspends the output to the console terminal and suspends program execution.	5-72
CONTROL-Q	Restores the output to the console terminal and restores program execution.	5-72
CONTROL-O	Toggles the output discard flag. Does not stop program execution.	5-72
CONTROL-P	Clears the discard flag set by CONTROL-O.	5-72





APPENDIX C

Basic Utility Routines

The following pages contain a description of several utility routines included in BASIC (some are only available in Extended BENTON HARBOR BASIC). They can be used with user-written machine language routines called by the USR function. See "Appendix D" for Utility Routine entry points.



BASIC - WINTER BASIC INTERPRETER,
SELECTED SOURCE LISTING.

HEATH X646M V1.0 92/18/77
13:12:38 01-AK-77 PAGE 1

999,000.....2.....TEXT MTR.....

74 *** BASIC - *WINTER* BASIC INTERPRETER.
75 *
76 * J. G. LÉWIN, ÖY/Y6, FOR *WINTER* CORPORATION.
77 *

79 *** COPYRIGHT 09/1976, *WINTER* CORPORATION.
80 * 902 N. 9TH ST.
81 * LAFAYETTE, IN. 47901

83 *** LOW-MEMORY CELLS USED BY BASIC
84.....
040,064.....040,064.....040,064.....040,066.....040,072.....040,074.....
85.....85.....86.....87.....88.....89.....90.....91.....92.....
ORG.....7*34,UVEC.....DS.....2.....DS.....4.....DS.....2.....DS.....4.....
ACCX TYPE.....ACCY TYPE.....ACCY TYPE.....

BASIC - WINTEK BASIC INTERPRETER.
USR - PERFORM USER ASSEMBLY LANGUAGE FUNCTION.

HEATH X8ASM V1.0 02/18/77
13:12:40 01-APR-77 PAGE 2

```
95 **      USR - CALL USER ASSEMBLY LANGUAGE FUNCTION.  
96 *  
97 *      THE *USR* FUNCTION IS ACTUALLY A CALL TO A USER-WRITTEN ROUTINE  
98 *      WHICH MUST HAVE BEEN PREVIOUSLY LOADED INTO MEMORY BY THE USER.  
99 *  
100 *     THE ADDRESS OF THE FUNCTION'S ENTRY POINT MUST BE IN *USRFCN*.  
101 *  
102 *     BASIC MUST HAVE BEEN PREVIOUSLY CONFIGURED SO THAT THE USER  
103 *     FUNCTION RESIDES IN MEMORY ABOVE THE STACK POINTER (HIGH MEMORY)  
104 *     OR ELSE BASIC WILL OVERLAY THE FUNCTION WITH DATA.  
105 *  
106 *     THE FUNCTION IS ENTERED WITH A POINTER TO THE SINGLE ARGUMENT (IN  
107 *     FLOATING POINT), AND MAY RETURN A FLOATING POINT OR STRING ARGUMENT.  
108 *     IF NO RETURN VALUE IS PLACED IN *ACCX*, THEN THE ORIGINAL ARGUMENT  
109 *     REMAINS THERE, AND USR() RETURNS ITS ARGUMENT AS ITS VALUE.  
110 *  
111 *     ENTRY (BC) = *ACCX  
112 *     EXIT (ACCX) CONTAINS VALUE  
113 *     USES ALL  
114  
115  
116  
--- 000 000 117 USRFCN DW 0          ADDRESS OF USER FUNCTION ENTRY POINT  
118 *          IF = 0, USR() IS NOT LEGAL
```

BASIC - WINTER BASIC INTERPRETER:
ERROR PROCESSING

HEATH X8ASM V1.0 02/18/77
13:12:41 01-AFR-77 PAGE 3

```

121 **      ERROR PROCESSING.
122 *
123 *      THESE ERROR PROCESSORS ARE ENTERED WHEN AN ERROR IS DETECTED.
124 *
125 *      CONTROL PASSES DIRECTLY BACK TO COMMAND MODE.
126
127
128 ERR.CC EQU *      CONTROL-C
129
130 ERR.CB EQU *      CTRL-B
131
132 ERR.DE EQU *      DATA EXHAUSTED
133
134 ERR.DO EQU *      /0
135
136 ERR.IN EQU *      ILLEGAL NUMBER
137
138 ERR.IU EQU *      ILLEGAL USAGE
139
140 ERR.NV EQU *      NEXT VARIABLE MISSING
141
142 ERR.OV EQU *      OVERFLOW
143
144 ERR.RE EQU *      RETURN ERROR
145
146 ERR.SL EQU *      STRING LENGTH
147
148 ERR.SN EQU *      STATEMENT NUMBER
149
150 ERR.SY EQU *      SYNTAX ERROR
151
152 ERR.TC EQU *      TYPE CONFLICT
153
154 ERR.TO EQU *      TABLE OVERFLOW
155
156 ERR.SR EQU *      SUBSCRIPT RANGE
157
158 ERR.SC EQU *      SUBSCRIPT COUNT
159
160 ERR.ND EQU *      NOT DIMENSIONED
161
162 ERR.IC EQU *      ILLEGAL CHARACTER
163
164 ERR.UF EQU *      UNDEFINED FUNCTION
165
166 ERR.TF EQU *      TAPE ERROR

```

BASIC - WINTER BASIC INTERPRETER,
UTILITY SUBROUTINES

HEATH XASM V1.0 02/18/77
13:12:42 01-AFR-77 PAGE 4

169 ** CXV - COPY VALUE INTO 'X' ACCUMULATOR.
170 *
171 * CXV COPIES A .4 BYTE VALUE INTO THE X ACCUMULATOR.
172 *
173 * ENTRY (DE) = ADDRESS OF VALUE
174 * EXIT COPIED
175 * USES A,F
176
177
178 CXV EQU *

180 ** CXY - COPY (ACDX) INTO (ACDY)
181 *
182 * ENTRY NONE
183 * EXIT NONE
184 * USES A,F,B,E
185
186
187 CXY EQU *

188
189 ** CXV - COPY X TO VALUE.
190 *
191 * CXV COPIES THE CONTENTS OF THE 'X' ACCUMULATOR INTO A MEMORY
192 * LOCATION.
193 *
194 * ENTRY (DE) = TARGET ADDRESS
195 * EXIT COPIED
196 * USES A,F
197
198
199 CXV EQU *

200
201 ** IFIX - SPLIT NUMBER INTO INTEGER AND FRACTION.
202 *
203 * IFIX FIXES ((DE)) INTO AN INTEGER.
204 *
205 * ENTRY (DE) = ADDRESS OF NUMBER
206 * EXIT (DE) = INTEGRAL PART OF 0<=N<=65535.
207 * TO ERR. IN OTHERWISE
208
209
210 IFIX EQU *

BASIC - WINTER BASIC INTERPRETER.
UTILITY SUBROUTINES

HEATH X8ASM V1.0 02/18/77
13:12:43 01-APR-77 PAGE 5

```
212 ** IFLT - FLOAT NUMBER.
213 *
214 * ENTRY (DE) = VALUE
215 * EXIT (ACCX) = NUMBER VALUE
216 * (DE) = #ACCX-1
217
218
219 IFLT EQU *
```

```
221 ** TDI - TYPE DECIMAL INTEGER.
222 *
223 * TDI TYPES AN INTEGER AS A 5 PLACE NUMBER. LEADING ZEROS ARE
224 * SUPPRESSED.
225 *
226 * ENTRY (DE) = NUMBER
227 * EXIT TYPED
228 * USES A,F,D,E
229
230
231 TDI EQU *
```

```
233 ** XCY - EXCHANGE (ACCX) WITH (ACCY).
234 *
235 * ENTRY NONE
236 * EXIT NONE
237 * USES A,F
238
239
240 XCY EQU *
```

```
242 ** ZRO - ZERO MEMORY.
243 *
244 * ZRO ZEROS A FIELD OF MEMORY.
245 *
246 * ENTRY (HL) = ADDRESS
247 * (DE) = COUNT
248 * EXIT NONE
249 * USES A,F,D,E,H,
250
251
252 ZRO EQU *
```





255 ** H8 FLOATING POINT FORMAT:
256 *
257 * SINGLE-PRECISION FLOATING POINT NUMBERS ARE REPRESENTED
258 * BY A 4-BYTE VALUE. THE NUMBER CONSISTS OF A 3-BYTE
259 * TWO'S COMPLEMENT MANTISSA, AND A ONE-BYTE BIASED BINARY
260 * EXPONENT. THE NUMBER FORMAT IS:
261 *
262 *
263 * N+0 LEAST SIGNIFICANT MANTISSA BYTE
264 * N+1 MID SIGNIFICANT MANTISSA BYTE
265 * N+2 MOST SIGNIFICANT MANTISSA BYTE
266 * N+3 BIASED EXPONENT
267 *
268 *
269 * EXPONENT:
270 *
271 *
272 * EACH FLOATING POINT NUMBER CONTAINS A BINARY EXPONENT.
273 * THE EXPONENT CONTAINS A BIAS OF 128 (200 OCTAL).
274 *
275 * THUS AN EXPONENT OF 0 IS ENTERED AS 200, AN EXPONENT OF -10
276 * IS CODED AS 1660, ETC. THE NUMBER ZERO IS TREATED AS A
277 * SPECIAL CASE: ITS EXPONENT (AND MANTISSA) IS ALWAYS 0.
278 *
279 *
280 * MANTISSA:
281 *
282 *
283 * THE MANTISSA OCCUPYS 3 BYTES, FOR A TOTAL OF 24 BITS. THE NUMBERS
284 * ARE STORED IN TWO'S COMPLEMENT NOTATION. THE HIGH ORDER
285 * BIT IS THE SIGN BIT, THE NEXT BIT (1000) IS THE MOST SIGNIFICANT
286 * DATA BIT. NOTE THAT FLOATING POINT NUMBERS SHOULD ALWAYS
287 * BE NORMALIZED, SO THAT THE MOST SIGNIFICANT DATA BIT IS THE
288 * OPPOSITE OF THE SIGN BIT'S VALUE.
289 *
290 * THE FLOATING POINT ROUTINES SUPPLIED WILL NOT OPERATE ON NON-
291 * NORMALIZED DATA.
292 *
293 * EXAMPLES:
294 *
295 *
296 * DECIMAL NUMBER M3 M2 M1 EX
297 *
298 * 1.0 000 000 100 201
299 * 0.5 000 000 100 200
300 * 0.25 000 000 100 177
301 * 10.0 000 000 120 204
302 * 0. 000 000 000 000
303 * 0.1 146 146 146 175
304 * -1.0 000 000 200 200
305 * -0.5 000 000 200 177
306 * -10.0 000 000 260 204

BASIC - WINTER BASIC INTERPRETER.
FLOATING POINT ROUTINES.

HEATH X8ASM V1.0 02/18/77
13:12:46 01-AFR-77 PAGE 7

```

309 **      FFADD - FLOATING POINT ADD.
310 *
311 *      ACCX = ACCX + (DE)
312 *
313 *      ENTRY (DE) = POINTER TO 4 BYTE FP VALUE
314 *      EXIT   ACCX = RESULT
315 *      SUPPLIED VALUE UNCHANGED
316 *      USES   A,F
317
318
319 FFADD EQU *
-----
```



```

321 **      FFPSUB - FLOATING POINT SUBTRACT.
322 *
323 *      FFPSUB COMPUTES (DE) - ACCX
324 *
325 *      ENTRY (DE) = POINTER TO 4 BYTE FP VALUE
326 *      EXIT   ACCX = RESULT
327 *      SUPPLIED VALUE UNCHANGED
328 *      USES   A,F
329
330
331 FFPSUB EQU *
-----
```



```

333 **      FFNRM - FLOATING POINT NORMALIZE.
334 *
335 *      FFNRM NORMALIZES THE CONTENTS OF (ACCX).
336 *
337 *      ENTRY NONE
338 *      EXIT   (ACCX) NORMALIZED
339 *      USES   A,F
340
341
342 FFNRM EQU *
-----
```



```

344 **      FFNEG - FLOATING POINT NEGATE.
345 *
346 *      FFNEG NEGATES THE CONTENTS OF ACCX.
347 *
348 *      ENTRY NONE
349 *      EXIT   (ACCX) = -(ACCX)
350 *      USES   A,F
351
352
353 FFNEG EQU *
-----
```



BASIC - WINTER BASIC INTERPRETER.
FLOATING POINT ROUTINES.

HEATH X8ASM V1.0 02/18/77
13:12:47 01-APR-77 PAGE 8

RELA
TIVE
BASIC

355 ** FPMUL - FLOATING POINT MULTIPLY.
356 *
357 * ENTRY (DE) = ADDRESS OF Y
358 * EXIT ACCX = ACCX * Y
359 * USES A,F
360
361
362 FPMUL EQU *

364 ** FPDIV - FLOATING POINT DIVIDE.
365 *
366 * ACCX = ACCX/Y
367 *
368 * ENTRY (DE) = POINTER TO Y
369 * EXIT (ACCX) = RESULT
370 * USES A,F
371
372
373 FPDIV EQU *

DAVID WINTER BASIC INTERPRETER.
ASCII/FLOATING CONVERSION ROUTINES.

HEATH X8ASM V1.0 02/18/77
13:12:48 01-APR-77 PAGE 9

```

376 ** ATF - ASCII TO FLOATING.
377 *
378 * ATF CONVERTS AN ASCII STRING INTO A FLOATING POINT VALUE
379 * IN ACCX.
380 *
381 * SYNTAX
382 *
383 * NNNN [.,NNNN] DE [+/-] NNN
384 *
385 * ENTRY (HL) = ADDRESS OF TEXT
386 * EXIT (HL) UPDATED
387 * (ACCX) = VALUE
388 * USES A,F,H,L
389
390
391 ATF EQU *

```

```

393 ** FTA - FLOATING TO ASCII.
394 *
395 * FTA CONVERTS A FLOATING POINT NUMBER INTO AN ASCII
396 * REPRESENTATION.
397 *
398 * ENTRY (ACCX) = VALUE
399 * (HL) = ADDRESS TO STORE TEXT
400 * EXIT (A) = LENGTH OF STRING RECORDED
401 * (DE) = ADDRESS OF LAST BYTE
402 * USES A,F,D,E
403
404
405 FTA EQU *

```



BASIC - WINTER BASIC INTERPRETER.
FLOATING POINT CONSTANTS.

HEATH X8ASM V1.0 02/18/77
13:12:49...01-APR-77...PAGE...10.

408 ** FLOATING POINT VALUES.
409 *
410
411 LON G LIST GENERATED BYTES
412
000 000 100 413 FP1.0 DB 0,0,1000,2010
201
000 000 120 414 FP10. DB 0,0,1200,2040
204
146 146 146 415 FFO.1 DB 1460,1460,1460,1750
175
000 000 000 416 FFO.0 DB 0,0,0,0
000
022 170 233 417 NPI.2 DB 0220,1700,2330,2010 -PI/2
201
022 170 233 418 NPI2 DB 0220,1700,2330,2030 -PI*2
203
022 170 233 419 NPI DB 0220,1700,2330,2020 -PI
202
022 170 233 420 NPI.4 DB 0220,1700,2330,2000 -PI/4
200
356 207 144 421 PI.4 DB 3560,2070,1440,2000 PI/4
200
422
423
425 END

ASSEMBLY COMPLETE
425 STATEMENTS
0 ERRORS
20312 BYTES FREE





APPENDIX D

Entry Points to BASIC Utility Routines

ADDRESSES FOR
EXTENDED RENTON HARBOR BASIC
ISSUE # 10.01.

ACDX 040.066	ACCY 040.074	ATF 076.051	CUX 067.041
CXY 067.061	CXY 067.056	ERR.CB 064.246	ERR.CC 064.236
ERR.D0 064.267	ERR.DE 064.256	ERR.IC 065.055	ERR.IN 064.274
ERR.IU 064.302	ERR.ND 065.046	ERR.NV 064.310	ERR.OV 064.317
ERR.RE 064.330	ERR.SC 065.033	ERR.SL 064.336	ERR.SN 064.350
ERR.SR 065.016	ERR.SY 064.361	ERR.TC 064.367	ERR.TD 065.004
ERR.TP 065.077	ERR.UD 065.070	FF0.0 103.206	FF0.1 103.202
FPI.0 103.172	FF10. 103.176	FFAID 073.106	FFDIV 075.006
FFMUL 074.051	FFNEG 074.030	FFNRM 073.330	FFSUB 073.314
FTA 076.336	IFIX 067.226	IFLT 067.264	NPI 103.222
NPI.2 103.212	NPI.4 103.226	NPI2 103.216	P1.4 103.232
TU1 071.307	USRECN 103.163	XBY 071.326	ZRD 071.360



ADDRESSES FOR

BENTON HARBOR BASIC
ISSUE # 05.01.

ACCX	040.066	ACLY	040.074	ATF	065.144	COX	060.121
CXV	060.211	CXY	060.206	ERR.CB	-none-	ERR.CC	057.100
ERR.DD	057.110	ERR.DE	057.105	ERR.IC	057.154	ERR.IN	057.113
ERR.IU	057.116	ERR.ND	057.151	ERR.NV	057.121	ERR.OV	057.124
ERR.RE	057.127	ERR.SC	057.146	ERR.SL	-none-	ERR.SN	057.132
ERR.SR	057.143	ERR.SY	057.135	ERR.TC	-none-	ERR.TD	057.140
ERR.TP	057.162	ERR.UD	057.157	FF0.0	-none-	FF0.1	072.101
FP1.0	072.071	FP10.	072.075	FFADD	063.033	FFDIV	064.069
FFMUL	063.274	FFNEG	063.260	FFNRM	063.207	FFSUB	063.173
FIA	066.006	IFIX	060.373	IFLT	061.031	NPI	-none-
NPI.2	072.105	NPI.4	-none-	NPI2	072.111	PI.4	-none-
TDI	062.327	USRFIN	072.065	XCY	062.346	ZRD	063.000



HASL #04.01.00

PAGE 1

117.220			ORG	120000A-160Q
063.207			EQU	063207A
117.220	003	FPNRM	START	INX B INC UP
117.221	003			INX B TO
117.222	003			INX B EXPONENT
117.223	012			LDAX B (A) = ACCX EXP
117.224	247			ANA A SET CONDX CODE
117.225	310			RZ
117.226	075			DCR A /2
117.227	312 233 077			JZ USR1 IF UNDER FLOW
117.232	075			DCR A /2 AGAIN (/4)
117.233	002	USR1		STAX B RET TO ACCX
117.234	315 207 063			CALL FPNRM NORMALIZE
117.237	311			RET IN CASE 0
117.240			END	START

STATEMENTS = 00016

FREE BYTES - 10331

NO ERRORS DETECTED.



APPENDIX E

An Example of USR

The following program demonstrates BASIC's USR function. This USR program, which performs the simple task of dividing the USR argument by four, was written under TED-8 (see Page 3-37) and assembled by HASL-8 (see Page 4-53). The object program can be written onto tape and then loaded into H8 memory, it can be written directly into memory by HASL-8, or because it is short it can be entered via the H8 keypad. The program works as follows, and is printed out on Page 5-96.

1. When the USR function is called, it finds the starting address of the user-written program at USRFCN. Therefore, the starting address of this program (117 220) is entered in these locations. See "Appendix D" for the Address of USRFCN.
2. The BC register pair point to BASIC's floating point accumulator (ACCX). See Page 5-95 for ACCX description.
3. To divide the number in ACCX by four, the ACCX exponent is shifted left twice. This is accomplished in the following steps.
 4. The BC pair is incremented three times, after which it points to the ACCX exponent.
 5. The contents of the ACCX exponent is loaded into the 8080 accumulator.
 6. The 8080 accumulator is shifted left one, dividing the exponent (and therefore the number) by two.
 7. After the first left shift, we test for an underflow, which would indicate the number is out of BASIC's range.
 8. If there is no under flow, we shift left again to complete the divide by four.



9. Once the divide by four is complete, the value in the accumulator is placed back in ACCX exponent.
10. We now call the floating point normalization routine (FPNRM) from BASIC. This normalizes the number in the ACCX in case the result was zero.
11. Once normalized, a RET (return) instruction is used to return to BASIC, ending the USR program.

To run this program, you must reserve room at the top of memory for it, and for PAM-8's stack should an RST be executed. For this reason, the program was started 160 (octal) bytes below the top of memory. BASIC must be configured with high memory set below 117 220 (location 20304 decimal). NOTE: You will need at least 12K of memory to use the USR Function.



INDEX

NOTE: Numbers printed in a bold type face refer to examples of the indicated statement or function.

- ASCII Function, 5-69
- Absolute Value, 5-61
- Addition, 5-14, 5-16
- AND, 5-19
- Arc Tangent Function, 5-62
- Arithmetic, 5-9
- Arithmetic, Functions, 5-61 ff,
- Arithmetic Operators, 5-14
- Arithmetic Priority, 5-14
- Arrays, 5-12 ff, 5-21, 5-33, 5-36
- Assignment Statement, 5-11, 5-45
- Asterisk, 5-7, 5-14

- Backspace, changing of, 5-71 (0-20)
- BASIC File, 0-15 ff, 5-30
- Basic Statements, 5-25
- BENTON HARBOR BASIC, 5-7
- Blanks (spaces), 5-73
- Boolean Values, 5-10
- Brackets, 5-26
- BUILD, 5-27

- Character Function, 5-68
- CHR \$, 5-68
- CNTRL-H, 5-71
- CNTRL, 5-34 ff, 5-10, 5-52
- CNTRL-Q, 5-72
- CNTRL-S, 5-72
- Checksum Error, 5-30, 5-32
- CLEAR, 5-33, 5-56
- Clear Varname, 5-33
- Clock, 5-36
 - Pause, 5-49
- Colon 5-25, **5-37**, 5-44
- Comma, 5-50 ff,
- Command Completion, 5-7, 5-72
- Command Mode, 5-23 ff, 5-33
- Comments, 5-55
- Concatenation, 5-22
- Continue, 5-23, 5-27, 5-60, 5-58, 5-71
- Control-B, 5-34
- Control-C, 5-71
 - Abort List, 5-47
- Control-O, 5-34, 5-47
 - Abort List, 5-47
- Cosine Function, 5-62
- Cube, **5-34**



- DATA, 5-54 ff, 5-56, 5-54
 Data Exhausted, 5-77
 Data Only Statement,
 One Line, 5-55
 Decimal Notation, 5-10
 DEF FN 5-58
 DELETE, 5-28, **5-28**
 DIM (Dimension), 5-12, 5-13, 5-36
 Discard Flag CNTRL-O, 5-72
 Discard Flag CNTRL-P, 5-72
 Displays Control, 5-35
 Divide by Zero, 5-75
 Division, 5-14
 Dollar Sign (\$), 5-21
 Double Commas, 5-52 ff,
 DUMP, 5-28
- END, 5-58
 Equal Sign, 5-18, 5-22, 5-46
 Errors, 5-75 ff, 5-42
 Errors Recover, 5-75
 ERROR Table, 5-77
 Exponential Format, 5-9
 Exponential Function, 5-62
 Exponential Notation, 5-9, 5-35
 Exponentiation, 5-14 ff,
 Expressions, 5-14
 Extended B.H. Basic, 5-7
- False, 5-18
 FOR, **5-24**, **5-37**, **5-39**, 5-39 ff.
 FREE EX, 5-41
 Free Space Function (FRE), 5-67
 Functions, Predefined, 5-61 ff,
- GOSUB, 5-34, **5-41**, 5-42 ff,
 GOTO, 5-44
- High Memory, 5-6
 iexp, 5-26
 IF GOTO, 5-45
 IF THEN, 5-18, 5-45
 IGNORE, 5-30, 5-32
 Immediate Execution, 5-23
 Input and Line Input, 5-59
 Inputting Control, 5-71
 Integer Function, 5-62
 Integer Numbers, 5-9
 I/O MAP, 0-49
- Label File, 0-12 ff,
 Left String Function, 5-69
 LET, 5-46
 Lexical Rules, 5-73
 Line Deletion, 5-74
 LINE Input, 5-59
 Line Insertion, 5-73
 Line Length, 5-74
 Line Numbers, 5-25
 Line Replacement, 5-74
 LIST, 5-47, **5-73**
 LOAD, 5-29
 Loading Basic, 5-7
 Logarithm Function, 5-62
 Loop, 5-39 ff,
- Map, I/O, 0-49
 Map, MEMORY, 0-50
 Maximum Function, 5-67
 Memory, 5-6
 Poke, 5-49
 Map, 0-50
 Middle String Function, 5-70
 Minimum Function, 5-67
 Multiple Statements, 5-24
 Multiplication, 5-14 ff,



“Name”, 5-26
Negation, 5-14, 5-15
Negotion, 5-14
Nesting Depth, 5-40, 5-44
Nesting, 5-40 ff
nexp, 5-26 ff
NEXT, **5-24**, **5-37**, **5-39**, 5-37 ff
NOT, 5-14, 5-19
Numeric Data, 5-9
Numeric Value Function, 5-70
NXT, 5-57

ON . . . GOSUB, 5-48
ON . . . GOTO, 5-48
Operators, 5-14
OR, 5-19
OUT, 5-48
Output Port, 5-48
Output Restoration, 5-72
Output Suspension, 5-72
Outputting Control, 5-72

PAD Function, 5-63
Parenthesis, 5-15
PAUSE, 5-49
PEEK, 5-63
POKE, 5-49
PORT, 5-50
Position Function, 5-63
Predefined Functions, 5-61 ff,
PRINT, **5-37**, **5-38**, 5-50 ff
Printing Strings, 5-51
Printing Variables, 5-50
Print Zone, 5-35
Priority, Arithmetic, 5-14 ff,
Program Loop, 5-37
Program Only Mode, 5-25 ff, 5-33, 5-58
Prompt,
 Basic, 5-7, 5-71
 Input, 5-59

Quotes,
 Input, 5-59
 Line Input, 5-59
 Strings, 5-68

Random Function, 5-63
READ, 5-54 ff,
Real Numbers, 5-9
Record, 0-12 ff,
Relational Operators, 5-18, 5-22
REM (Remark), **5-55**
RESTORE, 5-56
RETURN, 5-35, 5-42 ff,
Right String Function, 5-69
RND, 5-63
Rubout, changing of, 5-71 (0-20)
RUN, 5-30

SCRATCH, 5-31
Segment Function, 5-65
Semicolon, 5-52 ff,
Sep, 5-26
Sequence Error, 5-29, 5-32
sexp, 5-26
SGN 5-66
Sign Function, 5-66
Sine Function, 5-66
Single Statements, 5-24
Single Step Execution, 5-57
Space Function, 5-68
Spaces, see “Blanks”, 5-73
Special Feature Functions, 5-61 ff,
SQUARE (Example), 5-24, **5-34**, **5-56**
Square Root Function, 5-66
Statement Length, 5-25
Statements, 5-24 ff,
Statement Types, 5-25
Step, FOR/NEXT, 5-37 ff,
STEP, 5-57



- STOP, 5-60
String Buffers, 5-29
String Data, 5-10
String Functions, 5-68
String Operators, 5-22
Strings, 5-21
String Variables, 5-21
Subroutines, 5-42 ff,
Subscripted Variables, 5-12
Subtraction, 5-14, 5-16
SURE, 5-31
- Trailing Blanks, 5-24
True, 5-18
Truncation, 5-10
- Unary Operators, 5-14 ff
USE Error, 5-26
User Defined Function,
 Single Line (DEF-FN), 5-56
Machine Language (USR), 5-33, 5-66
- TAB Function, 5-68
Tangent Function, 5-67
Tape Error, 5-30
Text Rules, 5-73
- VAL 5-70
Var, 5-26
Variables, 5-11, 5-33
Verify, 5-31