SOFTWARE REFERENCE MANUAL

HDOS DISK OPERATING SYSTEM

VERSION 3.0

This manual was originally (1990) issued as a series of text files. In the conversion to .pdf format, I have made minor adjustments to pagination and layout as I have combined the separate files into a single document.

No changes have been made to the contents of the document other than the correction of an occasional misspelling.

Jack Rubin
SEBHC
May, 2003

HEATH DISK OPERATING SYSTEM

SOFTWARE REFERENCE MANUAL

VERSION 3.0

HDOS 3.02 is an enhanced version of HDOS 2.0. The long-awaited HDOS 3.0 was originally conceived by William G. Parrott III, David T. Carroll, and Dale L. Wilson. Due to circumstances beyond their control David and Dale dropped out, and another programmer, Richard Musgrave, joined Bill Parrott and assumed the responsibility of developing the new operating system. After completing HDOS 3.0, Bill moved on to other pursuits, and Richard has continued to develop, debug, and improve HDOS 3.0, providing an enhancement called HDOS 3.02. Refer to Chapter 14, page 14-3, "WHAT'S NEW", for a capsule summary describing the HDOS 3.02 features.

This manual, based on the original work, presents the new version, HDOS 3.02. Within this latest version, you will find many useful commands, MS-DOS emulating functions, such as "batch files," and other interesting features. This has proven to be the most fascinating operating system yet available for our Heath H89 family of computers.

HDOS 3.0 SOFTWARE REFERENCE MANUAL CONTENTS

A Table of Contents is included at the beginning of each chapter. In order to readily distinguish one chapter from another, the page numbers are sectionalized. For example, in chapter 1, the pages are serially numbered 1-1, 1-2, 1-3, etc., while in chapter 2, the pages are serially numbered 2-1, 2-2, 2-3, etc., and so on to include all of the chapters.

The overall manual is divided into the following chapters:

SOFTWARE REFERENCE
MANUAL

HDOS DISK OPERATING SYSTEM

VERSION 3.0

CHAPTER 1

SYSTEM CONFIGURATION

TABLE OF CONTENTS

INTRODUCTION

Most of the material found in chapters one through three cover the fundamentals of the HDOS 3.0 Operating System. This is necessary for beginners in order to provide the proper background to enable them to use HDOS 3.0 efficiently. If desired, advanced users may begin in Chapter 4, SYSCMD/Plus.

Heath Disk Operating System, Version 3.0, is an updated, sophisticated library of programs which controls and facilitates the many and diverse applications of your computer. The H8/H89/Z90/HDOS system has set a new standard in the hobby computer industry. This was the first hobby system to offer many of the functions of large, commercial computer systems. In addition, this genre of computers have proven to be very trustworthy and dependable. Most of the owners are very loyal to their computer systems. The satisfaction you will derive from the system is limited only by your own imagination.

Even if this is your first use of an H8, H89, or Z90, you will be able to set up your computer system quickly. Immediately following the "System Configuration" Chapter Two "General Operations" provides a description of the basic functions performed by the computer. Hearken well to these procedures, since you will be using them frequently, as long as you retain your computer. Hence, they are to be memorized. Continuing with Chapter Three, "System Optimization," you will be shown how to optimize your system for best results.

It will require about three hours to complete the "System Set-Up Procedure." Do not be reluctant to stand up and stretch or get a drink if you have to, even if you are in the middle of a copying operation. The computer will be patiently waiting when you return.

If in the course of following the instructions you mistype a word or command, and you notice it before you have pressed the RETURN key, you can instruct HDOS to ignore the entry. To do this, hold down the CTRL key while simultaneously typing the character 'U.' "^U" will appear on the screen, HDOS will automatically execute a carriage return, and you can then retype the line. This applies to a situation when you are at the HDOS system prompt, and not when you are in an editor or word processor.

If you do not notice a mistyped word until after you have pressed RETURN, do not panic. HDOS will attempt to make sense of your word (STSGEN, for example), and when it cannot, it will print an error message on the screen. The HDOS 3.0 system prompt will be redisplayed. To restart, just retype the command line again.

Please bear in mind that you must type a CARRIAGE RETURN, hereinafter referred to as <RTN>, after typing in a line. If you type a line and do not enter a carriage return afterwards, the computer will not receive your command. This applies to a situation when you are at the HDOS system prompt.

INTRODUCTION (Cont)

You may find it reassuring to know that it is impossible to damage the computer by typing in the wrong word or command. Unless you have a substantial understanding of assembly language, it is difficult to damage the HDOS 3.0/3.02 Operating System.

However, you may damage a disk's contents when using the INIT program. When using this program, it is important not to wipe out your system disk. Also if your system has multiple types of drives, consisting of both 48 tpi and 96 tpi drives, do not inadvertently instruct your computer to format a 96 tpi disk in a 48 tpi drive. This practice could damage a disk drive.

Most importantly, remember that the computer is patient. It will wait for you to type a command correctly, no matter how long it takes. And while it will not reward you for typing the command correctly (referred to as "syntax"), the computer will nevertheless not begrudge you many unsuccessful attempts.

> NOTE
>
> You will notice references made to "80-track" drives and "96 tpi drives"; also "40-track" drives and "48 tpi drives." These terms (i.e., 80-track/96 tpi; and 40-track/48 tpi) mean the same.

[To check out unfamiliar computer terms, refer to Appendix 1-A, "Glossary of Terms."]

****

NOTATION CONVENTIONS

0	Zero (Used where the numerical zero may be confused with the letter "O".)
^	Required space. Pay special attention to the spaces shown in the examples in this manual. If a space is either omitted or inserted improperly, it could cause your command to fail!
<RTN>	Carriage return. Produced by pressing the RETURN key. This moves the cursor to the first column to the left on the following line, depending upon the left margin setting. It also tells the computer to execute a command when entered at the end of a command line.
<xxx>	The default response. If you press the RETURN key after this message, HDOS assumes that you intend the reply which is inclosed in the < > symbols.

NOTATION CONVENTIONS (Cont)

" Statements made by the computer are set off by quotations, except where dialogue is obvious.

' Responses keyed by the user are set off in apostrophes, except where dialogue is obvious.

SYn: The primary boot disk drives, where "n" is the specific hardware-configured drive that boots by default. That is when you boot the computer by just pressing "B." HDOS can accommodate up to four disk drives on the H37 (soft-sectored) disk drive chain. For example, one usually addresses a disk drive with an expression such as SY1:, SY2:, or SY3:, or in HDOS 3.02, a simple 1:, 2:, or 3:. with the expression followed by a FULL COLON. This may be compared to addressing your friends, Bill, Tom, or Terry, or whatever.

DKn: The secondary boot drives, where "n" is the specific hardware-configured drive that boots on the secondary drive line. That is, when you boot the computer by pressing "Bn:." Heath standard is to assign the H17 (hard-sectored) as the primary drives, but this assignment may be changed. Essentially, DKn: means a different kind of disk drive than the primary type. Therefore, in some instances it could be soft-sector. In other instances, it might be an 8-inch drive. For details how to toggle primary/secondary drives, refer to page 1-10.

DYn: A number of Heath-approved vendors offer accessories, such as hard disks or RAM cards. These accessories generally come with their own device drivers. These device drivers are assigned unique filenames, such as "DYn:."

nn Used to indicate a numeric message that will vary from program to program. When you translate the instructions in a printed documentation to a command typed on the computer screen, you are expected to substitute a digit for each of the letters.

Asterisks: A row of asterisks indicates a change of topic.

****

CONTROL SEQUENCES

You can execute all of the following control codes except DELETE, BACKSPACE, and SHIFT-RESET, by holding down the CTRL key while typing the appropriate letter. Thus, to execute CTRL-C, hold down the CTRL key while simultaneously typing the C key.

DELETE
BACKSPACE — If you press either of these keys, the system will remove the character to the left of the cursor from the screen so that you can retype it. HDOS will either echo each deleted character so you can see which characters have been deleted, or will remove the character from the video screen. These two functions are controlled by a program called "SET.ABS." For details, refer to page 3-19.

CTRL-D — You can use this code to exit from a utility program, such as INIT, or ONECOPY, and the computer will return you to the HDOS system prompt. In many contexts, HDOS considers the CTRL-D code to signify "end of file" or "end of input data."

CTRL-G — This code will cause the computer to beep.

CTRL-U — When you mistype a command at the system prompt and notice the error before you enter a carriage return, CTRL-U instructs HDOS to ignore the line so that you can retype it.

CTRL-Z
CTRL-Z — When you strike these keys twice in succession, any ongoing HDOS activity is cancelled. You will generally use this code when all else fails to return to the HDOS command mode from a utility program, such as ASM or BASIC. This code will often let you escape from a program which has hung up.

SHIFT-RESET — When you press these two keys simultaneously, the H89/Z90 will return to the monitor ROM prompt. The monitor ROM prompt is what you see on the screen before you boot the system, normally H: on an unmodified H/Z89/Z90 computer. CAUTION: this sequence should be used to exit a hung program only in the event that CTRL-Z CTRL-Z will not work, since there is some risk of data loss. To make it hard to do this accidentally, only the right-hand SHIFT key will work. The left-hand key is locked out. [For H89/Z90 computers only.]

RESET
0 (H8) — When you press these two keys simultaneously, the H8 will return to system prompt. CAUTION: This sequence should be used to exit a hung program only in the event that CTRL-Z CTRL-Z will not work, since there is some risk of data loss. [For the H8 computer only.]

SYSTEM CONFIGURATION

The following paragraphs and tables describe the minimum requirements of your operating system. This section contains only information about software configuration. Refer to Appendix 1-B, "Configuring Hardware" for instructions on how to configure your hardware.

You must have an H8 or H89/Z90 computer system with a minimum of 32k bytes of operational memory. HDOS 3.0 is ORG-0. This does not mean that the program area (USERFWA) is near zero, but rather that the system itself (i.e., HDOS30.SYS) is loaded in low memory. This provides the user with about 5k of additional memory for programs. This memory block was not available in HDOS 2.0. HDOS30.SYS starts at memory location 000000, instead of 040.000A in HDOS 2. This situation is normal unless the computer system has been modified.

Table 1-1 outlines the port allocation scheme used on the H89/Z90 computer system. Table 1-2 lists port allocations for the H8. Both tables list the device and the device name used in the software. The software device name is a special name by which the software recognizes various physical devices (peripherals). For example, the software recognizes commands that involve the line printer only if the software device name for the line printer, LP:, is specified along with the command. For additional port data, refer to Appendix 1-C, "Port Assignments."

In the following tables, addresses are given only in octal, indicated by a series of digits followed by a "Q." The operating system does not use any ports below 100Q. The ports shown below are available for your own use.

PORT ALLOCATIONS FOR THE H89

TABLE 1-1

DEVICE	DEVICE NAME	I/F CARD	PORT ADDRESS	INTERRUPT LEVEL
Console Terminal	TT:	---	350-357Q	3
Line Printer	LP:	H88-3	340-347Q	(Note 6)
Alternate Terminal	AT:	H88-3	320-327Q	(Note 6)
5" Hard Sector Drive	SYn:/DKn:	H88-1	174-177Q	
5" Soft Sector Drive	SYn:/DKn:	Z89-37	170-174Q	
8" Soft Sector Drive	SYn:/DKn:	H89-47	170-173Q or 174-177Q	
8" Winchester HardDsk	(Note 4)	H89-67	170-173Q or 174-177Q	
3rd Serial Port	(Note 5)	H88-3	330-337Q	(Note 6)

NOTES:

1. If you are using an H36 DECWRITER as a line printer, you must connect it as device AT:. (NOTE: AT: is the Alternate Terminal.)

2. In most cases, the line printer should be connected as device LP:.

SYSTEM CONFIGURATION (Cont)

PORT ALLOCATIONS FOR THE H89 (Cont)

TABLE 1-1

3. See Chapter 1, Appendix 1-C, for details concerning port assignments for HDOS 3.0.

4. H67 was never supported by HDOS v.3.0, although a device driver was available from non-Heath vendors.

5. Certain printer drivers may be configured to use this port instead of the standard line printer port.

6. Interrupt levels 3, 4, and 5 are available on the 3-port serial card, but current Heath software does not use any interrupts. Certain non-Heath communications software (i.e. modem programs) require a level 5 interrupt on the third serial port.

PORT ALLOCATIONS FOR THE H8

TABLE 1-2

DEVICE	DEVICE NAME	I/F CARD	PORT ADDRESS	INTERRUPT LEVEL
Console Terminal	TT:	H8-5	372-373Q	3
	TT:	H8-4	350-357Q	3
Line Printer	LP:	H8-4	340-347Q	(Note 6)
Alternate Terminal	AT:	H8-5	320-327Q	(Note 6)
		H8-4	320-327Q	---
Front Panel	None	None	360-361Q	---
5" Hard Sector Drive	SYn:/DKn:	H8-17	174-177Q	
5" Soft Sector Drive	SYn:/DKn:	Z8-37	170-174Q	
8" Soft Sector Drive	SYn:/DKn:	H8-47	170-173Q or 174-177Q	
8" Winchester HardDsk	(Note 4)	H8-67	170-173Q or 174-177Q	
3rd Serial Port	(Note 5)	H8-4	330-337Q	(Note 6)

NOTES:

1. If you are using an H36 DECWRITER as a terminal, connect it as device TT:. If you are using an H36 as a line printer, connect it as device AT:. NOTE: The term AT: indicates an alternate terminal.

2. A line printer should be connected as device LP:.

3. See Chapter 1, Appendix 1-C, for details on port assignments HDOS 3.0.

SYSTEM CONFIGURATION (Cont)

PORT ALLOCATIONS FOR THE H8 (Cont)

TABLE 1-2

4. H67 was never supported by HDOS 3.0, although a device driver was available from non-Heath vendors.

5. Certain printer drivers may be configured to use this port instead of the standard line printer port.

6. Interrupt levels 3, 4, and 5 are available on the card, but current Heath software does not use any interrupts. Certain non-Heath communications software (i.e. modem programs) requires a level 5 interrupt on the third serial port.

In addition to port assignments, these tables list the interface card(s) normally used with each device. You will need to install jumpers on the interface card in order to select the appropriate address.

SETTING UP A SYSTEM

Disk Drives and Monitor ROMs

Limited by the design of the respective controller design, HDOS 3.02 supports up to seven disk drives: a maximum of three 5-1/4 inch drives connected to the H17 hard sector controller, and a maximum of four drives connected to the H37 or H47 soft sector controller. If your system includes the H47 controller, it can utilize a maximum of four 5-1/4 inch drives connected to the soft sector controller OR three drives connected to the hard sector controller, and two drives connected to the H-47 controller. There are only two sets of disk drive I/O ports on the H89 buss - 170-173Q and 174-177Q. By using a custom I/O decoder ROM, it is possible to add a third type of disk drive, using custom software also. The third (modified) controller will not be bootable, unless the Monitor ROM, MTR-90 is also modified.

Because the H8 buss is more complete than the H89 buss, implementation of additional types of disk drives is simpler.

If you have only one type of disk drive in your system, the drive which is hardware configured to be drive zero, is normally designated SY0: (system unit zero). Other disk drives of the same type are numbered sequentially from one (i.e. SY1:, and SY2:). If you have not altered your computer hardware, the SYn: (primary boot) drives are the 5-1/4 inch, H17 hard sector drives.

If you have drives of two different types in your system, HDOS assigns names in the format SYn: to all drives that have been hardware-configured as the primary boot drives. The "n" in the SYn: format typically corresponds to the hardware number of each primary boot drive. Those drives in the system which have been hardware-configured

SETTING UP A SYSTEM (Cont)
Disk Drives and Monitor ROMs (Cont)

as secondary non-boot drives are assigned names in the format DKn:. Again, the "n" in the DKn: format corresponds to the hardware number of each secondary drive, if you have not altered your hardware. Either the soft-sector drives or the 8-inch drives can be the DKn: (secondary boot) drives.

It is possible to reverse this assignment with dip-switch SW501, located on the CPU board. The "stock H89" comes configured to boot the H17 drives. This can be changed to boot the H37 or H47 drives as primaries, instead of the H17 drives. In this case, the H17 drives become the secondary drives. To perform the modification to make the H37 the primary boot drives, set pins 4 and 5 to "1" on SW501. To make the H47 the primary boot drives, set pins 1 and 3 to "1" on SW501. All other pins should be set to zero. If you lack the skill or confidence to change the switch, you can always boot a secondary drive from the monitor ROM with a different start-up command. For example: Boot SD1<RTN>, where the SD1 stands for secondary drive DK1:. Whatever drive you use to boot from becomes "SY0:," and the non-boot drives connected to a different type of controller become the "DKn:" drives.

SY0: is the "system" drive unit. This drive most usually contains a disk with the HDOS system files on it. If you are running an H17 computer system with one drive, and you require more disk space, there is a solution to your problem. After you boot your normal system disk, first load your drivers, then RESET your system disk, and install a data disk containing only SYSCMD.SYS and PIP.ABS. Prior to leaving HDOS via either the QUIT or BYE command, reset SY0:, and replace the original system disk.

Initially, only your distribution disk contains system files. After completing the "System Set-Up" procedure, you will have created a SYSTEM VOLUME, which will contain copies of all the necessary files from the distribution disk. For normal operation, the SYSTEM VOLUME will always be mounted in SY0:.

NOTE: Use the distribution disks only to prepare your first set of working SYSTEM VOLUMES. Do not use the distribution disk to run programs, other than those specified in this Software Reference Manual.

The distribution disks are write-protected to guarantee that you will always have an accurate copy of the operating system. Do not disable the write protection by removing the write-protect tabs on 5 1/4-inch disks or installing the write-protect tabs on 8-inch disks.

Under normal conditions you will probably need extra disks for storing data and programs. We recommend that you have at least two copies of your SYSTEM VOLUME, as well as a spare copy of any important information. The process of creating SYSTEM VOLUMES and "backup" disks will be discussed within the "System Set-Up Procedure" section.

SETTING UP A SYSTEM (Cont)
Disk Drives and Monitor ROMs (Cont)

NOTES

Details on booting techniques may be found in Chapter 2, Appendix 2-A, "Booting Techniques". Details on how to program your disk drives may be found in Chapter 2, Appendix 2-B, "Programming Drives." Details on programming disk drives may be found in Chapter 1, Appendix 1-B, "Configuring Hardware."

****

Setting Up A System with 5-1/4 or 8-inch Disk Drives

Assuming you have a hard-sector computer system, you should have seven distribution disks. The seven-disk set of HDOS 3.0 distribution disks contains HDOS 3.0 BOOTING DISK, EXECUTABLE DEVICE DRIVERS AND UTILITIES, DEVICE DRIVER SOURCE CODE (4 disks), and COMMON DECKS. This set includes an update to the HDOS Operating System. Console Debugger, Heath Ed Line Editor, Assembly Language, and Benton Harbor BASIC are carried forward from HDOS 2.0, except some of the chapters have slight modifications to enable them to work in the HDOS 3.0 environment. If you have a computer system with soft-sector drives primary, you will receive fewer disks.

The HDOS 3.0 BOOTING DISK contains most of the files which are essential to running programs, and it is from it that you will generate your system volumes.

The disk of EXECUTABLE DEVICE DRIVERS AND UTILITIES contains all the essential HDOS 3.0/3.02 utilities not found on the HDOS BOOTING disk. This includes some very nice utilities, patches, and MS-DOS emulating programs that make this operating system not only more flexible than HDOS 2.0, but more fun to use.

In addition to the distribution disks supplied, you will need at least two blank disks which do NOT have write-protect tabs installed. DO NOT remove the write tabs from the HDOS 3.0 BOOTING DISK and the other HDOS 3.0 distribution disks!. If you did so, it would make it too easy to trash a disk, - especially for a beginner.

****

The Differences Between 5 1/4 and 8-Inch Disks

One of the primary functions of a computer operating system is to enable the various physical parts of the computer to cooperate toward the execution of your commands. In order for this cooperation to take place, there must be communications between HDOS (software) and the physical parts of the computer (hardware). The computer cannot execute any command unless HDOS is communicating with the hardware.

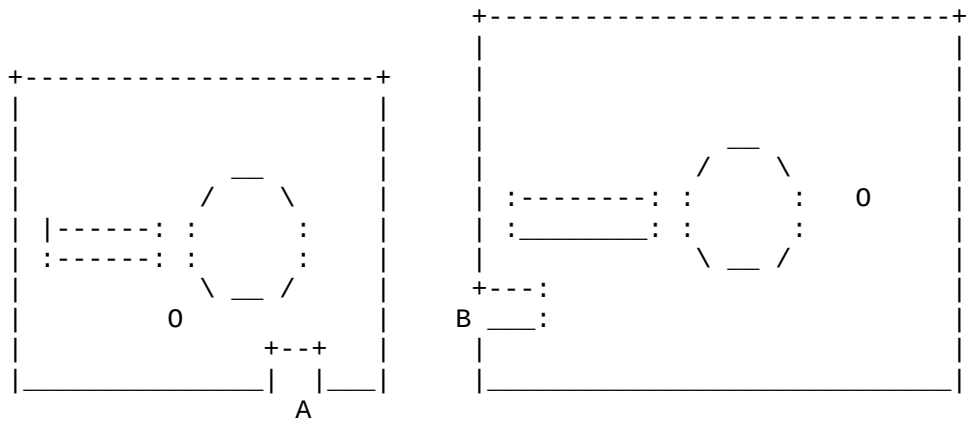
SETTING UP A SYSTEM (Cont)

The Differences Between 5 1/4 and 8-Inch Disks (Cont)

It is obvious that an H37 5 1/4-inch disk is physically smaller than an H47 8-inch disk, but the physical differences go deeper than that. Heath designed the H37 controller with a capacity of four drives, compared to the H47's capacity of two, and the H17 controller's capacity of three. The H89 computer can handle only two different types of drives; any combination of H37, H17, or H47.

Admittedly, certain non-Heath vendors had at one time designed a super controller that could handle the H37, the H47, and the H17 on one board, without sacrificing the three-port serial board. Unfortunately, these controllers are no longer available.

There is also a difference in the appearance of the H37 and H17 versus the H47 disks. To illustrate:



5 1/4-Inch Disk
H37 and H17

8-Inch Disk
H47

As illustrated, the 5 1/4-inch disk is shown on the left. Notice that the write-tab notch, "A," faces downward. Not so with the 8-inch disk, shown to the right. Notice that the write-tab notch, "B," faces forward.

The obvious physical difference in the appearance of the two disks, also displays a another difference. For example, with the H37 disk, when the write-tab notch is COVERED with a write tab, one cannot write to that disk. It is said that the disk is write-protected. With the H47, 8-inch disk, however, this situation is reversed. In this case, if the write-protect notch is covered, the disk is "WRITE-ENABLED." In order to cause the H47 disk to be write-protected, insure that the write-tab notch is left open. Operationally, there are no significant differences between the 5 1/4-inch and 8-inch drives.

****

APPENDIX 1-A: GLOSSARY

ACCESS: The act of finding a storage location in memory or on a mass-storage medium in order to read data from it or write data to it. Also see "direct access," "random access," and "sequential access."

ALLOCATION: The act of setting aside a certain amount of memory or an area of a mass-storage device to be used for running programs and/or storing data.

ALPHANUMERIC: Any string of characters containing both alphabetic and numeric characters.

ASCII: Abbreviation for American Standard Code for Information Interchange, a standard code used to store alphanumeric data.

BACKUP: A duplicate of a program or file stored on a separate disk or other storage medium, such as magnetic tape, removable winchester cartridge, etc. This is a "fail-safe" method to insure against loss of data.

BIT: A binary digit expressed as either a "0" or a "1." Bits are the smallest data unit used in a computer, and are combined into blocks. A block of 4 bits is called a "nibble," or sometimes "nybble." A block of 8 bits is called either a "byte" or an "8-bit word."

BLOCK: See "cluster."

BOOTSTRAP: The program or process by means of which communication is established between hardware and software. In order for the computer to "run," it must contain a program. In order to load programs into a computer, the computer must be running. In other words, the system must "lift itself by its bootstraps" before it can operate. Early computer systems were started, or "booted-up," by manually switching a series of binary instructions from the front panel. Nowadays, most computer systems have bootstrap programs already loaded into read-only memories, or ROMs. The bootstrap program enables the computer to run whenever the power is turned on. Bootstrap, or "boot," can be used to describe the process of transferring from a basic start-up program to a more sophisticated program, such as an operating system.

BUFFER: An area of user or system RAM which is set aside for communication with peripherals, including the disk drives. The HDOS disk buffer consists of 256 bytes of memory, which is the same size as a disk sector. When accessing a file, the operating system reads a sector into the buffer so that a program can gain access to

APPENDIX 1-A: GLOSSARY (Cont)

BUFFER: (Cont) the data. Buffers vary in size, depending upon the efficiency of the peripheral with which they are associated and the amount of available memory. A buffer for a terminal might consist of only one byte, while a disk buffer should be at least as large as the minumum unit of storage on the disk, 254 bytes.

CARRIAGE RETURN: This expression, often used in computer manuals, has been replaced by "RETURN." Pressing the "RETURN" key will send the cursor down one line and to column 1 on the left margin. <CR> is an expression originally used during the era of the typewriter.

CATALOG: A command that instructs the operating system to print a display of useful information about a set of files, such as filenames, sizes, and dates of creation or alteration.

CHARACTER SET: The characters which may be displayed or used for processing on a specific computer, printer, or other peripheral. Unique character sets may be utilized on the H89 by installing certain non-Heath TLB board ROMs, such as the SuperSet by TMSI.

CLOSE: A command that indicates to the operating system that a process no longer requires access to the data in a file. If the file was changed during the execution of the program, the disk storage area utilized for the file may be updated. The directory will also be updated to reflect the changes to the file, such as its size and location on the disk media.

CLUSTER: A contiguous portion of storage area on the disk medium. In the case of HDOS, the minimum cluster size is two sectors of 256 bytes each.

COMMAND: Information communicated to the operating system which instructs the system to perform some action, such as deleting a file.

CONSOLE: Another word for the peripheral from which a computer system is controlled. An "operator" or "user" communicates with the operating system by means of a console or terminal.

CONTIGUOUS: Describes objects or storage areas that are located next to each other. Similar to "continuous."

APPENDIX 1-A:GLOSSARY (Cont)

COPY: The act of placing the contents of one file from one peripheral device into another. The data contained in the two files is then identical; however, the names and physical locations of the files are different.

CPU: The computer's "Central Processing Unit." It is the "brains" of the system and controls all operations. It refers to the microprocessor, but also applies to the circuit board, on which the CPU is mounted. The H89's CPU is a Z80, while the H8 uses an 8080 CPU.

CRC: Acronym for "Cyclic Redundancy Check." This is an operation that measures the size of either binary or ASCII files. The result is a 5-digit expression that is used to verify that a copy of a file is identical to the original. Example: 52453.

CREATE: The act of setting up a new file, giving it a name for future reference. The operating system will find space for the file on the disk if sufficient space is available. It will also update the directory to indicate the presence of the new file, unique among the files on a particular disk.

DEFAULT: A condition that exists when no action is taken to override it. For instance, a device driver may print lines which are 80 characters in length unless it is instructed to make the lines shorter or longer. The default line length would then be 80 characters.

DELETE: A command that instructs the operating system to remove a file from the directory, and to free the area on the disk that it occupies, making the space available for other purposes.

DESTINATION: A file or peripheral device, to which data is to be written. For example, TT: or LP:.

DEVICE: A peripheral to which data is to be written, or from which data is to be read, by means of input/output commands or instructions.

DEVICE DRIVER: An operating system program that controls a peripheral, such as a disk drive, terminal console, or printer. See "device independence."

DEVICE INDEPENDENCE: A feature that allows a user program to refer to a peripheral by a symbolic name, as if it were a file, instead of requiring a section of the program to be written specifically for the purpose of controlling the peripheral. Thus, a program can command the operating system to input data to the named device or output data

APPENDIX 1-A: GLOSSARY (Cont)

DEVICE INDEPENDENCE: (Cont) from it. The operating system, in turn, uses a device driver which is associated with the device name in order to accomplish the I/O.

DIAGNOSTIC: A program used to troubleshoot a computer system, or the various components of a computer system. The most common "diagnostics" are programs that are used to find possible read/write errors in memory devices.

DIRECT ACCESS: A concept used with some disk systems to describe the ablity to access a given block of data by using the directory to find its physical position on the disk. This eliminates the need to read all the data that precedes the desired block as a means of finding it. The term "random access" is sometimes used to describe this capability.

DIRECTORY: A data area used by the operating system that holds the location and size of each disk file, referenced by its name. It is similar to a city telephone directory, but with filenames instead of people's names and addresses.

DRIVER: A nickname for "device driver."

EXTENSION: The portion of a filename that distinguishes it from another file with the same name. For instance, an assembly language program that is used to compute poker odds could be called "POKER.ASM," while the assembled machine-language instructions for the program could be stored on a file called "POKER.ABS." The extension is a portion of the filename that is located to the right of the period. Under HDOS, it may consist of zero to three characters.

FGN: The first group number. This file relates to the map.

FWA: The first work space address. This file relates to the memory map.

FILE: A data structure that is generally associated with a disk or other direct-access device. The disk is analogous to an office filing cabinet, with the files corresponding to the folders of information on the magnetic recording medium of the disk. Data is read from files and written to files by means of operating system commands which reference each file by a unique filename. The system handles the problems of finding the data and making it available to a process. Files must be "open" to be accessed and must be "closed" when no longer needed.

APPENDIX 1-A: GLOSSARY (Cont)

FREE: The act of making an area of memory available for other purposes. For example, when a file is closed, its buffer is "freed." Also, when you CAT a disk, the final summary tells how many sectors are used and how many sectors are "free" to be used.

H17: A disk drive connected to the hard-sector controller. Also, a reference to the hard-sector controller.

H37: A disk drive connected to the soft-sector controller. Also, a reference to the soft-sector controller.

H47: A disk drive connected to the soft-sector controller. NOTE: The standard Heath configuration permits only two of the above units to operate at any given time.

H67: A hard-drive wired into the computer system. This applies to a specific Heath winchester product.

HANDLER: See "device driver."

HARD ERROR: A disk read/write error caused by a malfunction in the electronic or electromechanical hardware which does not go away when successive attempts to read or write are made. A hard error is usually the result of an error in writing caused by dust, static electricity, a scratched disk, or by various kinds of electronic interference or noise from electric motors, radio transmitters, and so on.

INITIALIZE: A command to the operating system that instructs it to prepare a floppy-disk for data storage. A new floppy disk must be initialized before you can use it. If the floppy disk already contains data, that data will be destroyed if that volume is initialized.

INITIALIZATION: The process of initializing a disk.

I/O: Abbreviation for input/output.

INTERRUPT: A hardware signal to the computer, used extensively by operating systems, that causes the current process to temporarily cease, and another to take its place. This facility speeds up the operation and handling of peripherals. The interrupt routine is similar to a subroutine in that it eventually returns control to the original process. The difference is that an interrupt may occur at almost any time, and is controlled by external events, such as a keystroke at the terminal.

APPENDIX 1-A: GLOSSARY (Cont)

LGN: The last group number. Applies to the memory map.

LSI: Last sector index. Applies to the memory map.

LWA: The last work space address. Applies to the memory map.

LIBRARY: A collection of programs that may be used in conjunction with each other. For example, an operating system can be a library of separate programs that are capable of calling one another. Also, refer to "Archive" in Chapter 7.

LOAD: The process of transferring data from a peripheral into RAM.

LOADER: A program that transfers data from a peripheral into RAM.

MAP: A picture of how data and programs are distributed in memory, or a table which shows where files are located on a mass-storage device.

MEDIUM: Generally a magnetic substance, such a floppy disk, on the surface of which data can be recorded. Media can usually be removed and replaced by other physically similar media.

OPEN: A command to the operating system that makes the contents of a specific file available to a process.

OPERATING SYSTEM: A complicated set of programs that is generally associated with disks and other mass storage devices. Its function is analogous to that of a policeman directing traffic at a busy intersection. Specifically, it may keep track of large amounts of data on disk files, control peripherals, control the distribution of memory among various programs, regulate the execution of programs, keep track of the amount of time and memory that are used for various purposes, and even improve its own speed and efficiency. The degree of sophistication is generally directly related to the size and cost of the computer system. For example, HDOS is an acronym for Heath Disk Operating System.

OVERHEAD: That portion of the computer system's time, memory, and storage required to implement the functions of the system, and is thus not available to the user.

APPENDIX 1-A: GLOSSARY (Cont)

OVERLAY: An overlay is a program that is kept on disk and swapped in and out of the memory automatically when needed. Overlays save buffer space and reduce the size of the binary program. For example, EDIT19 uses eight overlays.

PRIMARY MEMORY: The high-speed RAM in which programs are executed and in which data is stored so as to be more immediately accessible.

PROMPT, MONITOR ROM: The expression that you see on the screen when you turn on AC power. The monitor ROM prompt that Heath provides is: "H:". Non-Heath monitor ROMs provide a different prompt.

PROMPT, SYSTEM: The unaltered HDOS system prompt is ">." In HDOS 3.0 the prompt may be altered by typing a command into the file, "AUTOEXEC.BAT," or by the prompt command.

PROTECTION: The means by which any of the various processes of the operating system are prevented from over-writing an important area of memory or disk space.

RAM: An acronym for "Random Access Memory". RAM allows any given memory location to be read from or written to in the same amount of time as any other equivalent location, regardless of physical position.

READ: The act of examining the contents of a memory location, or the process of transferring the contents of a file into a buffer area of RAM.

REAL-TIME CLOCK: An electronic counter that interrupts the processor at given time intervals. The stock H89 and H8 have a real-time clock which generates interrupts at intervals of two milliseconds.

RENAME: A command that changes the name of a file without affecting its contents or physical location.

RESOURCE: A valuable portion of a computer system, such as a peripheral, a portion of memory, or a program. Resources can be shared by several processes in advanced systems. In any case, they are reusable and relatively permanent.

ROM: An acronym for "Read-Only Memory." A ROM is a memory chip whose contents cannot be changed.

ROTATIONAL LATENCY: The time required for the desired sector of a disk to rotate under the disk drive head.

APPENDIX 1-A: GLOSSARY (Cont)

SECONDARY MEMORY: Generally, a large-volume, low-cost, and relatively slow memory device. It can be a peripheral such as a disk or a tape storage unit.

SECTOR: The minimum accessible unit of storage on a disk. The size may be determined by physical or logical parameters, or both. In the case of the disk and HDOS, the sector size is 256 bytes; while the minimum file size is one cluster, or at least two sectors.

SEEK: The action taken by a disk drive head in finding the correct track when data is read from a file or written to a file. "Seek Time" partially determines the speed along with "rotational latency."

SEQUENTIAL ACCESS: A type of I/O in which a unit of storage can be made available for reading or writing only by reading every unit of storage which precedes it on the recording medium. This may result from the physical characteristics of the storage device, such as a magnetic tape, or it may be a limitation imposed by the operating system.

SOFT ERROR: An error in reading a disk or other storage device that may be caused by dust, noise, or an interrupt at the incorrect time. It is similar to the "hard error" except that a soft error may be corrected by an attempt to repeat the failed process. If several retries do not correct the problem, the error is reclassified as a hard error.

SOURCE: In the case of operating system commands, the source is the original file, which is to be renamed or copied. In the case of programs, the source is the highest level code which is converted by the compiler, interpreter, or assembler into machine-executable instructions, or "object code."

STRING: A connected sequence of characters, words, or symbols. To initiate a program, the computer requires the user to type a command string (i.e., EDIT19 SY1:MICRO.DOC).

SWAP: The act of removing the contents of a memory area temporarily while the memory is used for other purposes. Also see "overlay."

SWITCH: A symbolic code that is used to issue a command to the operating system. Also a variable that is interpreted by a process in order to influence its flow-of-control.

APPENDIX 1-A: GLOSSARY (Cont)

SYNTAX: The formal or "rigid" order in which commands or instructions must be written to enable the operating system or other software process to recognize them, prior to performing an instruction.

TLB: Terminal Logic Board located behind the CPU board in the H89 Computer.

TPI: Tracks Per Inch. Used to describe the size of a disk or disk drive. For example: a double-sided 96 tpi disk contains 1600 sectors, while a double-sided 48 tpi disk contains 800 sectors of storage space.

TRACK: A circular area on a disk that consists of a given number of sectors. In the case of standard H17 5-1/4 inch disks, HDOS allocates 40 tracks per disk, with each track composed of 10 sectors. In the case of 8-inch H47 disks, HDOS allocates 77 tracks per disk. The tracks on a single density 8-inch disk are subdivided into 13 sectors. The tracks on a double density 8-inch disk are subdivided into 26 sectors. In the case of a standard 5-1/4 inch H37 disk, HDOS allocates either 40 or 80 tracks (depending upon whether you have a 40 or an 80 track disk drive), with each track composed of 16 sectors in the double-density recording mode. With the H37 and H47 disk systems, the effective number of tracks may be doubled by reading and/or writing to both sides to the disk.

USERFWA: The program area of the memory map.

UTILITY: A program which is called upon by either the user or the operating system in order to perform a function,or a group of functions. Examples of HDOS system utilities are PIP, EDIT, INIT, SYSGEN, ONECOPY, etc.

VOLUME: An interchangeable storage unit, such as a cassette tape or a floppy disk. The volume contains data and is placed in a "drive" so that data may be "accessed." A volume may also refer to the number assigned to a disk during the INITILIZATION process.

WRITE: The act of transferring data into a memory location or register, or outputting it to a disk or peripheral. The head on a disk writes binary information onto the magnetic medium, which is the physical location of a given file.

APPENDIX 1-B: CONFIGURING HARDWARE

Table of Contents

APPENDIX 1-B: CONFIGURING HARDWARE (Cont)

INTRODUCTION:

The source for this data is mainly the Heath Publication, H-88/H-89/Z-89/Z-90 Configuration Guide, Part Number 597-2571. The source for pages 32 through 34 is the Heath Double Density Controller, Model Z-89-37.

In order to make its computer systems as flexible and as useful as possible, the Heath Company and Zenith Data Systems have developed several configurations of the H88, H89, Z89, and Z90 series of computers. This application note has been prepared to assist users and service personnel in selecting and verifying the proper configuration for their desired application.

FIRMWARE

MEMORY DECODE ROM

The Memory Decode ROM is located at U517 on the CPU Board. Two ROMs have been used. Part number 444-42 was originally used. This ROM precluded the use of more than 48K of memory or CP/M. It has been superseded in all production units by 444-66 which allows the ROM based 48K mode, the ROM based 56K mode, and an all RAM 64K mode. ALL users should upgrade to this part regardless of configuration. There are NO negative consequences connected with this upgrade.

Associated with this ROM are three or four jumpers, JJ501 thru JJ504. Older CPU boards have all four jumpers; they should be set as follows:

When using the old ROM (444-42)

	JJ501	JJ502	JJ503	JJ504
16K	0	0	0	0 (or B)
32K	1	0	0	0 (or B)
48K	0	1	0	0 (or B)

When using the new ROM (444-66)

16K	0	0	**	0 (or B)
32K	1	0	**	0 (or B)
48K	0	1	**	0 (or B)
64K*	1	1	**	0 (or B)

* Requires the WH-88-16 accessory PC board.

APPENDIX 1-B: CONFIGURING HARDWARE (Cont)

** A jumper is required between the center pin of JJ503 and pin 17 of P509, or P4 of WH-88-16 (which connects to pin 17 of P509). This jumper may have been soldered on the back of the CPU board during manufacture (for 134-Z-89-FA and some other models), or it may be ordered as part number 1120, and installed by the user. Neither tools nor soldering are required.

Newer CPU boards (which only have three jumpers, JJ501 through JJ503) are supplied with the new decode ROM (444-66) already installed and the jumper wire incorporated directly into the PC board foil. These boards should not be used with the old ROM (444-42). The jumpers should be set as follows:

	JJ501	JJ502	JJ503
16K	0	0	0 (or B)
32K	1	0	0 (or B)
48K	0	1	0 (or B)
64K	1	1	0 (or B)

I/O DECODE ROM

The I/O decode ROM is located at U550 on the CPU board. Two parts are available: 444-43 and 444-61.

Part number 444-43 supports the hard-sector single-density 5" disk system (H-88-1), the three-port serial I/0 card (HA-88-3), and cassette tape (H-88-5) in the expansion area.

Part number 444-61 supports two disk devices and the three-port serial I/O card, but does not support cassette tape.

Users with cassette tape must use 444-43. Users who have only the serial I/O accessory and a 5 1/4-inch single density hard sectored disk system may use either part; other users should use part number 444-61.

CODE ROM

The code ROM is located at U518. Three ROMs are available:

Name	Part Number	Manual Part No.
MTR-88	444-40	595-2349
MTR-89	444-62	595-2508
MTR-90	444-84	595-2696
MTR-90 (newest)	444-142C	595-2696

Each of these is normally supplied with full source code and a user manual.

APPENDIX 1-B: CONFIGURING HARDWARE (Cont)

MTR-88 is used with cassette tape and the H-88-1 hard-sector single density 5 1/4-inch disk system. It cannot be used with other types of disk systems.

MTR-89 supports both the 5 1/4-inch hard sector single density disk and the H/Z-47 dual 8-inch floppy disk. Cassette tape is not supported.

MTR-90 is a general purpose part which supports all disk mass storage devices (H-77, Z-87, H/Z-47, Z-37, Z-67). Cassette tape, however, in NOT supported. This part is supplied with Z-89-37 and Z-89-67. Because this is a 4K part while both MTR-88 and MTR-89 are 2K parts, the secondary address decoder must be changed to use this part.

The secondary address decoder is located at U516.

Two parts are available, 444-41 and 444-83.

Part No. 444-41 is used with MTR-88 and MTR-89.
Part No. 444-83 is used with MTR-90.

There are four jumpers wires associated with the code ROM and the secondary address decoder. These are either JJ505, JJ506, JJ507 and JJ508 (on older units) or JJ504, JJ505, JJ506 and JJ507 (on newer units). These should be set as follows:

Older Units:	JJ505	JJ506	JJ507	JJ508
Newer Units:	JJ504	JJ505	JJ506	JJ507
MTR-88, MTR-89	0	0	0	1 (or B)
MTR-90	1	*	1	1 (or B)

* A jumper should be installed between the center pin of JJ506 (or JJ505) and pin 14 of P508 when you are using as MTR-90. Part number 134-1159 may be used. Neither tools nor soldering are required.

OTHER CONFIGURATION ITEMS

Unless use is confined to cassette tape, the following parts should be installed:

Part number 444-19, the HDOS ROM, at U520. Two 2114 1Kx4 RAMs (part Number 443-764) at U523 and U525.

A 78H12 (442-650) on the power supply at U103 (only required if an internal drive is installed).

In order to use the WH-88-16 memory expansion, it is necessary to change U562 from a 74LS132 (443-792) to a 74S132 (443-901).

APPENDIX 1-B: CONFIGURING HARDWARE (Cont)

OTHER CONFIGURATION ITEMS (Cont)

Final production units use a 78H05SC or MC78TO5 (442-651) at U101 instead of an LM309K (442-30). In addition, U101, U102 and U103 are mounted with heat sinks (215-658) and thermal compound (352-31). These changes improve the power output and heat dissipation capacity of the power supply. They are required and included with the Z-89-37 disk controller kit, and may be incorporated into any unit if power supply and/or heat problems are encountered.

DIP SWITCH SETTINGS

DIP switch SW501 is used to program the initial power-up configuration. Its settings depend on and vary with the monitor ROM used.

SETTING SW501 WITH MTR-88

Only the three most significant bits are used, switch sections 5, 6 and 7.

Sections 6 and 7 select the power up baud rate used for communications with the terminal (which is normally the internal H-19 terminal logic board). The four options are:

Section 7	Section 6	Baud Rate
0	0	9,600
0	1	19,200
1	0	38,400
1	1	57,600

The selected baud rate must match the baud rate set a S401 on the terminal logic board. The standard terminal logic board firmware only supports 9,600 baud at this time (19,200 can be selected and used, but characters will occasionally be lost). Therefore, both sections 6 and 7 will normally be set to zero. Reliable higher terminal baud rates are possible by installing TMSI's SuperSet on the TLB board.

You can set switch section 5 to force a memory test on reset or power up. To force the test, set the switch to "0". Since the test will not stop until the switch is reset, you must set the switch to "1" before you can use the computer for normal operation.

APPENDIX 1-B: CONFIGURING HARDWARE (Cont)

SETTING SW501 WITH MTR-89

The settings of SW501 for use with MTR-89 are defined as follows:

Switch	Setting	Description
1,0	00*	Port 174(7CH)/177Q(7FH) has an H-88-1 controlled disk (normal).
	01	Port 174/177Q has an H/Z-47 type disk.
	10	Undefined.
	11	Undefined.
3,2	00**	Port 170(78H)/173Q(7BH) is not in use (normal without H/Z-47).
	01	Port 170/173Q has an H/Z-47 (normal with H-47).
	10	Undefined.
	11	Undefined.
4	0	Boots from device at port 174/177Q (H-88-1 normal).
	1	Boots from device at port 170/173Q (H/Z-47).
5	0	Performs memory test upon boot up (not currently supported).
	1	Does not perform memory test (normal).
6	0	Sets Console to 9600 baud (normal).
	1	Sets Console to 19,200 baud (not currently supported). By adding TMSI's SuperSet baud rates higher than 19,200 may be achieved reliably.
7	0	Normal boot (normal).
	1	Auto boot on power up or reset (not recommended).

* Right column is switch 0.
** Right column is switch 2.

SETTING SW501 WITH MTR-90

The settings of SW501 for use with MTR-90 are the same as those for use with MTR-89 except that positions 0, 1, 2 and 3 are redefined as follows:

APPENDIX 1-B: CONFIGURING HARDWARE (Cont)

Switch	Setting	Description
	00*	Port 174(7CH)/177Q(7FH) is H-88-1 controlled disk.
1,0	01	Port 174(7CH)/177Q(7FH) is H/Z-47 disk.
	10	Port 174(7CH)/177Q(7FH) is Z-67 disk.
	11	Undefined.
	00**	Port 170(78H)/173Q(7BH) is Z-89-37 controlled disk.
3,2	01	Port 170(78H)/173Q(7BH) is H/Z-47 disk.
	10	Port 170(78H)/174Q(7BH) is Z-67 disk.
	11	Undefined.

* Right column is switch 0.
** Right column is switch 2.

USING THE EXPANSION SLOTS

The H-88-1 hard sectored disk controller should be installed at P506/P512, the right connector in the right expansion area.

The H-88-3, HA-88-3 and Z-89-11 serial I/O boards should be installed at P505/P511, the center connector in the right expansion area. There never was support for the Z-89-11 parallel card.

The Z-89-37 soft sectored double-density disk controller and the H-88-5 cassette I/O card should be installed at P504/P510, the left connector in the right expansion area.

Z-89-47 and Z-89-67 interface boards may be installed in either the right or left positions in the right expansion area (P506/P512 or P504/P510). However, they must be jumpered differently, depending on which of these positions are actually used. See the appropriate manual supplied with the interface. If both a Z-89-47 and a Z-89-67 board are used (together), the Z-89-67 should be installed at P506/P512.

Additional boards may be installed in conjunction with bus expansion devices such as those available from FBE Research, Mako Data Products, and Kres Engineering.

The WH-88-16 memory expansion should be installed at P503/P509, the right expansion slot in the left expansion area.

APPENDIX 1-B: CONFIGURING HARDWARE (Cont)

WRITE PROTECT PULLUP

A 4700 Ohm (6-472) pullup resistor is required between pins 1 and 12 of P512. This resistor is provided on the hard sectored disk controller H-88-1) usually installed at P512. It is also included on Z-89-67, with a jumper connector to enable or disable it, depending on whether the Z-89-67 is installed at P506/P512 or P504/P510 (the pullup must be disabled if P504/P510 is used). Customers desiring to use a Z-89-47 at P506/P512 (either by itself or in conjunction with a Z-89-37 interface installed at P504/P510) will have to install this resistor between pins 1 and 12 of P512 on the Z-89-47 interface board. Customers who have no interface installed at P506/P512 should install part number 100-1816 directly on P512. This part is supplied with Z-89-37.

DRIVE PROGRAMMING

Drive programming is illustrated in Pictorials 1, and 2 on pages 1-31 and 1-32, respectively. There were some early H-17-1 drives which are different than those shown. These are covered in the H-17 manuals, but they have not been used for several years.

The use of H-17-4 drives with the hard sectored controller (giving 400K bytes of storage) is not supported at this time and is not expected to be supported in the future.

The use of H-17-4 drives INTERNALLY within the H/Z-89 is not supported at this time, but is expected to be supported in early 1982. The track width and signal levels coming off the read head are substantially less than those found in the H-17-1, and the reduced signal to noise ratio which results prevents reliable operation on the inner tracks in a significant minority of units. Improved shielding for the internal drive is under development and will eliminate the problem.

H-17-4 drives are available from Quikdata, Inc., 2618 Penn Circle, Sheboygan, WI 53081-4250.

MEDIA NOTES

H-17-4 drives should ONLY be used with media certified for 96 or 100 TPI service; double density recording should only be done on diskettes which are certified for such use.

We do not recommend the use of "flippy" diskettes. The liner used to trap oxide, dust, and other contaminates inside the diskette jacket has a nap (grain) to it, and reversal of diskette rotation direction (as occurs when a diskette is turned over) can release a large portion of the trapped contaminates, with undersirable results.

APPENDIX 1-B: CONFIGURING HARDWARE (Cont)

MEDIA NOTES (Cont)

We specifically recommend against using head cleaning diskettes; and besides, it is not necessary. It is unlikely that disk heads will EVER require cleaning.

H-89 DISK CONFIGURATIONS

System Components	Single Density Hard Sector	Double Density Soft Sector	Total Capacity
HS-89 or Z-89-81	One 100K internal drive	----	100K
Z-89-80 Z-87	Two 100K external drive(s)		200K
HS-89 HS-77	One 100K internal drive & one or two 100K external drives	----	200K or 300K
Z-89-81	One 100K internal drive & two 100K external drives	----	300K
Z-90-80	----	Two 160K external drives	320K
Z-90-80	----	Two 640K external drives	1.28M
HS-89 or Z-90-82	----	One 160K internal drive	160K
HS-89 Z-89-37 HS-77	----	One 160K internal drive & one or two 160K external drives	320K or 480K
HS-89 Z-89-37	One 100K internal drive	Two 640K external drives	1.4M
Z-90-82 Z-87	----	One 160K internal drive & two external drives	480K

APPENDIX 1-B: CONFIGURING HARDWARE (Cont)

DISK DRIVE DIP SHUNT PROGRAMMING PLUG CONFIGURATIONS

**** HARD SECTOR H-17/H-88-1 CONTROLLER ****

Wangco - Siemens 82

	HW U0	HW U1	HW U2
HS	o---o	o---o	o---o
DS1	o o	o o	o---o
DS2	o o	o---o	o o
DS3	o---o	o o	o o
MX	o o	o o	o o
BLANK	o---o	o---o	o---o
HM	o o	o o	o o

Tandon TM-100-4

	HW U0	HW U1	HW U2	
HS	o o	o o	o o	HS must be open when
NDS0	o o	o o	o---o	using HUG driver.
NDS1	o o	o---o	o o	
NDS2	o---o	o o	o o	
NDS3	o o	o o	o o	
MX	o o	o o	o o	
SPARE	o---o	o---o	o---o	SPARE can be either
HM	o o	o o	o o	open or closed

Pictorial 1 - H17 Drives

APPENDIX 1-B: CONFIGURING HARDWARE (Cont)

DISK DRIVE DIP SHUNT PROGRAMMING PLUG CONFIGURATIONS (Cont)

**** SOFT SECTOR Z-89-37 CONTROLLER ****

Wangco - Siemens 82

	HW U0	HW U1	HW U2
HS	o---o	o---o	o---o
DS1	o---o	o o	o o
DS2	o o	o---o	o o
DS3	o o	o o	o---o
MX	o o	o o	o o
BLANK	o---o	o---o	o---o
HM	o o	o o	o o

Tandon TM-100-4

	HW U0	HW U1	HW U2	HW U3
HS	o---o	o---o	o---o	o---o
NDS0	o---o	o o	o o	o o
NDS1	o o	o---o	o o	o o
NDS2	o o	o o	o---o	o o
NDS3	o o	o o	o o	o---o
MX	o o	o o	o o	o o
SPARE	o o	o o	o o	o o
HM	o o	o o	o o	o o

Hardware Unit 3 not supported by Heath.

Pictorial 2 - H37 Soft Sector Drives

Note: HW Ux = HardWare Unit number, i.e., SY0:, DK0:, etc.

** INSTALLING THE H37 CONTROLLER **

The card connector positioned on the top of the card, P3, is designed for an internal drive. It may or may not be connected. If it is not connected, there is no effect on the card performance.

The card connector positioned on the bottom of the card, P4, is designed for external drives. It is to be cabled to the rear panel of the H89. Then external drives may be connected from the outside.

Although the H37 card will support 4 drives physically, the operating system or the software may not.

J4 - Jumper J4 in order to connect a drive which has been mounted inside the computer. The drive requires a terminating resistor. The drive is usually connected using a short 34-pin flat cable.

APPENDIX 1-B: CONFIGURING HARDWARE (Cont)

INSTALLING THE H37 CONTROLLER (Cont)

00 - Do not jumper any pins if you want to attach 3 drives to the rear panel connector. The last drive in the chain must have a terminating resistor installed.

REPLACING THE KEY CHIPS

In order to make the H37 card work, the following chips on the CPU board must be replaced:

CHIP REFERENCE DESIGNATION	OLD PART	NEW PART
U516	8013Y (444-41)	74S188C (444-83)
U518	444-62	2732 (444-142) [MTR-90]
U550	444-61	444-61 (Not the improved part)
U558	443-754(Save) (A modified 74LS240)	100-1817 (Special ROM Part) (A modified 74LS240)
U557	Remove (Save)	Cable plugs in here. Unit consists of a 20-pin socket with a 4700 ohm resistor connected between pins 1 and 8.

POWER SUPPLY UPGRADE

In addition, the power supply must be upgraded in order to handle the increased current demand. The following parts are required:

CHIP REFERENCE DESIGNATION	QUANTITY	PART NUMBER	DESCRIPTION
N/A	3-Each	215-658	Heatsink, HD
U101	1-Each	442-651 (UF78H05SC or MC78T05)	5-volt regulator

****

APPENDIX 1-C: PORT ASSIGNMENTS

I/O PORT ADDRESS MAP

OCT	HEX	DEC	H-8 Usage	H-89/90 Usage	Super 89 Usage
377	FF	255	Reserved	Reserved	Reserved
376	FE	254	Reserved	Reserved	Reserved
375	FD	253	AT: on H8-5 Card	Reserved	Reserved
374	FC	252	Alt. Terminal	Reserved	Reserved
373	FB	251	TT: on H8-5 Card	Reserved	Reserved
372	FA	250	Console Terminal	Reserved (NMI)	Reserved
371	F9	249	Cassette on H8-5	Cassette on H88-5	Reserved
370	F8	248	Interface Card	Interface Card	Reserved
			Currently Unassigned		
362	F2	242	Reserved	General Purpose	Same as H-89
361	F1	241	H-8 Front Panel	Reserved	Reserved
360	F0	240	Switches & LED's	Reserved (NMI)	Reserved
357	EF	239	TT: on H8-4 Card	TT: Native Card	Same as H-89
350	E8	232	Console Terminal	Console Terminal	Same as H-89
347	E7	231	LP: on H8-4 Card	LP: on H88-3 Card	Same as H-89
340	E0	224	Line Printer	Line Printer	Same as H-89
337	DF	223	DTE on H8-4 Card	DTE on H88-3 Card	Same as H-89
330	D8	216	Modem	Modem	Same as H-89
327	D7	215	AT: on H8-4 Card	AT: on H88-3 Card	Same as H-89
320	D0	208	Alt. Terminal	Alt. Terminal	Same as H-89
			Currently Unassigned		
307	C7	199	Reserved	Reserved	Bank Select
300	C0	192	Reserved	Reserved	
267	B7	183	Reserved	Reserved	Extra Serial
260	B0	176	Reserved	Reserved	Port OnBoard
217	8F	143	Reserved	Reserved	Real-Time Clock
200	80	128	Reserved	Reserved	Lower 4 Bits
177	7F	127	Primary Device	Primary Device	Same as H-89
174	7C	124	H-17 Disk Drive	H-17 (H-47)	Same as H-89
173	7B	123	Secondary Device	Secondary Device	Same as H-89
170	78	120	H-47 (H-37)	H-47 (H-37)	Same as H-89

APPENDIX 1-C: PORT ASSIGNMENTS (Cont)

Currently Unassigned

077	3F	63	For Your User Applications	For Your User Applications	Same as H-89
000	00	0			Same as H-89

** IO PORTS **

```
000.360  00F0   IP.PAD   EQU    360Q    PAD INPUT PORT
000.360  00F0   OP.CTL   EQU    360Q    CONTROL OUTPUT PORT

    **    Front Panel Control Bits
    *     CB.* set in OP.CTL

000.020  0010   CB.SSI   EQU    00010000B   SINGLE STEP INTERRUPT
000.040  0020   CB.MTL   EQU    00100000B   MONITOR LIGHT
000.100  0040   CB.CLI   EQU    01000000B   CLOCK INTERRUPT ENABLE
000.200  0080   CB.SPK   EQU    10000000B   SPEAKER ENABLE

000.360  00F0   OP.DIG   EQU    360Q        DIGIT SELECT OUTPUT PORT
000.361  00F1   OP.SEG   EQU    361Q        SEGMENT SELECT OUTPUT PORT
000.362  00F2   IP.CON   EQU    362Q        H-88/H-89/HA-8-8
                                             Configuration

    **    Configuration Flags
    *     These bits are read in IP.CON

000.003  0003   CN.174M  EQU    00000011B   Port 174Q Device-Type Mask
000.014  000C   CN.170M  EQU    00001100B   Port 170Q Device-Type Mask
000.020  0010   CN.PRI   EQU    00010000B   Primary/Secondary:
    *       1=>primary == 170Q
000.040  0020   CN.MEM   EQU    00100000B   Memory Test/Normal Switch:
    *       0=>Test; 1=>Normal
000.100  0040   CN.BAU   EQU    01000000B   Baud Rate:  0=>9600;
                                             1=>19,200
000.200  0080   CN.ABO   EQU    10000000B   Auto-Boot:  1=>Auto-Boot


    *     These values valid ONLY in CN.174M

000.000  0000   CND.H17  EQU    00B         H-17 Disk
000.001  0001   CND.H47  EQU    01B         H-47 Disk
000.002  0002   CND.H67  EQU    10B         H-67 Disk
000.003  0003   CND.NDI  EQU    11B         No Device Installed
```

APPENDIX 1-C: PORT ASSIGNMENTS (Cont)

```
    *     These values valid ONLY in CN.170M

000.000  0000   CND.H37  EQU    00B         H-37 Disk
000.001  0001   CND.H47  EQU    01B         H-47 Disk
000.002  0002   CND.H67  EQU    10B         H-67 Disk
000.003  0003   CND.NDI  EQU    11B         No Device Installed

000.362  00F2   OP2.CTL  EQU    362Q        H-88/H-89/HA-8-8 Control
                                              Port

    *     CB2.* set in OP2.CTL

000.001  0001   CB2.SSI  EQU    00000001B   Single Step Interrupt
000.002  0002   CB2.CLI  EQU    00000010B   Clock Interrupt Enable
000.040  0020   CB2.ORG  EQU    00100000B   ORG 0 Select
000.100  0040   CB2.SID  EQU    01000000B   Side 1 Select

   **    Secondary Control Bits
   **    Monitor Mode Flags

000.000  0000   DM.MR    EQU    0           MEMORY READ
000.001  0001   DM.MW    EQU    1           MEMORY WRITE
000.002  0002   DM.RR    EQU    2           REGISTER READ
000.003  0003   DM.RW    EQU    3           REGISTER WRITE
```

HDOS SOFTWARE REFERENCE MANUAL

HEATH DISK OPERATING SYSTEM

VERSION 3.0

CHAPTER 2

GENERAL OPERATIONS

HEATH DISK OPERATING SYSTEM

SOFTWARE REFERENCE MANUAL

VERSION 3.0

 Through the years it continued to be improved by successive revisions which included 1.5, 1.6, and finally 2.0. It was entered into public domain on 19 July 1989 per letter by Jim Buszkiewicz, Managing Editor, Heath Users' Group, P.O. Box 217, Benton Harbor, MI 49022-0217 (616)982-3463. A copy of this letter is available for public inspection.

This manual is indicative of further improvements and provides for the latest revision, HDOS 3.0 and HDOS 3.02. The revision 3.0 is detailed in chapters 1, 2, and 3, while chapters 4 through 8, 13 and 14, are related to revision 3.02. Chapters 9 through 12, with minor improvements, are essentially picked up from the original HDOS 2.0 manual. Indeed, HDOS is still alive and well!

Chapter 2, General Operations, tells how to boot, initialize, and sysgen disks in a general, basic manner. It tells how to make a working disk and configure line printers. It provides information about booting techniques which include booting from drive SY1: or SY2:, instead of the normal technique of booting from SY0: (refer to appendix 2-A for details), how to program drives (refer to appendix 2-B for details), and provides a conversion chart (refer to appendix 2-C for details).

SPECIAL DISCLAIMER: The Heath Company cannot provide consultation on either the HDOS Operating System or user-developed or modified versions of Heath software products designed to operate under the HDOS Operating System. Do not refer to Heath for questions.

Instead, you are invited to direct any questions concerning the Heath Disk Operating System (HDOS) to Mr. Kirk L. Thompson, Editor "Staunch 89/8" Newsletter, P. O. Box 548, #6 West Branch Mobile Home Village, West Branch, IA 52358.

TABLE OF CONTENTS

INTRODUCTION

At this point, your computer, disk drives, and peripherals, should be connected and ready to operate.

The purpose of this chapter is to acquaint you with the procedure for generating both data disks and bootable disks (i.e., disks that contain the HDOS 3.0 operating system).

In case you have different types of drives (i.e., either H17 and H37, or H17 and H47), simply perform the procedure which concerns itself with the type of drive that you will be using as the PRIMARY boot drive.

A word about our terminology. Throughout the "System Setup Procedure," we will refer to steps such as STEP 1, Bootstrap; STEP 2, INIT; and so on. These "STEPS" refer to SECTIONS that have a title in full capital letters, underlined with the "+" key, and a heading next to that title.

For example, when you are instructed to proceed to STEP 1, Boot, look for BOOTSTRAP, STEP 1. Therefore, the term "STEP" always refers to an entire section, and NOT to an individual instruction such as:

    Press the SHIFT and RESET keys.

In the event that you are unable to complete the entire "System Setup Procedure," you can safely remove the disk and turn off the AC power after completing any of the 8 sections indicated below.

If you do not finish the entire procedure, mark where you have left off. To continue with the procedure later, boot your system disk and then reenter at the point in the instructions where you put your mark.

The following paragraphs divide the presentation into three segments, as follows:

    [1] H89 COMPUTERS WITH MULTIPLE DRIVES.

    [2] H89 COMPUTERS WITH A SINGLE DRIVE.

    [3] H8 COMPUTERS WTIH THE HEATH H19 TERMINAL.

This chapter will walk you through the eight fundamentals that will show you how to set up your computer system. The tasks that are an integral part of these fundamentals will be used continually as you use your computer, regardless of whatever components your Heath computer system contains.

INTRODUCTION (Cont)

The eight fundamental steps are as follows:

STEP 1 - Bootstrap (alias Boot)

STEP 2 - Initialization (alias Init)

STEP 3 - Sysgen

STEP 4 - Summary of Test

STEP 5 - Set

STEP 6 - Preparing a Working Disk

STEP 7 - Configuring Line Printers

STEP 8 - Power Down

STANDARD OPERATING PROCEDURE

SYMBOLS: A direct statement from the computer will be set off in quotation marks. For example:

"MEDIA CHECK? <YES>?"

Commands or options made by the operator will be set off in apostrophes. For example:

'<RTN>'

CAUTION: Do not type the quotation or apostrophe marks. Also, pay attention to the spaces, or the absence of spaces, since HDOS is very particular about spaces.

NOTE: A row of asterisks indicates a change of topic.
****

[1] H89 COMPUTERS WITH MULTIPLE DRIVES

BOOTSTRAP BOOTING A DISK STEP 1

Introduction

One of the primary functions of a computer operating system is to enable the various physical parts of the computer to cooperate to execute your commands. In order for this cooperation to take place, there must be communication between HDOS 3.0 (software) and the physical parts of the computer (hardware). The computer cannot execute any command unless HDOS is communicating with the hardware.

[1] H89 COMPUTERS WITH MULTIPLE DRIVES

BOOTSTRAP (Cont) BOOTING A DISK STEP 1

Bootstrap is a small "hidden" program stored within the hardware. It serves to establish a communications link between HDOS 3.0 and the various functional parts of the computer. The bootstrap procedure is so-named because, by means of this procedure, you will be causing HDOS 3.0 to "pull itself up by its bootstraps" -- that is, the responses you give in this procedure, will enable HDOS to lift itself off the disk and place itself in the computer's memory. Having been installed in memory, HDOS 3.0 may then issue instructions to and coordinate the actions of the appropriate parts of the computer in response to your commands.

This procedure will be referred to several times throughout the HDOS 3.0 manual. BE SURE TO PERFORM THE SEQUENCE EXACTLY.

CAUTION

DO NOT INSERT ANY DISK INTO THE DRIVES UNTIL AFTER AC POWER IS APPLIED TO THE COMPUTER SYSTEM!!! USE OF A POWER OUTLET BOX WITH SPIKE PROTECTION TO PLUG IN COMPUTER AND PERIPHERALS IS HIGHLY RECOMMENDED.

Procedure

(1) Insure that the off-line key is in the up position. This connects the keyboard to the computer.

(2) Turn on the AC power to the computer system. Now apply AC power to the computer and then the drives, if applicable. If you have an H89, you should hear two beeps from the computer. If you have an H90, some models beep twice and some models only beep once. If you are using a Heath ROM, an "H:" will appear in the upper left hand corner of your screen. However, if you are using a non-Heath ROM, such as the Kres KMR-100 ROM, the presentation will be different. This display is referred to as the "Monitor ROM" prompt. Press the right SHIFT and RESET keys similtaneously, and note how the monitor prompt reappears. The right-hand SHIFT and RESET keys pressed simultaneously will always return you to the monitor prompt.

(3) Insert the bootable system distribution disk into the drive that has been configured as "primary boot drive SY0:." Always close the door of the drive unit after you have inserted the disk.

(4) Type the letter 'B' from the keyboard (ignore the apostrophe marks). The computer will complete the statement "oot." Now the complete statement on the screen is "BOOT."

(5) Press the RETURN key. RETURN will hereafter be signified by the expression <RTN>. You should hear some soft hissing and clicking sounds from the disk drive. This is normal. You will hear such sounds whenever the disk drive unit reads from or writes to the disk. In addition, the drive light will turn on briefly.

[1] H89 COMPUTERS WITH MULTIPLE DRIVES

(6) If your H89 is equipped with the MTR-88 ROM, HDOS will now print the message:

"TYPE SPACES TO DETERMINE BAUD RATE"

If your H89 is equipped with the MTR-90 ROM, or some other late model non-Heath ROM, this statement will not appear. However, the computer will hang up until you type spaces.

NOTE

Every time you are booting a freshly-initialized and sysgened disk for the first time, you MUST hit the SPACE BAR a few times so that the system can determine the disk baud rate. The same is true if you change the computer baud rate on the TLB board to a different value. If there is no write-protect tab on the disk to be booted, this baud rate will be written onto the disk's boot track, and you will not need to type SPACES for that disk again.

(7) After you type 'B' for "BOOT," within HDOS 3.02, the computer will immediately begin the boot process.

(8) The screen will be filled with large letters, each about 3/4 inch high. The operating system title will be printed on the screen as follows:

"HDOS 3.0
ISSUE 50-07-00"

(9) HDOS 3.02 will then print:
"System Has 64k of RAM."

Exception: If you don't have the standard 64k RAM, the HDOS system will print out the amount of RAM your system actually has.

The program continues with:
"Drivers found - TT:, SY:, DK:, LP:"

Exception: Unless you are booting the HDOS 3.02 system bootable disk, the message may include different drivers. It prints all of the drivers on disk whatever they are.

(10) Then the message appears:

"Date <30-Aug-89>?"

[1] H89 COMPUTERS WITH MULTIPLE DRIVES

How to enter the date:

(A) For HDOS 3.0:

Enter today's date in the format DD-MMM-YY. DD is a two-digit day, MMM is a three-digit month, and YY is a two-digit year. Be sure to set off the date data groups with hyphens. Thus, if today's date were 30 Aug 89, you would enter: '30-AUG-89'.

(B) For HDOS 3.02:

(1) If you are booting up for the first time, or the first time in a day, month, or year, follow the standard HDOS method of providing the date, i.e. the entire date.

(2) If you are booting up during the same month for the second or higher instance, just type the current day. For example: '30<RTN>'. The remainder is understood.

(3) If you are booting up for the first time in a new month, just type the day and the month. The remainder is understood.

HDOS then halts and requests the time:

"Time: (00:00:00)?"

NOTE

This question only appears when you copy either the CLOCK.TAS or the CLOCK89.TAS file to your system disk. The file CLOCK.TAS is used with the standard H89/Z90 computer system. The file CLOCK89.TAS is only used if you have a D-G Electronics CPU board installed.

To activate the clock, you must also add the command START CLOCK to your AUTOEXEC.BAT file. Do this using an ordinary text editor such as PIE or TXTPRO. This procedure prompts the computer to print the clock question which appears during boot to systems so prepared.

If you type in the correct time, the system clock starts up. From that time on, no matter how many times you cold boot during that session, the system clock keeps up with the latest time.

[1] H89 COMPUTERS WITH MULTIPLE DRIVES

BOOTSTRAP (Cont) BOOTING A DISK STEP 1
++++++++++++++++ ++++++++++++++ ++++++

After you type your choice of either the correct time or just a RETURN, the system continues its boot procedure until the system disk is mounted.

(11) The disk comes up to boot:

"Volume 01000 Mounted in SY0: (The volume number may vary.)
Label: HDOS 3.0 - 80-Track System Disk"

The HDOS system prompt appears:
"S:"

It is possible to customize your prompt. To do so, you must type the command into the file AUTOEXEC.BAT, using a text editor such as PIE or TXTPRO. For example, if your name is BOB, type the command:

'PROMPT: BOB+><RTN>'

****

Refer to Appendix A-2: BOOTING TECHNIQUES, for additional details concerning booting from drives other than the one hardware-configured for SY0:.

Refer to Appendix A-3: PROGRAMMING DRIVES, for instructions on how to hardware-program disk drives.

****

[1] H89 COMPUTERS WITH MULTIPLE DRIVES

INIT INITIALIZING A DISK FOR THE H89 STEP 2

Introduction

INIT is an abbreviation for INITIALIZATION, a program designed to write a map on the disk which HDOS will use to locate files. It is necessary to initialize all blank disks prior to use.

There are actually two levels of mapping that are written onto the disk during INIT. The first is a low-level initialization of the disk surface. This map is for the benefit of the floppy disk controller. It lays out a pattern on the surface of the disk that indicates the position of each of the tracks and each of the sectors on each of those tracks. The data recorded on each track includes the track number, the side number, and a pattern that signals the start of each of the data sectors on the track. Each sector header also includes the sector number, as well as a block of 256 bytes of dummy data. This process is repeated until all tracks are formatted. This process enables the floppy disk controller to read the track and sector that is currently passing under the heads, instead of counting steps from track zero, or counting milliseconds from the appearance of the index hole. The second level of mapping is referred to as HIGH-LEVEL. This causes the three system files to be transferred to the disk being initialized. Among other things, the files that are transferred are: GRT.SYS, RGT.SYS, and DIRECT.SYS.

When INIT is complete, the disk that has been initialized is said to be a DATA DISK, since it contains no system files that would enable it to boot the disk. The chief advantage of a data disk is that it can store more files than a disk of the same capacity that has been INITed and then SYSGENed. This is not much of a problem for H37 disk, due to their relatively large storage space, but it is a problem for the H17 disks, due to their relatively low storage space.

INIT is a conversational program in that it asks you questions to help you to decide what you want to do. If this is your first time through INIT, you will doubtless find the questions helpful. If you are an experienced INIT user, refer to "INIT Options," Chapter 3, page 3-7.

In SYSGEN, STEP 3, you will copy the HDOS Operating System files to your SYSTEM VOLUME. Thereafter you will be able to substitute your own SYSTEM VOLUME for the DISTRIBUTION DISK, supplied in the HDOS 3.02 package, thus keeping your distribution disk safe from inadvertent error. For now you should have your bootable system distribution disk installed in SY0: and a blank disk installed in SY1:.

NOTE

Certain portions of the INIT program differ between computer systems with H37 and/or H47 drives versus computer systems with H17 drives. These data differences will be explained in the text.

[1] H89 COMPUTERS WITH MULTIPLE DRIVES

Procedure for multiple H37 Drives:
---

(1) At the HDOS prompt, type 'INIT<RTN>'.

(2) INIT will introduce and describe itself. Below this paragraph the following message will be printed:

"YES/NO)^<NO>?"

(3) Type 'YES', and the following message will print on the screen:

"Dismounting all Disks"

Drive SY2: was empty, so no message is given.

"Volume 60000, Dismounted from SY1:
Label: HDOS 3.02 - MANUAL FILES CHAPTER 1"

"Volume 10, Dismounted from SY0:
Label: HDOS 3.02 - DISTRIBUTION DISK"

"Remove the disk(s). Hit RETURN when ready:"

Remove the disk mounted in SY1:.

(4) When the query:

"Device<SY0:>?"

is displayed, type:

'SY1:<RTN>'

The system will print:

"Insert the volume you want to initialize into SY1:
Remember, any data on this volume will be destroyed."

and then:

"Hit RETURN when ready.
Ready?"

(5) Type '<RTN>'. The computer will check SY1: and report:

"The volume in the drive .....
Apparently has not been initialized before."

(Assuming that you are using a new disk.)

[1] H89 COMPUTERS WITH MULTIPLE DRIVES

"Type NO to cancel. Type YES to
erase and initialize the disk. (YES/NO)?"

(6) Type 'YES<RTN>'

(7) The computer will instruct:

"Enter a volume serial number between 0 and 65535."

(8) Type '10<RTN>'. (Just an example.)

(9) The computer will instruct:

"Enter a volume label of 60 characters or less."

(10) Type 'SYSTEM VOLUME<RTN>'.

"Enter BOOT step rate: (30,20,12,6)<30>."

(11) For a double-sided drive connected to the H37 controller enter: '6<RTN>'.

Most double-sided drives will operate at this step time. In case you notice read/write errors on a specific drive, you may decide to initialize future disks at a steptime of 12.

The computer stamps the boot step rate you selected onto the disk, and the following message is printed on the screen:

"Double density? <YES>"
"Double sided? <YES>"
"80 tracks? <YES>"

(12) Your response should be '<RTN>' to the first two questions in most cases. However, your response to the third question depends upon what kind of disk drive you are INITing on. In case you are INITing on a 48 tpi drive, simply type 'NO<RTN>'. This will inform the program that you want the disk to be initialized at 48 tpi, not at 96 tpi. If you were attempting to INIT a 48 tpi disk on a 48 tpi drive, but in error, typed a return to the last question, you could damage the drive.

Procedure for computer systems with multiple H17 drives:
---

(11) For computer systems with multiple H17 drives, no message concerning the BOOT step rate will appear since most single-sided drives are factory set to step at a super-conservative rate of 30 milliseconds. In addition the H17 controller card imposes limitations on drive speed in the form of WAIT STATES.

[1] H89 COMPUTERS WITH MULTIPLE DRIVES

Instead, the following message is printed on the screen:

"Number of sides (1 or 2)? <1>
"Recording density (1=48 tpi, 2=96 tpi)? <1>"

(12) If you have two or more single-sided drives connected to the H17 controller, respond to the first question by typing: '<RTN>'. Then respond to the second question by typing: '<RTN>'.

If you have one or more double-sided drives connected to the H17 controller, respond to the second question by typing: '2'. Then respond to the second question by typing: '2'.

CAUTION

You can damage a disk drive during this step if you instruct the computer to INIT a disk in a 48 tpi drive at 96 tpi. Take care!

For computer systems with multiple H37 or H17 drives:

(13) After you have instructed the computer as to how you want your disk INITed and hit the last '<RTN>', the computer starts the INIT process.

The computer shows you the status of the INIT as it goes along by printing a line of asterisks. If you are initializing a 48 tpi disk the line of asterisks will print halfway across the screen.

If you are initializing a 96 tpi disk the line of asterisks will print all the way across the screen. However, when you initialize a 48 tpi disk, the line of asterisks will only print halfway across the screen.

The line of asterisks looks like this for a 48 tpi drive:

"**********"

(14) At the completion of the INIT process, the computer prints the following message on the screen:

"MEDIA CHECK? <YES>?"

To start the media check, type: '<RTN>' (NO SPACES!)

NOTE

Even if you have first-quality, top brand-name disks, it is ALWAYS a good idea to run the media check, since you never know if there might be bad sectors on any disk. The media check will provide a positive check for you.

When the media check begins, it will also start printing a horizontal row of asterisks, such as that printed by the INIT process above. If media check finds bad sectors it will write the number of the addresses on the screen.

For computer systems with multiple H17 drives:
---

(14) You will not be given an opportunity to perform a media check.

For computer systems with multiple drives:
---

(15) You will be asked if you have any bad sectors. If bad sectors have been identified, you may now type in their addresses.

The computer prints:

"Enter the number of bad sectors, one at a time.
Hit RETURN after each entry, and when finished.
Sector?"

NOTE: DETERMINING BAD SECTORS FOR H17 DISKS

Prior to running INIT for your H17 disks, there are two methods of determining bad sectors:

The first method is to cross over to HDOS 2 and perform the TEST17 "Media Check." If you decide to do this, be sure to write down on a piece of paper the bad sectors determined by the test, if any. Follow-up this action by initializing your disks in HDOS 3.02 and plug in the bad sector addresses when you get to this place in the program.

The second method is to use other software such as "BAD.ABS" which is explained in Chapter 7, page 7-38, or the equivalent.

[1] H89 COMPUTERS WITH MULTIPLE DRIVES

NOTE: OPTIMIZING H17 DRIVE SPEED

The H17 step rate may be speeded up on a disk-to-disk basis, but not during INIT. The best method is to first run the Seek Time test, part of TEST17 from within HDOS 2. This will determine the maximum error-free step rate of individual drives. Once this rate is determined, mark the drive with a stick-on label with the maximum reliable steprate determined by test. After this, EACH DISK to be used in that drive may be speeded up by using the SET command (but not during INIT). For example, type: 'SET ^SY1:^HELP<RTN>.' A list of drive SET options will appear on the screen. Then type: 'SET^SY1:^STEP^nn<RTN>', where the expression nn stands for the best step rate determined by test. The SET command must be repeated for each data disk. After you determine the slowest drive in the chain, set all of the drives to that speed.

(16) If bad sectors are found by the media check, you must type the bad sector numbers one at a time after the question "Sector?" For example:

"Sector?"'159<RTN>'

If, on the other hand, no bad sectors were detected, just type a '<RTN>' after "Sector?" and continue. The computer prints:

"Disk Initialization complete."

The computer prints:

"Insert the volume you wish to initialize into SY1:;
Remember, any data on this volume will be destroyed."

(17) You now have three options:

(A) INIT a new disk:

If you want to INIT another disk just remove the disk that was freshly INITed and insert another disk that you want to INIT. Type '<RTN>' to start. A good practice at this time is to apply a self-stick label on the completed disk, so that you can identify it later. The option to INIT multiple disks in a continuous fashion is a big help and is time-saving.

[1] H89 COMPUTERS WITH MULTIPLE DRIVES

For systems with multiple H37 or H17 drives:

(B) INIT a new disk in a different drive:.

You might want to do this if your computer system consists of both 48 and 96 tpi drives.

To do this after the computer prints:

"Hit RETURN when ready,
Ready?"

Hit 'CTRL-D' once.

The computer will ask:

"Device <SY0:>?"

Type 'SY2:' for example, where SY1: is an 80 track drive, and SY2: is a 40 track drive. At this time, you will be instructed to place your disk into SY2:. Then the INIT process repeats.

(C) Exit INIT.

To exit after the computer prints:

"Hit RETURN when ready,
Ready?"

Hit: 'CTRL-D' twice.

At this time the computer will print on the screen:

"Do you have any more disks to initialize? (YES/NO)^<NO>?"

Hit a '<RTN>' and the computer completes the INIT process and then mounts the disk in SY0: without requiring further instructions. Then it prints:

"Volume 00000, mounted on SY0:
Label: HDOS 3.0 - Distribution Disk"

The HDOS 3.02 system prompt reappears.

(18) Before you forget what contents you had planned for the initialized disks, you are strongly urged to identify the disk(s) with label(s).

[1] H89 COMPUTERS WITH MULTIPLE DRIVES

INIT (Cont) INITIALIZING A DISK FOR THE H89 STEP 2

Remove the disk you have just initialized. With a felt tip pen (NEVER PENCIL OR BALLPOINT) once the label is affixed to the disk write the volume number, 10, for example and disk title, SYSTEM VOLUME on a standard disk label, and attach the label to the disk.

[1] H89 COMPUTERS WITH MULTIPLE DRIVES

SYSGEN SYSTEM GENERATION STEP 3

Introduction

SYSGEN is an abbreviation for "System Generation." In this procedure you will copy HDOS 3.02 system files from the distribution disk to your SYSTEM VOLUME. After performing this procedure, you will be able to substitute the SYSTEM VOLUME into any of the preceding steps that formerly required the distribution disk. If you have performed this procedure before, refer to "SYSGEN Options," Chapter 3, page 3-10.

During the SYSGEN operation, the computer is instructed to perform two main tasks. One main task is to copy system files (or other files by your command) from the source disk to the destination disk. Another main task is to set flags on files and add HDOS boot data on track 0 of the destination disk.

When the following command examples contain the symbol [^], this indicates that a space must be inserted. Do not type the symbol [^] instead of the space, just type the space.

NOTE

There is essentially no difference in SYSGEN for a computer system with multiple H37, H47 and H17 drives. systems with multiple H17 drives. Therefore, the following instructions will apply to all these configurations.

Procedures

(1) Insert the bootable distribution disk into SY0: and press <RTN> to boot it.

(2) For information concerning the three forms of SYSGEN, refer to Chapter 3, SYSTEM OPTIMIZATION. To perform SYSGEN in the standard fashion, proceed as follows:

Type: 'SYSGEN<RTN>' HDOS prints:

"SYSGEN
Version 3.0
ISSUE: #50.07.00"

"Destination Device<SY0:>?"

(3) Type: 'SY1:<RTN>' and the computer prints:

"Dismounting all Disks:"

"Volume 00010. Dismounted from SY0:.
LABEL: SYSTEM VOLUME"

[1] H89 COMPUTERS WITH MULTIPLE DRIVES

"Remove the disks.
Hit RETURN when ready!"

(4) Press the RETURN key. The computer prints:

"Insert the Source Diskette in SY0:
Hit Return when Ready:"

(5) Press the RETURN key. The computer prints:

"Insert the Destination Diskette in SY1:
Hit Return when Ready:"

(6) The computer whirs and buzzes for a period of time, and the red leds on the drives SY0: and SY1: turn on and off for a few seconds, as they normally do when files are being transferred from one disk to another. Then the computer finishes its task and is momentarily quiet.

The computer prints:

"5 Files Copied:"

The files that it copies are as follows:

1. HDOS30.SYS 40
2. SYSCMD.SYS 40
3. TT.DVD 13
4. PIP.ABS 49
5. SY.DVD 20

Remember, the files: GRT.SYS, RGT.SYS, and DIRECT.SYS were copied to the destination disk during the INIT procedure. After the files that SYSGEN copies to the disk, you now have a total of 8 system files.

Then HDOS 3.0 automatically mounts SY0:, requiring no instructions.

"Volume 00010, Mounted on SY0:;
Label: HDOS 3.0 - SYSTEM VOLUME"

Go on to Step 5, "SET."
****

[2] H89 COMPUTERS WITH A SINGLE DRIVE

BOOT BOOTING A DISK STEP 1

An H89 Computer with a single drive boots a disk exactly like an H89 Computer with multiple drives. For details on booting, refer to Page 2-3.

****

INIT INITIALIZING A DISK STEP 2

Introduction:

For details concerning "Introduction," see page 2-8.

With respect to performint INIT, there are some differences between an H89 with multiple drives and an H89 with a single drive. The following data illustrates how to INIT a blank disk or a disk to be recycled.

Procedure

(1) At the HDOS 3.02 prompt, type 'INIT<RTN>'

(2) INIT will introduce and describe itself. Then it will ask you if you want to continue. Below this paragraph the following message will be printed:

"(YES/NO)^<NO>?"

(3) Type 'YES', and the following message will print on the screen:

"Dismounting all Disks"

"Volume 00000, Dismounted from SY0:
Label: HDOS 3.02 - SYSTEM DISTRIBUTION DISK"

"Remove the disk(s). Hit RETURN when ready:"

At this time remove the disk in SY0:.

NOTE: The INIT program is a stand-alone program. It loads into memory and runs without needing a system volume in SYO:.

(4) When the query:

"Device<SY0:>?"

is displayed, type '<RTN>'

[2] H89 COMPUTERS WITH A SINGLE DRIVE

INIT (Cont) INITIALIZING A DISK STEP 2

The system prints:

"Insert the volume you wish to initialize into SY0:;
Remember, any data on this volume will be destroyed."

(5) Insert the disk to be initialized into drive SY0: and then the system prints:

"Hit RETURN when ready.
Ready?"

(6) Type '<RTN>'

The computer prints:

"The volume in the drive .....
Apparently has not been initialized before."

(Assuming that you are using a new blank disk.)

Without pause, the computer prints:

"Type NO to cancel. Type YES to
erase and initialize the disk. (YES/NO)?"

(7) Type 'YES<RTN>'

The computer instructs:

"Enter a volume serial number between 0 and 65535."

(8) As an example, type: '10<RTN'

The computer instructs:

"Enter a volume label of 60 characters or less."

(9) Type 'SYSTEM VOLUME<RTN>'

NOTE: For computer systems with a single drive connected to the H17 controller, no message concerning the BOOT step rate will appear. Most single-sided drives are factory set to step at a super-conservative rate of 30 milliseconds. In addition, the H17 controller card imposes limitations on drive speed in the form of WAIT STATES.

The computer prints the following message on the screen:

"Number of sides (1 or 2)? <1>"
"Recording density (1-48 tpi, 2=96 tpi)? <1>"

[2] H89 COMPUTERS WITH A SINGLE DRIVE

INIT (Cont) INITIALIZING A DISK STEP 2

(10) If you have one single-sided drive connected to the H17 controller, respond to the first and second questions by typing '<RTN>'. This is the default response.

If you have one double-sided drive connected to the H17 controller, respond to the first and second questions by typing '2'.

CAUTION

You can damage a disk drive during this step if you instruct the computer to INIT a disk in a 48 tpi drive at 96 tpi.

After you have instructed the computer as to how you want your disk INITed and hit <RTN>, the computer starts the INIT process.

In HDOS 3.0, the computer shows you the status of the INIT as it goes along by printing a horizontal line of asterisks. If you are initializing a single-sided 48 tpi disk, the line of asterisks will print halfway across the screen.

Initializing an H17 disk does not provide you with the opportunity to do a media check. Instead, you will be asked if you have any bad sectors. If bad sectors have been identified, you may now type in their addresses.

NOTE: FINDING BAD SECTORS

One way to find whether there are any bad sectors is to boot HDOS 2.0 and perform the TEST17 "Media Check." HDOS 3.0 does not offer these tests, since they do not run in the HDOS 3.0 environment. After this check is made, the data determined the data should be recorded so that it may be typed in when the computer asks for it.

Another way to find whether there are any bad sectors is to use the utility program called "BAD.ABS," or an equivalent. This program is briefly described in Chapter 7, page 7-38, and is available from Kirk Thompson as an add-on to the HDOS 3.0/3.02 Operating System Manual. "BAD.ABS" may be run after the INIT is completed.

When the computer is finished INITing your disk, it instructs:

"Enter the number of bad sectors, one at a time.
Hit RETURN after each entry, and when finished."

[2] H89 COMPUTERS WITH A SINGLE DRIVE

(11) If you have determined whether there any bad sectors on your disk by running the "Media Check" in HDOS 2.0, now is the time to type in their addresses. For example:

"Sector?"'100159<RTN>

If you have more than one bad sector, every time the computer prints:

"Sector?"

You type in the addresses until all the addresses are done. Then, when the computer prints the next "Sector?" just type a <RTN>.

If you decide to ignore this process or if you have not determined whether any bad sectors exist, you may just type <RTN> after the query "Sector?" to enable the program to move on to the next step. However, this practice is discouraged, unless you intend to use "BAD.ABS" later.

After you type <RTN>, the computer prints:

"Insert the volume you wish to initialize into SY0:;
Remember, any data on this volume will be destroyed."

"Hit RETURN when ready"

(12) At this time you have three options. Refer to page 2-13 for details. However, if you are done with the INIT program, just type:

'CTRL-D' twice.

The computer will ask:

"Do you have any more disks to initialize? (Yes/No)^(NO>?"

Just type '<RTN>' or 'NO'.

If you remove your freshly INITed disk, and swap it with the bootable system disk, the HDOS Operating System will reboot your system disk automatically.

Now go on to STEP 3 for instructions on SYSGEN, or how to make your data disk bootable. These instructions are immediately following.
****

[2] H89 COMPUTERS WITH A SINGLE DRIVE

SYSGEN SYSTEM GENERATION STEP 3

The Heath Company has kindly provided for those with a computer system having only one drive. SYSGEN is a stand-alone program. This means that once started, the program continues on to completion with only minimal cooperation from the user.

You don't NEED multiple drives to SYSGEN disks. However, having multiple drives makes things more convenient.

Insure that the program SYSGEN.ABS is on it, and then BOOT up with your normal system disk. Type: 'SYSGEN<RTN>'. SYSGEN starts up by identifying itself and then it gives you a chance to abort. If you continue, the program will print:

"Destination Device <SY0:>?"

Type: '<RTN>' and switch disks. Remove your normal system disk and insert the disk you have just INITed during STEP 2, INIT. Then the SYSGEN program begins. After it copies the five system files the program is complete.

At this time, insert your normal system disk. HDOS 3.0 does the rest, as it brings your system disk up to BOOT without the need for further instructions.

For details, follow the computer/user dialogue starting on page 2-16.

****

[3] H8 COMPUTERS WITH H19 TERMINAL

BOOTSTRAP BRINGING A DISK TO BOOT LEVEL STEP 1

Introduction

One of the primary functions of a computer operating system is to enable the various physical parts of the computer to cooperate toward the execution of your commands. In order for this cooperation to take place, there must be communication between HDOS (software) and the physical parts of the computer (hardware). The computer cannot execute any command unless HDOS is communicating with the hardware.

Bootstrap is a small program stored within the hardware which serves to establish a communications link between HDOS and the various physical parts of the computer system. The bootstrap procedure is so named because, by means of this procedure, you will be causing HDOS to "pull itself up by its bootstraps" -- that is, the responses you give in this procedure will enable HDOS to lift itself off the disk and into the computer's memory. Having been installed in memory, HDOS can then issue instructions to and coordinate the actions of the appropriate parts of the computer system in response to your commands.

[3] H8 COMPUTERS WITH H19 TERMINALS

SYSGEN SYSTEM GENERATION STEP 3

Procedures

(1) First apply A-C power to the H8 and your terminal.

(2) Apply A-C power to your disk drives.

(3) Install the distribution disk in the disk drive unit that has been hardware configured as primary boot drive 0. Close the door of the drive unit after you have inserted the disk.

NOTE

All attempts to run HDOS 3 with an H8 using a PAM-8 ROM will be doomed to failure. HDOS 3 is "ORG-0," and requires that both the Org 0 board and the XCON8 ROM be installed. The PAM-8 ROM will not work.

(4) If you are using the XCON8 ROM with your H8, press the 1 key on the H8 front panel. The front panel LEDS will display:

Pri H17 or
Pri H37 or
Pri H47

depending upon which kind of drive you have hardware configured to be SY0: on the Org-zero level.

If you have the H8-4 card installed, regardless of what you enter at the H8 front panel, the front panel leds will display: SPACE. However, this is ROM dependent. There are some non-Heath ROMs available that do not show SPACE.

NOTE

The unique features of the H8 have been presented above. Essentially, comparing the H8 to the H89, the initial steps of the BOOT sequence are the only thing that differs.

****

TEST17/TEST37/TEST47 STEP 4

NOTE: TEST17/TEST37/TEST47 must be run on HDOS Version 2.0, as it is not available in HDOS Version 3.0.

Three test programs are provided: TEST17/TEST37/TEST47. These tests provide checks for testing both the disk media and the disk drives themselves. Each of these tests contain the same type of tests as the other, but each has been designed to operate under different conditions. For example, TEST17 tests H17, 5-1/4 inch hard-sectored disks and drives, while TEST47 tests 8-inch disks and drives, and TEST37 tests H37, 5-1/4 inch soft-sectored disks and drives.

Therefore, due to the close similarity between the three tests, describing the testing processes may be combined into one section.

The programs, TEST17.ABS, TEST37.ABS, and TEST47.ABS are included in the HDOS 2.0 distribution disks. Therefore, those persons interested are invited to perform the tests desired in HDOS 2.0. It should be noted that these tests will not run in the HDOS 3.02 environment.
****

SET SYSTEM OPTIMIZATION STEP 5

NOTE: When the following command examples contain the symbol [^], this indicates that a space must be inserted. Do not type the symbol [^] instead of the space; just type the space.

This section will enable you to communicate effectively with HDOS by using some special features of your computer.

(1) At the HDOS system prompt, type:'SET^TT:^BKS<RTN>'. This instructs the system to allow you to backspace (using the BACKSPACE, or DELETE keys) in order to delete characters.

(2) At the next HDOS system prompt, type:'SET^TT:^NOMLI<RTN>'.This instructs the system to allow you to input lower case letters as well as upper case.

(3) At the next HDOS system prompt, type:'SET^TT^NOMLO<RTN>'. This tells HDOS to display all lower case input as lower case output instead of expressing it all in capital letters.

(4) At the next HDOS system prompt,type:'SET^WIDTH 255<RTN>'. This tells HDOS to set an "unlimited" right-hand margin. This will be useful for certain applications, such as working spreadsheets, typing source code, etc.

(5) At the next HDOS system prompt, type:'SET^TT:^1SB<RTN>'. This sets the terminal driver to one stop bit.

(6) At the next HDOS system prompt, type:'SET^TAB'. This lets the terminal process tabs faster.

SET (Cont) SYSTEM OPTIMIZATION STEP 5

For more information about the SET command and SET options, refer to the SET section of Chapter 3, page 3-19, "System Optimization."

Leave the SYSTEM VOLUME in the drive and go on to Step 6, "Preparing a WORKING DISK." Put your original distribution disk away in a safe place, separate from the working disks that you will create. You should not use it again unless something accidentally damages your new working SYSTEM VOLUME disk.

****

SYSGEN/COPY PREPARING A WORKING DISK STEP 6

Introduction

In this section you will prepare a system volume which will contain the most-used files contained in the Heath distribution disk, done the "easy way." Since this disk will contain the essential HDOS system files, you will be able to use it to perform Bootstrap.

This section presumes that you will be starting with a disk that has been INITed.

Procedure

(1) At the HDOS system prompt, type:'SYSGEN /MIN<RTN>'.

(2) When the message "DESTINATION DEVICE<SY0:>" is printed, type: 'SY1:<RTN>'.

(3) HDOS will instruct you to remove the disks. Instead, type: '<RTN>'.

(4) Since both source disks and destination disks are mounted in their respective drives, SYSGEN will begin copying the minimum number of system files that are required to make the disk bootable.

(5) After performing SYSGEN in HDOS 3.0, the system disk in SY0: is automatically re-mounted, and HDOS is ready for your commands.

(6) Copy only the files on your DISTRIBUTION DISK that you desire to have on your destination disk. For example, use the following command:

'COPY SY1:BASIC.ABS=SY0:BASIC.ABS<RTN>'

****

CONFIGURING LINE PRINTERS STEP 7

If you have an H14, H24 (TI-810), H25, H44 (Diablo), or MX-80 printer, this section will instruct you how to incorporate it into your system. With an H-89 there is no jumper to control the port address. The H89 has two printer ports on the rear panel: 340Q and 320Q. If you have an H8 computer, make sure that the jumpers on your serial I/O card are configured for address 340Q. If you need to set the jumpers, first turn off the AC power, perform the change, and return to this manual to continue.

(1) Insert your SYSTEM VOLUME and boot up the system.

(2) At the HDOS system prompt, copy all of the device driver files to a freshly initialized data disk.

(3) The following is an enhanced list of all of the printer drivers on the HDOS 3.0 distribution disks:

FILENAME	SIZE	DATE	FLAGS	MODE	PRINTER
AT84.DVD	5	6-OCT-86	WC D	Serial	*AT
AT85.DVD	5	6-OCT-86	WC D	Serial	*AT
H1484.DVD	6	7-OCT-86	WC D	Serial	H14
H1485.DVD	6	7-OCT-86	WC D	Serial	H14
H2484.DVD	6	7-OCT-86	WC D	Serial	H24(TI-810)
H2584.DVD	10	5-OCT-86	WC D	Serial	H25
H4484.DVD	8	9-OCT-86	WC D	Serial	Diablo
MX8084.DVD	8	10-OCT-86	WC D	Serial	MX-80
MX8011.DVD	8	10-OCT-86	WC D	Parallel	MX-80

This is a directory of all of the HDOS 3.02 line printer device driver files.

NOTES:
1. * AT: Alternate Terminal - configured at port 320Q via an H8-4 card for the H8 Computer, or port 320Q for the H89 Computer.

2. * AT: Alternate Terminal - configured at address 374Q via an H8-5 card for the H8 Computer.

In most cases you only need one of the files, and the one that you need depends upon what line printer you are using, and also whether it is connected in the serial or parallel mode. For instance, if you have an H14 printer, you should select H1484 if you are using the serial mode, or H1485 if you are using the parallel mode.

To access the parallel mode, you must be either using the Heath H89-11 parallel card, or some non-Heath vendor's parallel card.

CONFIGURING LINE PRINTERS (Cont) STEP 7

UNOFFICIAL NOTE

At this time there are some printer drivers provided by non-Heath vendors that will print your files in the HDOS 3.0 environment. For example, Quikdata sells the "Ultimate Driver" written by Bill Lindley of Lindley Systems. Quikdata's address is: Quikdata Computer Services, Inc., 2618 Penn Circle, Sheboygan, WI, 53081. Kirk Thompson, editor of the Staunch 8/89'er Newsletter, may also have some printer drivers. Contact Kirk at the following address: Mr. Kirk L. Thompson, Lot #6 West Branch Mobile Home Village, West Branch, IA, 52358.

(4) The next step is to copy the appropriate printer driver to your System Volume. After you have selected the printer driver that you want to use, copy it to your System Volume in accordance with the following example:

'COPY SY0:AT84.DVD=SY1:AT84.DVD<RTN>'

(5) Having copied the device driver to your System Volume, you must rename it. According to the rules for device drivers, the filename must be composed of two letters, and the extension must always be ".DVD." Rename the device driver as follows:

'RENAME SY0:LP.DVD=AT84.DVD<RTN>'

After renaming a device driver, you must reboot in order to get HDOS 3.02 to recognize the driver.

In the HDOS 3.0 environment, one may have many printer drivers. However, each of the drivers must have a different two-letter filename. Some common examples of filenames for printer drivers are: UD.DVD, EP.DVD, etc. However, the primary printer driver should be named LP.DVD, since many user programs have been built to search for that driver before they can continue with their assigned task. One example is QUERY!2.

HDOS will now recognize commands to utilize LP:, the primary printer driver, the next time you boot from the working disk. For more information about configuring line printers and other peripherals, refer to the PERIPHERALS" section of Chapter 3, System Optimization.

PRINTER DRIVERS WITH MULTIPLE UNITS:

Some printer drivers, such as the Lindley driver for HDOS 3.0/3.02 will enable you to make your printer print out in different modes. For example, 10, 12, 15, 17, or 20 characters per inch (CPI), or 6 or 8 lines per inch (LPI) draft mode, letter quality mode, etc, depending upon whatever features have been built-in to your printer. These modes are controlled by printer codes.

These printer codes are obtained from your printer manual. For example, in the Panasonic KX-P1124 Printer Manual, the codes are listed under the chapter for Mode Commands. One example is the printer code for EMPHASIZED PRINTING: for Hex to turn it on, type 1B 45. To turn it off type 1B 46.

CONFIGURING LINE PRINTERS (Cont) STEP 7

Each text editor has different ways of imbedding these codes. For example, in EDIT19 to set the printer codes, follow these procedures:

1. Put the cursor on the line where you want the printer code.
2. Press the "ERASE" key to go into Command Mode.
3. Type C<RTN>.
4. The line to have the printer codes added will appear in the command screen.
5. Type CTRL-D, then the first byte of the control code.
6. Type CTRL-D again for the second byte.
7. Continue typing CTRL-D and follow up with a third byte, and continue this until the last printer code is typed.
8. Press RETURN and the line will be restored to the editing screen.
9. Press ERASE to return the cursor to the editing screen.
10. Note the bullet-shaped control codes in the text, where you placed them while in the command screen.

Often it is necessary to translate the codes provided in your printer manual to other types of notation. For example: If your manual shows the control codes in decimal, but the printer driver requires the codes to be converted to OCTAL, you need a conversion chart. Refer to Appendix 2-C, page 2-38 for a decimal to octal to hex to to ASCII conversion chart.

Remember to reboot after you create these code strings, so that the system will recognize your printer drivers.

Leave the disk in the drive and go on to STEP 8, "Power Down."

POWER DOWN STEP 8

Part of the operating system resides in memory at all times, and part of it resides on the disk. Any alterations you have made to the portion of the system that is being stored in memory may or may not have been written to the disk. Thus, whenever you are finished using your computer, perform one of the following procedures to insure that any configuration changes you have made to the system are written back to the disk:

POWER DOWN (Cont) STEP 8

If the message "Install a Bootable disk in SY0: Hit RETURN to reboot' is displayed on the terminal, such as occurs after running INIT, it is always safe to remove the disk and turn off the power. Be sure that you remove all disks before turning off the A-C power, since disks left in the drives during power-down may be magnetically damaged.

If the HDOS system prompt is displayed on the terminal:

(1) Type 'BYE<RTN>'.

(2) Remove all disks.

(3) Turn off the A-C power to the drives, computer, and power bar. It is a good practice to turn off the power to the drives first before turning off the power to the computer to avoid drive head-banging. If you have a power bar, just leave the switches on the computer and disk drives on and switch off the power bar.

****

SUMMARY

You now have two bootable disks. Volume 1, SYSTEM VOLUME, is a complete copy of the distribution disk. Volume 10, WORKING DISK, is a subset of the SYSTEM VOLUME and contains the files necessary to operate the system as well as BASIC.

You will need to perform parts of the "System Set-Up Procedure" again. All new disk must be initialized. If you use H17 disks, we recommend that you run the M (Media Check) option to TEST17.ABS, or equivalent. This check will tell you whether there are any bad sectors on the disk.

Another option is to use the utility "BAD.ABS," which may be purchased separately from Kirk Thompson, since it is quicker and easier to use.

If there are any bad sectors, use INIT to instruct HDOS not to write to any of the bad sectors. Remember that even if the media check finds no bad sectors, you must reinitialize the the disk upon which the test was performed if you use the Media Check associated with TEST17.ABS

Since the SYSTEM VOLUME is a duplicate of the DISTRIBUTION disk, you can substitute your SYSTEM VOLUME into any procedure that formerly required the DISTRIBUTION disk. A system volume does not have to be an exact copy of the distribution disk. A system volume is simply a disk that contains HDOS system files. All disks that have been SYSGENed are system volumes. In this sense the WORKING DISK is actually a system volume. Only system volumes may be used to perform Bootstrap, so if you have a one-drive system, you will probably want to SYSGEN all of your disks.

To create several SYSTEM VOLUMES each with a specific system resource, such as EDIT, DBUG, ASM, or BASIC:

SUMMARY (Cont) STEP 7

(1) Perform Bootstrap using your new SYSTEM VOLUME.

(2) Initialize the blank disk. Name it anything you like, but it is a good practice to name your disks according to their function so you can distinguish among them. Note that each volume name and number should be unique -- that is, you should avoid assigning the same volume number or disk label to two different disks.

(3) Run SYSGEN. The SYSTEM VOLUME is the source and the blank disk is the destination.

(4) Run ONECOPY if you have a single drive computer system, or PIP if you have a multiple drive system. See Chapter 3, "System Optimization" for more information about ONECOPY and PIP. Instead of specifying *.* after the OC: or P: prompt, specify the name of the individual file or files that you want to be copied to your new volume.

(5) Run SET, using the same commands you used in the preceding SET section.

(6) Perform the Power-Down procedure.

If you are using 5-1/4 inch drives as your primary boot drives, you will need to use either PIP or ONECOPY to transfer system utilities such as ASM, DBUG, and EDIT from your SOFTWARE TOOLS disk. The use of both PIP and ONECOPY is documented in Chapter 3.

If you have a multiple disk drive system, the COPY command will copy either single files or multiple files most handily. One neat utility program that is furnished with the HDOS 3.02 Operating System programs is called "MegaPip," or "MP.ABS."

In addition, certain well-respected vendors have programs that will also copy files. For instance, T & E Associates provide two disks called "HDOS Enhancements." These disks include the files: DM.ABS and FM.ABS. These programs enable one to easily copy files singly or the entire disk, while verifying the copies. These programs were originally designed for HDOS 2.0, but they work fine in the HDOS 3.0 environment. The address for T & E Associates is: P.O. Box 352, Millersville, MD 21108, phone: (301) 987-4748.

You will probably want to proceed from here directly to the HDOS 3.0 Manual, Chapter 3. At some point you will want to scan the data provided in the appendixes for chapters 1, 2, and 3. This information will come in handy when you start working HDOS 3.0 in earnest. Refer back to Chapter 2, "General Operations" for details on INIT and SYSGEN as required.
****

APPENDIX 2-A: BOOTING TECHNIQUES

DETAILS CONCERNING BOOTING:

ASSUMPTION: These instructions assume that you have a multiple disk drive computer system with three disk drives connected to the H37 soft-sector controller and two drives connected to the H17 hard-sector controller. Even if you have fewer drives than that, or have a different type of controller such as the H47 as long as you have a minimum of two drives connected to each type of controller, these instructions will work. If your system is different from that shown, you will have to modify the instructions to fit your system.

(1) Decide which type of drive that you want to use the most: H17, H37, or H47, hard or soft-sectored. Due to considerations of speed and disk storage capacity, most people choose to make either the H37 or H47 types their primary boot drives. Remember: primary drives are always drives located on the SYn: drive chain. Secondary drives are always located on the DKn: drive chain.

(A) If you desire to use the hard sector drives as primary boot drives, ignore step (1) or whatever doesn't apply to your system.

(B) If you desire to use the H37 or H47 drives as primary boot drives, you must make one simple, power off, hardware adjustment to your CPU board. The following instructions apply if you are using a MTR-90 Monitor ROM, Heath part number 444-142, 444-41, or 444-83. (Also some non-Heath Monitor ROMs such as the Kres KMR-100.)

(a) With A-C power shut off and the line cord disconnected from the wall socket, open your computer top cover. Disconnect the fan cable and remove the top cover. DIP Switch SW501 is located on the lower right of the H89/Z90 CPU card. The switch is marked with a legend printed on the board. Be sure you identify the correct switch.

(b) Instruct your computer to designate the H37 drives as primary by insuring that all seven of the little switches on DIP Switch SW501 are set to 0 (zero). Similarly, if you want to designate the H47 as primary drives, set DIP Switch SW501 to the following: PIN 1 = 1, PIN 3 = 1. All other pins are set to zero. This is easily done by using the eraser end of a long wooden pencil to perform any switch changes that are required. The one and zero positions are marked on the part.

(c) Step (b) will insure that the high capacity drives will function as the primary booters and that the H17 drives will function as the secondary booters, provided that the drives have been correctly physically designated. You may have to redesignate them now by adjusting the programming plug on the disk drive.

NOTE: To physically designate a drive means to disconnect the drive from your computer system and adjust the programming plug to the appropriate settings. For details on how to program drives, refer to Appendix 2B, page 2-35.

APPENDIX 2-A: BOOTING TECHNIQUES (Cont)

If the configuration of your computer system at first designated the SYn: chain of drives as "hard sector," and the drives were single-sided 400-sector types, you will have to gain access to the drives and manually reset the programming plugs to change them to be either DK0:, DK1:, or DK2. Likewise, your soft sector, double-sided drives must be set to SY0:, SY1, and SY2 in a like manner.

(2) Normally you boot the drive that has been configured as SY0:. The first command given to the computer when at the monitor ROM prompt, usually the "H:" prompt is:

'B<RTN>'

It is also possible to boot from the second or third primary drive. In the case of booting from drives other than the drive physically programmed to be SY0:, the command is:

For Booting SY1: 'B1<RTN>' For Booting SY2: 'B2<RTN>'

(3) If you want to boot from the secondary drives, assuming that you would like to boot from hardware configured drive DK0: or DK1:, respectively, the commands are:

For Booting DK0: ';SD<RTN>' For Booting DK1: ';SD1<RTN>' etc.

BOOTING THEORY

If you have an H89 computer with two or more drives, and are using one of the following combinations: H17 + H37, H17 + H47, or H37 + H47 drives, when you boot from a drive other than the one designated SY0:, all the drives change their names. In order to operate the computer, you have to be able to predict what the new name will be for each drive that you have. The following data is presented in the form of charts which show this information presented in an easy-to-read fashion.

It is important to note that whatever drive you boot from, whether it be H17, H37, or H47, that drive and others connected to the same disk controller become SY0:.... SYn:, and the drives connected to the other disk controller become DK0:.... DKn:.

The alternate device, DKn:, always has a logical number the same as the physical drive number. DK0: is the drive physically designated as hardware unit 0 (zero), DK1: is the drive physically designated as hardware unit 1 (one), etc. The primary drive logical numbers rotate among all the possible unit numbers, whether or not a disk drive is physically connected. Note in the tables shown below, the logical drive names rotate as you boot from physical units 0, 1, 2, or 3.

The system shown is composed of both H17 and H37 drives. Drives SY0: through SY2: are H17, hard-sector. Drives DK0: through DK2: are H37, soft-sector.

APPENDIX 2-A: BOOTING TECHNIQUES (Cont)

BOOT MAP: COMBINATION OF H17 AND H37 DRIVES

BOOT DRIVE	H17 #0	H17 #1	H17 #2	H37 #0	H37 #1	H37 #2	H37 #3
H17 (0)	SY0:	SY1:	SY2:	DK0:	DK1:	DK2:	DK3:
H17 (1)	SY2:	SY0:	SY1:	DK0:	DK1:	DK2:	DK3:
H17 (2)	SY1:	SY2:	SY0:	DK0:	DK1:	DK2:	DK3:
H37 (0)	DK0:	DK1:	DK2:	SY0:	SY1:	SY2:	SY3:
H37 (1)	DK0:	DK1:	DK2:	SY3:	SY0:	SY1:	SY2:
H37 (2)	DK0:	DK1:	DK2:	SY2:	SY3:	SY0:	SY1:
H37 (3)	DK0:	DK1:	DK2:	SY1:	SY2:	SY3:	SY0:

BOOT MAP: COMBINATION OF H17 AND H47 DRIVES

BOOT DRIVE	H17 #0	H17 #1	H17 #2	H47 #0 (LEFT)	H47 #1 (RIGHT)
H17 (0)	SY0:	SY1:	SY2:	DK0:	DK0:
H17 (1)	SY2:	SY0:	SY1:	DK0:	DK1:
H17 (2)	SY1:	SY2:	SY0:	DK0:	DK1:
H47 (0)	DK0:	DK1:	DK2:	SY0:	SY1:
H47 (1)	DK0:	DK1:	DK2:	SY1:	SY0:

HOW TO USE THE CHARTS

It is easy to boot from the drive physically designated as SY0:, but there may be times when you would like to boot from the drive next to it which is configured as SY1:. One good reason for this is if you have a computer system composed of three H37 drives, where the first drive is a 40 track drive, and the second and third drives are 80 track. If you want to produce a 40 track soft sectored disk, you must boot from one or the other of the 80 track drives. In that case, the logical drive names will change. Another good reason is if one of your drives, say, the drive normally used as SY0: should fail.

APPENDIX 2-A: BOOTING TECHNIQUES (Cont)

To use the charts provided, keep in mind the type of system you have (i.e., H17 + H37 or H17 + H47), and then simply determine the type of drive you wish to boot from. With that data available, go to the appropriate table, and scan down the left hand column until you find the type of drive you want to boot from and the number.

For example: you want to boot from an H37 drive physically jumpered as SY1:. On the extreme left column, find the H37 (1). Scan that line to the right and find the associated drive names that apply to your system. You may have to study the tables carefully before the logic pattern becomes apparent. If you are still not certain, one sure-fire test is to perform a "DIR" to the drives, first SY1:, then SY2:, etc. The drive where the red light comes on bears the name you directed the DIR to.

Even if you only have two drives connected to the H37 controller, the mapping stays the same. If you have two H37s jumpered as physical units 0 and 1, then boot from drive 0, everything will make sense: the drives will respond to SY0: and SY1:. But if you boot from the drive jumpered as physical drive 1, the names change in an unexpected way. The boot drive (physical unit 1) becomes logical drive SY0: but the other drive (physical unit 0) becomes logical drive SY3:. In this case, one would reasonably think that the drive (physical unit 0) would become logical drive SY1:, but it doesn't. Any call going out to SY1: or SY2: will prove it. To unlatch the lockup this causes, type CTRL-Z twice.

DEFINITIONS:
Physically jumpered or designated drives are drives that have been physically set to operate as SY0: through SY3: and/or DK0: through DK2:. To physically jumper or designate a drive one must adjust the drive plug that resides on the logic card of the disk drive.

Logical drive names may be completely divorced from the physically jumpered or designated drive names. A logical drive name is one that HDOS requires when a program is being run. Logical drive names change around without anything physical being done to them.
****

APPENDIX 2-B: PROGRAMMING DRIVES

There are two different types of disk drives suitable for use with the H8, H89, and Z90 on the market today. The older style is the full-size drive, and the newer style is the half-height drive. Although the drives are different in appearance, the general manner of designating a drive is the same.

TERMINATING RESISTOR

The last drive on a chain must contain a terminating resistor. This applies whether you have a single drive system or a multiple drive system. This terminating resistor looks like a standard microcircuit, but instead of the conventional circuitry, it consists of a series of resistors in a row. These resistors are connected from one pin to the other pin across the body of the device. Each drive is sold with its own terminating resistor. Therefore, insure that the correct part number of terminating resistor is installed. CAUTION: Do not install a terminating resistor in every drive!

FULL SIZE DRIVES:

Full size drives are manufactured by Tandon, Siemens, Control Data, and a number of other manufacturers.

To set up a computer system, one must give each drive a different name. These names are SY0: thru SY3: or DK0: thru DK2.

The means to designate a drive and make it different from its fellows is to adjust the programming plug which is found on the logic card of the drive assembly. Look for an integrated circuit socket which contains a jumper block instead of a chip.

H37 SOFT SECTOR DRIVE PROGRAMMING:

NAME	HS	DS0:	DS1:	DS2:	DS3:	MX	BLANK	HM
SY0:	(SHORT)	(SHORT)	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN
SY1:	(SHORT)	OPEN	(SHORT)	OPEN	OPEN	OPEN	OPEN	OPEN
SY2:	(SHORT)	OPEN	OPEN	(SHORT)	OPEN	OPEN	OPEN	OPEN
SY3:	(SHORT)	OPEN	OPEN	OPEN	(SHORT)	OPEN	OPEN	OPEN

H17 HARD SECTOR DRIVE PROGRAMMING:

NAME	HS	DS0:	DS1:	DS2:	DS3:	MX	BLANK	H#
DK0:	(SHORT)	OPEN	OPEN	(SHORT)	OPEN	OPEN	OPEN	OPEN
DK1:	(SHORT)	OPEN	(SHORT)	OPEN	OPEN	OPEN	OPEN	OPEN
DK2:	(SHORT)	(SHORT)	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN

APPENDIX 2-B: PROGRAMMING DRIVES (Cont)

Blank jumper blocks may be purchased from Quikdata Computer Services, Inc., 2618 Penn Circle, Sheboygan, WS 53081. T0 program a given drive first determine whether you want to designate a soft sector drive (H37) or a hard sector drive (H17). Consult the proper table and look at your drive logic circuit card. Notice the information printed near the drive jumper block socket. Orient your jumper block to match the data on the circuit card and make your jumper block like the information shown on the table. When you are done, there should be only two different circuits shorted. All the others should be open.

HALF-HEIGHT DRIVES:
A so-called half-height drive is a drive that is nearly half the width of the standard full size drives. It is called half-height because on the newer computers it is mounted on its side and from that angle it lives up to its description, "half-height."

Similar to the full size drives, the half-height drives have a programming jumper block on the logic circuit card. However, instead of having a microcircuit socket that a programming plug fits into, these type of drives have a series of paired pins that stick out from the card in a row. Similar nomenclature is printed on the circuit card near the set of jumpers. The object is to fit the tiny shorting piece over the pin so as to short the pins. Only one jumper bar need be set.

Some drives may have two sets of these paired pins. Look for the set that has the following type of data printed next to it: HS, DS0, DS1, DS2, DS3, MX, BLANK, and HM. This set is usually positioned toward the rear of the card.

Unfortunately, the method of having the set of programming set of pins is not standard. There seems to be two different methods that drive manufacturers use. One type is to call their pins: DS1, DS2, DS3, and DS4. The other type is to call their pins: DS0, DS1, DS2, and DS4. Some drives only offer only three possible pins: DS0, DS1, and DS2 or DS1, DS2, and DS3. Before you purchase a new drive, check to see that there are four sets of programming pins so that you can have four drives on the H37 soft sector controller if you desire.

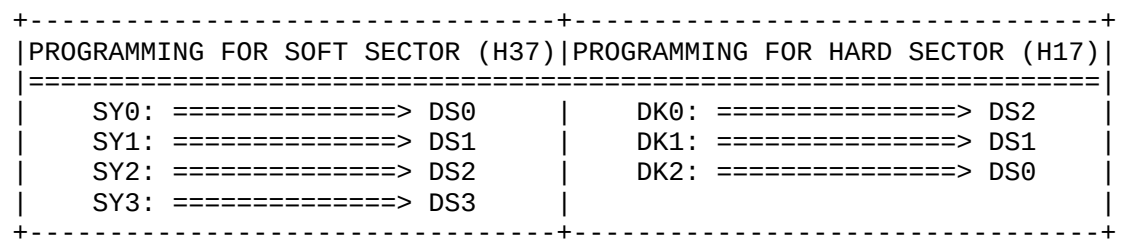
Visually inspect the logic circuit card and check for printed legends.

For example, if the data printed on the circuit card says DS1, DS2, DS3, DS4, fit the second shorting bar as follows:

PROGRAMMING FOR SOFT SECTOR (H37)	PROGRAMMING FOR HARD SECTOR (H17)
SY0: ==========> DS1	DK0: ==========> DS3
SY1: ==========> DS2	DK1: ==========> DS2
SY2: ==========> DS3	DK2: ==========> DS1
SY3: ==========> DS4	

APPENDIX 2-B: PROGRAMMING DRIVES (Cont)

For example, if the data printed on the circuit card says: DS0, DS1, DS2, DS3, fit the second shorting bar as follows:



APPENDIX 2-C
DECIMAL TO OCTAL TO HEX TO ASCII CONVERSION

DECIMAL	OCTAL	HEX	ASCII	CTRL-n	DECIMAL	OCTAL	HEX	ASCII
0	000	00	NUL	CTRL-@	48	060	30	0
1	001	01	SOH	CTRL-A	49	061	31	1
2	002	02	STX	CTRL-B	50	062	32	2
3	003	03	ETX	CTRL-C	51	063	33	3
4	004	04	EOT	CTRL-D	52	064	34	4
5	005	05	ENQ	CTRL-E	53	065	35	5
6	006	06	ACK	CTRL-F	54	066	36	6
7	007	07	BEL	CTRL-G	55	067	37	7
8	010	08	BS	CTRL-H	56	070	38	8
9	011	09	HT	CTRL-I	57	071	39	9
10	012	0A	LF	CTRL-J	58	072	3A	:
11	013	0B	VT	CTRL-K	59	073	3B	;
12	014	0C	FF	CTRL-L	60	074	3C	<
13	015	0D	CR	CTRL-M	61	075	3D	=
14	016	0E	SO	CTRL-N	62	076	3E	>
15	017	0F	SI	CTRL-O	63	077	3F	?
16	020	10	DLE	CTRL-P	64	100	40	@
17	021	11	DC1	CTRL-Q	65	101	41	A
18	022	12	DC2	CTRL-R	66	102	42	B
19	023	13	DC3	CTRL-S	67	103	43	C
20	024	14	DC4	CTRL-T	68	104	44	D
21	025	15	NAK	CTRL-U	69	105	45	E
22	026	16	SYN	CTRL-V	70	106	46	F
23	027	17	ETB	CTRL-W	71	107	47	G
24	030	18	CAN	CTRL-X	72	110	48	H
25	031	19	EM	CTRL-Y	73	111	49	I
26	032	1A	SUB	CTRL-Z	74	112	4A	J
27	033	1B	ESC	CTRL-[	75	113	4B	K
28	034	1C	FS	CTRL-\	76	114	4C	L
29	035	1D	GS	CTRL-]	77	115	4D	M
30	036	1E	RS	CTRL-^	78	116	4E	N
31	037	1F	US	NOTE 1	79	117	4F	O
32	040	20	SP		80	120	50	P
33	041	21	!		81	121	51	Q
34	042	22	"		82	122	52	R
35	043	23	#		83	123	53	S
36	044	24	$		84	124	54	T
37	045	25	%		85	125	56	U
38	046	26	&		86	126	56	V
39	047	27	'		87	127	57	W
40	050	28	(		88	130	58	X
41	051	29	)		89	131	59	Y
42	052	2A	*		90	132	5A	Z
43	053	2B	+		91	133	5B	[
44	054	2C	,		92	134	5C	\
45	055	2D	-		93	135	5D	]
46	056	2E	.		94	136	5E	^
47	057	2F	/		95	137	5F	_

APPENDIX 2-C
DECIMAL TO OCTAL TO HEX TO ASCII CONVERSION (Cont)

DECIMAL	OCTAL	HEX	ASCII	DECIMAL	OCTAL	HEX	ASCII
96	140	60	`	112	160	70	p
97	141	61	a	113	161	71	q
98	142	62	b	114	162	72	r
99	143	63	c	115	163	73	s
100	144	64	d	116	164	74	t
101	145	65	e	117	165	75	u
102	146	66	f	118	164	76	v
103	147	67	g	119	167	77	w
104	150	68	h	120	170	78	x
105	151	69	i	121	171	79	y
106	152	6A	j	122	172	7A	z
107	153	6B	k	123	173	7B	{
108	154	6C	l	124	174	7C	|
109	155	6D	m	125	175	7D	}
110	156	6E	n	126	176	7E	~
111	157	6F	o	127 NOTE 2	177	7F	DEL

NOTE 1: DECIMAL 31: CTRL-SHIFT-HYPHEN
NOTE 2: DECIMAL 127: DELETE KEY
****

HDOS SOFTWARE REFERENCE MANUAL

HDOS DISK OPERATING SYSTEM

VERSION 3.0

CHAPTER 3

SYSTEM OPTIMIZATION

HEATH DISK OPERATING SYSTEM

SOFTWARE REFERENCE MANUAL

VERSION 3.0

 Through the years it continued to be improved by successive revisions which included 1.5, 1.6, and finally 2.0. It was entered into public domain on 19 July 1989 per letter by Jim Buszkiewicz, Managing Editor, Heath Users' Group, P.O. Box 217, Benton Harbor, MI 49022-0217 (616)982-3463. A copy of this letter is available for public inspection.

This manual is indicative of further improvements and provides for the latest revision, HDOS 3.0 and HDOS 3.02. The revision 3.0 is detailed in chapters 1, 2, and 3, while chapters 4 through 8, 13 and 14, are related to revision 3.02. Chapters 9 through 12, with minor improvements, are essentially picked up from the original HDOS 2.0 manual. Indeed, HDOS is still alive and well!

Chapter 3, System Optimization, describes the most efficient ways to perform the basic functions of the HDOS 3.02 Computer System.

SPECIAL DISCLAIMER: The Heath Company cannot provide consultation on either the HDOS Operating System or user-developed or modified versions of Heath software products designed to operate under the HDOS Operating System. Do not refer to Heath for questions.

Instead, you are invited to direct any questions concerning the Heath Disk Operating System (HDOS) to Mr. Kirk L. Thompson, Editor "Staunch 89/8" Newsletter, #6 West Branch Mobile Home Village, West Branch, IA 52358.

TABLE OF CONTENTS

INTRODUCTION

The purpose of this chapter is to show you easier methods to accomplish some standard tasks in HDOS 3.02 and also provide additional detail not available anywhere else in the manual.

****

UTILITY OPTIMIZATION

BOOTSTRAP

The Bootstrap procedure is normally performed from the drive that has been hardware configured as primary drive 0 (SY0:). Only disks that have been SYSGENed (system volumes) are bootable disks. The term "Bootstrap" describes the process by which a series of small programs lifts the operating system off of the disk and into the computer's memory. This process is described in the following paragraphs.

Turning on the A-C power to your computer activates a Read-Only Memory (ROM) chip on the CPU board, which contains a program called the "Monitor ROM Bootstrap Program." If you have an H89, this ROM prints "H:" at the upper left hand corner of the console terminal screen. If you have an H8, the Monitor ROM lights the LEDs on the front panel monitor. The ROM then awaits input.

The valid Bootstrap inputs vary according to which ROM you have installed in your computer and what kind of computer you are using. Regardless of what you enter and how, whether 'B<RTN>' at the H89 terminal or '1' on the H8 front panel LEDs, or 'RST/0' and 'GO' on the H8, the Monitor ROM interprets your input as an instruction to jump to the starting address of the controller Bootstrap ROM.

Having assumed control, the disk controller Bootstrap ROM moves the disk read-write head into a position where it can access the first nine sectors of the disk. The read-write head accesses the disk and then reads in a program in the first nine sectors off of the disk and into memory. These nine sectors are the "disk Bootstrap sectors," which means that these sectors contain a program which is capable of moving HDOS off of the disk and into the computer's memory. How the loading of these sectors into memory is accomplished depends upon what type of device you are using to perform Bootstrap.

In the case of the H47 (8-inch disk), or the H37 (5-1/4 inch disk), the Bootstrap ROM loads the first two of these nine sectors into memory. The first two bootstrap sectors contain a disk driver which enables the Bootstrap ROM to find the other seven sectors. The other seven sectors contain the disk Bootstrap program itself. The two sectors that are in memory then cause the Bootstrap ROM to read the other seven sectors into memory.

In the case of the H17 (hard sector disk) all nine bootstrap sectors are read into memory at once, as the disk driver program is contained within the H17 Bootstrap ROM. The first two sectors of a 5-1/4 inch disk are "dummied." The other seven bootstrap sectors contain the same loading program as their counterparts on an 8-inch disk.

BOOTSTRAP (Cont)

The program contained in the nine bootstrap sectors which are now in memory assumes control from the Bootstrap ROM, and determines the address of the console terminal. The system then checks the disk to see if the terminal baud rate is written there. If the baud rate of the terminal has been stored on disk as a result of a previous boot-up, the system proceeds. If the system cannot find a baud rate stored on the disk, it waits for you to press the space bar, so that it can determine the baud rate. The number of spaces HDOS requires to determine the baud rate will vary. Terminals which transmit at 9600 baud should require three or four spaces. Terminals which transmit at a less frequently used baud rate, such as 110, may require that you type six spaces.

Having determined the port address and baud rate of your terminal, the system displays on the terminal, the message:

"Boot?"

If you do not receive this message, repeat the Bootstrap procedure. Make sure you are using a disk that contains the HDOS system files (i.e., the distribution disk or a copy of it). Also, check to ascertain whether you have correctly installed the disk controller board and interconnecting cable. If your H8 is interfaced to the terminal by means of an H8-4 card, check the cabling between the card and your terminal. If your hardware has been properly installed and you still cannot get the system to boot up, refer to the "In Case of Difficulty" section of the appropriate hardware manual.

If you type 'B' or <RTN> after the "Boot?" message, the nine bootstrap sectors load the file called HDOS30.SYS into memory. HDOS30.SYS is composed of two parts: one part is always resident in memory whenever the operating system is running, and the other part is "one-time" code, which is used only when the system is being booted up. If there are no errors in reading HDOS30.SYS into memory, the bootstrap sectors begin to execute the one time code portion of HDOS30.SYS. The bootstrap sectors then transfer the terminal baud rate and terminal interface type on to the permanently resident portion of HDOS30.SYS. The work of the bootstrap sectors is now complete, and the one-time code assumes control. First, this code determines how much memory is installed in your computer. The one-time code then moves the permanently resident portion of HDOS30.SYS into the upper 3k of RAM. Having done this, the one-time code prints:

"System Has 64K of RAM" (NOTE: HDOS 3.0 is Org-zero.)

"HDOS 3.0
Issue # 50.07.00"

The one-time code now scans the disk directory for file entries of the form XX.DVD. These .DVD files are device driver files, which HDOS uses to communicate with the peripheral devices in your system, including the disk drives.

BOOTSTRAP (Cont)

Based upon information it finds in the .DVD files, the one time code builds a device table, which lists the characteristics of each device. These characteristics include such information as whether a given device is capable of transmitting data, whether the device is capable of receiving data, or whether the device is capable of both transmitting and receiving data. After building the device table, the one time code checks both memory and the disk for a date, provided that the SET HDOS NO-DATE option has not been implemented (see the SET section of Chapter 2 for details). If there is a date both in memory and on the disk, the one time code substitutes the date in memory into the (DD- MMM-YY) format of the date message. If a date has been recorded on the disk only, then the one time code substitutes that date into the (DD- MMM-YY) format of the date message. If there is no date recorded either in memory or on the disk, the one time code prints:

"Date (dd-mmm-yy)?"

Since you are using HDOS 3.0 or later, the date entry is much simpler. First, if you boot up on HDOS 3.0 or later, on the same day, and you have already supplied the date information, the question will not even be asked. If you are booting a day or so later and you are still in the same month and year, all you have to supply is the day, and then a <RTN>. If you enter a new month, you will have to supply the day and the month.

After you have entered a date, or simply typed <RTN>, the one time code stores the date data in memory.

After storing the date, the one time code mounts the disk in SY0:, and then prints:

"Volume nnn, Mounted on SY0:
Label: System Volume"

If either the date or terminal baud rate in memory is not the same as the date or terminal baud rate on disk, the one-time code writes the baud rate and date that are in memory to the disk, unless the disk that is being booted is write-protected. Thus, each disk that has been booted at least once, and which is not write-protected has a terminal baud rate and date written on it. The effect of this is that you do not need to type spaces when you are booting a disk which you have previously used to boot up the system, unless you have altered the terminal baud rate since you last booted that disk. In the same way you never really need to enter a date unless you want catalog listings of the contents of that disk to accurately reflect the date on which each file was created.

Since the one-time code cannot write the date and baud rate onto write protected disks such as the distribution disk, you must always type spaces when you boot from a write-protected disk. The date that appears when you boot a write-protected disk is the date such disks were created.

BOOTSTRAP (Cont)

The final function of the one-time code is to turn control of the system over to a file called SYSCMD.SYS. SYSCMD.SYS is the system command processor which processes all your commands and invokes sub programs, such as INIT, as needed to execute your commands. It is SYSCMD.SYS that displays the HDOS prompt. This prompt indicates that HDOS is in the command mode and awaits your instructions.

****

BOOTSTRAP OPTIONS

As we mentioned earlier, HDOS records the TLB (Terminal Logic Board) baud rate on the disk. If you boot using a terminal whose baud rate does not match the TLB baud rate stored on the disk, you will receive binary garbage, or nothing at all, instead of the "Boot?" message. If this happens, hit the terminal 'BREAK' key and then press the space bar a few times. HDOS will determine the new baud rate and print "Boot?."

Whenever the message "Boot?" is displayed on the screen, you have four options.

(1) First, if you type "H," or any character except "I," or "B," or <RTN>, the system will print:

```
"LEGAL COMMANDS:
 BOOT - BOOT HDOS
 HELP - PRINT THIS LIST"

"Boot?"
```

(2) Second, if you type "I," the system will ignore prologues.

(3) The third option is to do absolutely nothing. HDOS will wait several seconds for some response to the "Boot?" message, and will then boot itself up to the "DATE (DD-MMM-YY)?" prompt, if applicable.

(4) The forth option is normal boot-up. You can boot the system from the "Boot?" message by typing either a '<RTN>' or the letter 'B'.

****

INITIALIZATION (INIT)

INIT is a program designed to prepare HDOS disks for data storage. INIT is a stand-alone utility. This means that HDOS writes the INIT program into a memory buffer, and then runs the program without accessing the disk which contains the utility.

When you type INIT<RTN>, HDOS loads INIT into a memory buffer, dismounts all disk(s), and then passes control to INIT. INIT then scans the HDOS device table for the disk drive device driver entries.

Having found these, INIT checks the end of the disk driver files for INIT parameters. These parameters are assembly language routines which initialize the disk surface, among other things. INIT then loads the drivers and parameters into memory, ignoring any device driver entries which lack INIT parameters, and continues by printing the message:

"Device <SYO:>?"

After you have entered a destination device name and inserted a disk, INIT tries to read the volume name, which is located on the tenth sector of the first track. If it finds a volume name, it prints it. If it does not find a volume name, INIT assumes that the disk has not been previously initialized. INIT then asks whether you really want to initialize the disk. If you type NO, INIT exits to the HDOS Command Mode Level. If you type YES, INIT asks for a volume number and label, which it stores in memory.

After you have entered a volume name and number, the initialization process begins. INIT erases the information on the first track, formats the track into ten 256-byte sectors, and then proceeds to the next track, repeating this procedure until all tracks have been erased and formatted. INIT then writes nine Bootstrap sectors onto the first track of the disk. It retrieves the volume name and serial number you entered from memory, and writes this data to the tenth sector of the first track.

INIT now asks for the numbers of bad sectors. As you enter the sector numbers, if applicable, INIT builds a map that indicates where the bad sectors are and then writes this map to the disk. INIT then finds 20 consecutive sectors and uses 18 of these to write the DIRECT.SYS file to the disk. DIRECT.SYS is a file that contains the directory which HDOS uses to locate files.

INIT uses the other two sectors to store a file called GRT.SYS, which HDOS uses to determine which sectors on the disk are not being used. After transferring DIRECT.SYS and GRT.SYS to the disk, INIT prints the message "Disk Initialization Complete."

Since HDOS uses DIRECT.SYS and GRT.SYS to locate and store your files, and since DIRECT.SYS and GRT.SYS are transferred only by means of the INIT program, you must initialize all new disks and disks which have been run through the TEST17, TEST37, or TEST47 procedures.
****

INIT OPTIONS

To run INIT, perform Bootstrap from SY0:, using a distribution disk or a system volume which contains the INIT program. You can run INIT from the HDOS prompt, in either of two ways:

The Standard Method:

NOTE: Within the following paragraphs, computer dialogue will be enclosed by quotation marks. The responses of the operator will be enclosed by apostrophe marks.

(1) If you simply type 'INIT<RTN>', the program will describe itself, and ask if you really want to proceed. If you type YES<RTN>, the program will dismount the disk(s) and continue. If you type NO<RTN> or simply <RTN>, you will return to the beginning of Bootstrap. If you have typed YES, it will ask which drive you want to use as the destination device.

The screen will display:

"Device: <SY0:>?"

If you have only one drive in your system, type <RTN> at this point. This will cause SY0: to be used as the destination drive. If you have a multiple drive system, you can type the name of one of your other drives. For example:

"Device: <SY0:>?"

'SY1:<RTN>'

In this example, the operator instructed INIT to use drive SY1: as the destination drive. The source drive for INIT is always SY0:.

The Shortcut Method:

(2) The second method of running INIT is to type INIT and then the name of the disk drive you want to use as the destination drive. For example:

'INIT SY1:<RTN>'

The effect of this example is to run INIT as usual, except that after dismounting the disks, INIT will proceed to instruct you to insert the volume you wish to initialize. Using this method of running INIT eliminates the need to wait for INIT to print the description of itself and the message asking whether you really want to proceed. Moreover, since in using this method you have already told INIT which disk drive you want to be the destination drive, INIT doesn't need to print the "Device <SY0:>?" message.

INIT OPTIONS (Cont)

Having determined which drive is to be used as the destination drive, INIT will ask you to insert the disk you wish to initialize. After you have inserted the disk and typed <RTN>. INIT will identify the volume, unless it is a brand new disk. It will then ask if you want to proceed, as follows:

"Type NO to cancel, type YES to erase and
Initialize the disk. (YES/NO)?"

If you type 'NO<RTN>', INIT will repeat the message which instructs you to insert the disk you want to initialize so you can insert another disk at this point. If you type 'YES<RTN>' in response to the message, the procedure will continue.

If you are initializing an 8-inch disk, INIT will ask whether you want the initialized disk to be double density. If you do want the initialized disk to be double density, type 'YES<RTN>', or simply '<RTN>'. However, if you want the disk to be single density, type 'NO<RTN>'.

INIT will now ask you for a volume serial number. This can be any integer between 0 and 65535. It is a good idea that your disks have a unique volume number.

When INIT asks you to enter a volume label, enter anything you like, as long as it is from 1 to 60 characters long. It is good practice to assign meaningful labels to your disk, such as SYSTEM VOLUME, DATA FILES, etc. In this way, the label will help you to determine which of your disks contains a given file. After you have entered the label and entered a '<RTN>', INIT will begin to initialize the disk.

The initialization process will take several seconds. When it is almost complete, you will be asked to enter the numbers of any bad sectors on the disk you are initializing. You will not be able to identify these sectors until you have run the M option of TEST17, TEST37, or TEST47, unless you are using a non-Heath device driver that adds a media check option to the INIT code. If you have run the appropriate test, or determined the bad sectors by an equivalent method, enter the address number of each sector and type a '<RTN>'. This will instruct INIT to "flag" the bad sectors so HDOS will not try to write to them. If the Media (M) test found no bad sectors, just type '<RTN>'.

INIT will again print the message instructing you to insert the disk you want to initialize. If you do have another disk to be initialized, insert it and type '<RTN>', and then continue. If you do not have any more disks to initialize, type 'CTRL-D' twice. Then INIT will ask if you have more disks to initialize. Type either 'NO<RTN>' or simply '<RTN>'. You will return to SYSCMD.SYS, HDOS Command Mode upon exit.

SYSGEN

The SYSGEN program is a stand-alone utility designed to generate the HDOS system. The effect of this process is to transfer essential HDOS files from the source disk to an initialized destination disk.

NOTE: Relative to the following text, when the computer prints on the screen, the message is set off by quotation marks. Computer user responses are set off by quotation marks.

When you type 'SYSGEN<RTN>', HDOS loads the SYSGEN program into a memory buffer and passes control to the SYSGEN program. SYSGEN then asks you for a destination device. If the destination device you specify is not SY0:, SYSGEN loads a disk driver for that drive. Then SYSGEN dismounts all disks, mounts the source disk, and, if the destination drive is not SY0:, SYSGEN mounts the destination disk. If the destination drive is SY0:, SYSGEN instructs you to insert the destination.

Having mounted the disk(s), SYSGEN copies the HDOS 3.0 system files:

```
HDOS30.SYS - the nucleus
SYSCMD.SYS - the command processor
TT.DVD ----- computer screen driver
PIP.ABS ---- the "handyman file"
SY.DVD ----- primary drives controller
```

If the source drive and destination drive are both SYn: drives, SYSGEN copies only the SY: device driver to the destination disk. If the destination drive is a DKn: drive, SYSGEN copies both disk drivers from the source drive to the destination, and renames them appropriately during the transfer. SYSGEN then sets a flag which identifies the disk as having been SYSGENed. It then copies the files which are contained in an internal list within the SYSGEN program. When all these files have been copied, SYSGEN prints the message:

"nn Files Copied"

and returns you to the HDOS Command Mode.

NOTE: When the following command examples provided in this section, contain the symbol [^], this indicates that a space must be inserted. Do not type the symbol [^] instead of the space.

****

SYSGEN OPTIONS

SYSGEN can be performed in three different ways:

(1) The standard way

The command:

'SYSGEN<RTN>'

SYSGEN OPTIONS (Cont)

This is the simplest fashion in which SYSGEN is entered. This version of the command will copy all of the system files and a few other important files to the destination disk. However, it is not always the best way to go, especially when there is a disk space limitation.

(2) The most efficient way

The command:

```
">"'SYSGEN^/MIN<RTN>' (NO SPACES!)
```

will initiate the SYSGEN program as usual, except that the SYSGEN program will copy only the MOST ESSENTIAL HDOS system files from the source disk to the destination disk. That is, the /MIN switch will cause SYSGEN to transfer only the minimum number of HDOS system files which are indispensable to the system. These files are:

```
 FILENAME      SIZE

HDOS30.SYS ---- 40
SYSCMD.SYS ---- 40
TT.DVD -------- 13
PIP.ABS ------- 49
SY.DVD -------- 20
```

If you SYSGEN to a drive other than an SYn: drive, SYSGEN will transfer both types of disk drivers and rename them appropriately.

The /MIN switch is useful if you want to store a large amount of data on a disk, and you also want to use that disk to perform Bootstrap. Note that "ERRORMSG.SYS" is not transferred, so all volumes created with this switch will show error codes (digits) instead of codes and messages.

(3) Copy all files

The command:

```
">"'SYSGEN^*.*/Q'  (NO SPACES)
```

will initiate SYSGEN and copy all the files from the source disk to the destination disk. Thus, using 'SYSGEN^*.*' will clone the source disk. Using the *.* switch eliminates the need to use ONECOPY to transfer non-system files such as BASIC.ABS and ASM.ABS. Using the /Q switch provides a means of accepting or rejecting given files to be copied, since all files are listed one by one and the operator can type either a 'Y' to copy the file, or a 'N' to reject the file.

After having been invoked by HDOS, SYSGEN will print the message: "Device<SY0:>?" At this point, you can either type '<RTN>' or you can type a disk drive name and '<RTN>'. If you simply type '<RTN>', HDOS

SYSGEN OPTIONS (Cont)

will use SY0: as both the source and destination drives. If you have only one drive, this is your only option. If you have a multiple drive system, you can avoid having to swap back and forth between source disk and destination disk, by typing the name of one of your other drives after the "Device<SY0:>? message. For example:

"Device<SY0:>?"'SY1:<RTN>'

In this example, the operator made SY1: the destination drive. The "Device<SY0:>?" message is the last opportunity to type 'CTRL-D' and exit from the SYSGEN procedure.

When SYSGEN asks you to remove the disk(s), you have two options: you can either use the disk from which you booted to perform SYSGEN, or you can remove the diskette from which you booted and insert another disk which contains all the necessary HDOS system files. Only the distribution disk and SYSGENed disks (except disks created using the SYSGEN/MIN switch) contain the SYSGEN program unless you manually copied the file SYSGEN.ABS to your disk using either the COPY command or PIP. If you want to SYSGEN from the disk you used to perform Bootstrap, do not remove the disk and simply type '<RTN>'. If you do want to replace the disk you used to boot the system, remove the boot disk, replace it, and then type '<RTN>'. If you do replace the disk, make sure that the name of the alternate disk driver is the same as the boot disk and the disk with which you replace the boot disk (i.e., both should be named SY.DVD or both should be named DK.DVD). It is good practice to boot from a disk that contains the SYSGEN program and the driver in order to avoid having to switch disks.

When SYSGEN is complete, the system will return you to the beginning of the Bootstrap procedure. After SYSGENing, perform Bootstrap using the destination disk. This will insure that the disk has been properly SYSGENed.

Remember that the first time you boot any new disk, HDOS will need to determine the disk baud rate. Therefore, you will need to press the space bar one or more times to get the disk to boot.

****

ONECOPY

ONECOPY enables you to copy files from one disk to another using only one disk drive. ONECOPY is useful if you have only one drive in your system. You have the option of copying one file, or multiple files in a single operation, as will be explained under "ONECOPY" in Chapter Two, "General Operation." Because ONECOPY accesses drive SY0: exclusively, it requires that you swap back and forth between a source disk and a destination disk. Unlike SYSGEN, ONECOPY cannot generate a usable system volume. This utility cannot link a set of programs together to be used as an operating system.

ONECOPY (Cont)

To use ONECOPY, (OC.ABS on the system disk) you need a system volume (the distribution disk or a copy of it) installed in SY0:. The diskette to which you will copy files must have been initialized by means of the INIT program.

ONECOPY is a stand-alone utility. This means that HDOS writes the ONECOPY program into a memory buffer and then runs the program without accessing the disk which contains the utility.

When you specify source files, at the OC: prompt, the ONECOPY program instructs you to insert a source disk, and then searches the disk you insert to insure that your source disk contains the specified source files. If the source disk does contain the files you have specified, ONECOPY loads the contents of the files into a memory buffer and then asks you to insert the destination disk. The program will then write the contents of the buffer onto the destination disk, repeating this process until all files you have selected have been copied.

NOTE: Within the following paragraphs, statements made by the computer are set off by a pair of quotation marks, while responses by the user are set off by a pair of apostrophes.

To copy single files using ONECOPY, use the format:

"OC:"'FNAME.EXT,FNAME.EXT<RTN>'

Note that you can specify only one, or many filenames in this format:

"OC:"'*.EXT,*.EXT<RTN>' or

"OC:"'FNAME.EXT<RTN>'

Practical Example: "OC:"'BASIC.ABS.*,ERRORMSG.SYS<RTN>'

or, to copy all the files on the disk:

"OC:"'*.*<RTN>'

Like PIP, ONECOPY recognizes the /VER, /L, /L,S, /S, and /B,S switches.

As in PIP, ONECOPY utilizes the /MOU switch, although the effect of OC /MOU differs from PIP /MOU. /MOU makes it possible to switch disks whenever the OC: prompt is displayed. Thus:

"OC:"'/MOU<RTN>'

"Insert New Disk"

At this point, you should insert an initialized disk and type a <RTN>. The OC: prompt will again be displayed:

"OC:"

ONECOPY (Cont)

The new disk is now the source. You may specify any number of files to be copied from it, and you may use any of the switches to obtain file listings. If you wish to switch to yet another disk, type /MOU again, remove the disk, and insert another. Note that ONECOPY^/MOU both dismounts and mounts a disk, while the PIP switch, PIP^/MOU serves only to mount a disk. In this sense, OC^/MOU corresponds more closely to PIP/RES than to PIP/MOU.

****

ONECOPY OPTIONS

When you are using ONECOPY with either the distribution disk or a system volume which is a duplicate of the distribution disk, the valid file choices under ONECOPY are:

```
SYSGEN.ABS
INIT.ABS
BASIC.ABS
SY.DVD
DK.DVD
```

Exactly which files you select will depend upon your requirements. If you type 'OC^*.*', all files in the preceding list will be copied from the distribution disk or system volume to the destination disk.

To exit from the OC: prompt of ONECOPY, type 'CTRL-D'.

****

PIP NOTES

The PIP.ABS utility program has been substantially modified for HDOS 3.02. For details, refer to Chapter 5, "PIP/PLUS."

****

CONCATENATION

Concatenation is the process of joining two or more small files into one.

When multiple file designations are used with the CO[py] command, or with a copy command within PIP/Plus, the result will be a file that is a combination of the files which are specified on the right-hand side of the = symbol. An example which utilizes the files listed in the previous example follows:

```
">"'COPY^BIGFILE.DOC=NEWFILE.DOC,ERRORMSG.SYS<RTN>'
```

The result of this command is a file on SY0: called BIGFILE.DOC, which is a concatenation of NEWFILE.DOC and ERRORMSG.SYS. The following example illustrates concatenation using the PIP/Plus copy format:

```
"P:"'BIGFILE.TXT=NEWFILE.DOC,ERRORMSG.SYS<RTN>'
```

NOTE: You must use a comma between each filename and have no intervening spaces on the command line.

****

WILDCARDS

Multiple Listing:

There are two ways to manipulate more than one file with the same command. The simplest way is to use more than one filename in the source or destination fields of the commands. For example:

```
'TYPE NEWFILE.DOC,ERRORMSG.SYS<RTN>'
"THIS IS A TEST"
"128 CTRL-C Struck"
"129 CTRL-B Struck"
"130 Data Exhausted"
"Type a CTRL-C"
"Error Message No. 1"
etc.
```

The contents of both files are printed on the console. Typing CTRL-C terminates the output.

Using Wildcards:

A wildcard is a simple expression for grouping a number of files that have identical filenames or extensions or identical parts of filenames. Wildcards are used to manipulate files quickly and with the least possible effort. HDOS allows the use of two main types of wildcards. Option 1: *.* and Option 2: ?????. Details are provided in the following paragraphs.

****

WILDCARD OPTION 1

(Wildcard Example: *.*)

This option allows you to manipulate files by groups based upon some common characteristic. For example, you may want to group all the EDIT19 files and copy them off to another disk with one command. This command may be phrased like this:

```
">"'CO SY2:=SY1:EDIT19.*.*<RTN>'
```

Therefore, the *.* wildcard is yet another way of accessing multiple files. It can be used with CAT, COPY, and DELETE.

DVn:*.EXT or DVn:FNAME.* or DVn:*.*

The effect of the wildcard option is to permit you to access all files that have the unmodified portion of the filename in common. HDOS recognizes *.DOC, for example, as a command to search for all files that match the format *.DOC, in other words, any filename (*.) with an extension of .DOC. The following example illustrates the use of a wildcard to obtain a directory listing:

```
">"'PIP<RTN>'
"P:"'*.SYS/B/S<RTN>'

   GRT.SYS          HDOS30.SYS     SYSCMD.SYS
   ERRORMSG.SYS     RGT.SYS        GRT.SYS        DIRECT.SYS
```

This is a brief listing of the essential system files. Note that all the filenames have the same .SYS extension, while all the primary filenames are unique.

You can substitute the * into either the FNAME or the .EXT field, and you may use * in both the FNAME and .EXT fields. Thus, *.* is a valid command which signifies that you want to manipulate all the files on the disk, since all files match the *.* reference.

The *.* command is particularly useful for copying files within ONECOPY. If you type *.* after the OC: prompt, ONECOPY will COPY all non-system and system files (unless the system files are hidden with an S flag) from the system volume to another disk.

If you have a multiple drive system, you can transfer many files between any two drives in one operation. First, mount an initialized disk in the drive to which you want to transfer files, and then, using either COPY or PIP/Plus, specify drive names in the format:

">"'DVn:*.*=DVn:*.*' or ">"'CO SY1:*.*=SY0:*.*'

or in HDOS 3.02, simply:

">"'CO^SY1:*.*=SY0:'

Note the lack of filename for the destination.

WILDCARD OPTION 1 (Cont)
(Wildcard Example: *.*)

In either case, all files will be transferred from the source drive to the destination drive. HDOS 3.02 is unique in this regard.

If you are copying files between drives, the *.* wildcard enables you to transfer a given file in such a way that the copy that is transferred will have the same name as the original, as is illustrated in the following example:

```
">"'CO^SY1:*.*=SY0:BASIC.ABS<RTN>'
```

The effect of the preceding command is to create a file on SY1: named BASIC.ABS. Of course, you could have produced the same effect by typing:

```
">"'COPY^SY1:BASIC.ABS=SY0:BASIC.ABS<RTN>'
```

The * wildcard may also be used to rename files during the copying process. This is the abbreviated method which is quicker than the long way to accomplish the same thing. An example of the long way is:

```
">"CO^DK1:*.TXT=SY0:NEWFILE.DOC<RTN>
```

The effect of this example was to copy NEWFILE.DOC to a file on DK1: called NEWFILE.TXT. The next example illustrates the use of wildcards to rename multiple files in a multiple drive copy operation:

```
">"'CO^SY2:*.TXT=DK0:*.DOC<RTN>'
```

In this example, the operator copied all files on the disk mounted in DK0: that have the .DOC extension to the disk in SY2:, and assigned the extension .TXT instead of .DOC to the new files in SY2:. The FNAMEs of the files on DK0: were preserved during the transfer.

You can also copy files using a combination of both wildcards and multiple designations, but you may only designate multiple source files. Thus, the following is a valid combination of wildcard and multiple file designation:

```
">"'CO^SY1:*.*=HELP.DOC,ERRORMSG.SYS<RTN>'
```

While the following is an invalid combination of wildcard and multiple file designation:

```
">"'CO^SY1:*.CAT,*.DOG=HELP.DOC,ERRORMSG.SYS<RTN>'
```

This second example is invalid because more than one file was specified in the destination field of the command.

The fundamental formula to use for copying files is: DESTINATION = SOURCE.

****

WILDCARD OPTION 2

(Wildcard Example: ????????.???)

You can use another type of wildcard as a substitute for letters in a portion of a filename. This wildcard is the "?." Since the FNAME portion of a filename may be up to eight characters in length, and the .EXT portion may be up to three characters, the wildcard ????????.??? is the same as *.*.

If you use "?" in the FNAME portion of a file designation, you must use at least as many "?" marks as there are characters in the name of the file you want to manipulate. Thus:

```
">"'C[at]????.ASM<RTN>'
```

will list those .ASM files whose FNAME field contains four characters or less.

The "?" can be used along with the "*" wildcard. For example, if you had several files on the disk in SY0: called CHAPTER1.DOC, CHAPTER2.TXT, and CHAPTER3.DOC, the command.

```
">"'C[at] CHAPTER?.*<RTN>'
```

would list all three of these documents.

Note that you can use the C[at] command and the /L and /S switches with multiple file designations and wildcards, but you may not refer to more than one device in the same command. The next example is INVALID, and will cause an error message:

```
">"'C^SYSHELP.DOC,SY1:HDOS.ACM<RTN>'
```

This example is INVALID because both SY0: and SY1: are used in the same CAT command. Also, the space after the comma on .DOC and before SY1: should be deleted when using this form. The following is a VALID use of CAT with a multiple file designation:

```
">"'C^SYSHELP.DOC,HDOS.ACM<RTN>'
```

There is also a restriction on the use of the *.* wildcard with C[at]. If you type C[at]*.*<RTN>, the computer will produce a listing of all non-system files. If you want to list all files on the disk using the *.* wildcard with C[at], you will have to use the /S modifier to override the S flag on the system files, as in the following example:

```
">"'C^DK2:*.*/S<RTN>'
```

This may also be used with a partial filename. For example:

```
">"'C^SY1:BOZO.*<RTN>'
```

****

SET - SYSTEM OPTIMIZATION

SET (System I/O Configuration)

The SET command is used to configure your system for the particular input and output devices that you have. Input and output are abbreviated as "I/O." For example, you can use the SET command to specify how many characters your terminal can handle on one line, to set the step rate of your disk drives, to set the baud rate of your line printer, and so on. The general form of the SET command is:

">"'SET^DVn:^OPTION<RTN>'

You can obtain this general command format for SET from the command

mode by typing:

">"'SET^HELP<RTN>'

Or, if you want to know the possible SET options for a given device, type:

">"'SET^DVn:^HELP<RTN>'

For instance, if you wanted to determine the possibilities for optimizing the configuration of your line printer, type:

">"'SET^LP:^HELP<RTN>'

Once you have SET a given option, HDOS writes it on the disk so you do not have to re-SET the option each time you reboot the system. However, your initial configuration is not indelible; unless you set the "W" (write protect) flag on the device driver file after using SET. Therefore, if you alter your hardware - or for any reason at all - you can re-SET the option. Any changes you make by means of SET remain in effect until you reuse the command.

IMPORTANT NOTE: You MUST reboot your computer system in order to incorporate the changes you make with SET.ABS.

.....

SET HDOS STAND-ALONE

Within HDOS 3.0, there is no separate command to SET HDOS STAND-ALONE. All disks created by the HDOS 3.0 system are automatically set to stand-alone.

SET HDOS STAND-ALONE<RTN> is a command that applies only to HDOS 2.0.

.....

SET - SYSTEM OPTIMIZATION (Cont)

Like PIP/Plus, the SET program has been assigned a version number, which you can display by entering:

```
">"'SET^VER<RTN>'
```

The tables that follow summarize the SET options, default values for the options, and the devices to which the SET command applies. Table A lists all HDOS devices. Table B lists the SET options for HDOS; Tables C through K list the options for each device. The options that are preset on the distribution disk are marked with an asterisk [*]. To make the most of this information, you will probably want to refer to the "Peripherals" section, which follows immediately.

Table A
HDOS DEVICES

Device Name	Description
SY:	Handles system disk drives (primary boot).
DK:	Handles alternate disk drives (secondary boot).
TT:	Handles console terminal, input and output.
LP:	Handles line printer.
AT:	Handles alternate terminal.

Table B
SET HDOS OPTION

Option	Description
HELP	Prints SET HDOS options.
*DATE	User prompted for date at boot-up.
NODATE	Suppress date prompt at boot-up.

NOTE: "NODATE" files created under HDOS Version 3.0 cannot be cataloged under previous versions of HDOS.

SET (Cont)
(System I/O Configuration)

Table C
SET DRIVE SYn: OPTION
SET DRIVE DKn: OPTION

STEP n Sets the speed at which any 5-1/4 inch primary boot drive steps between tracks on the disk. Step time for all 5-1/4 inch SYn: drives is set using this command. The step time for 8-inch drives is preset. Use TEST17 to determine the value of "n:." It should be between 16 and 30 for H17 (hard sector) disk drives. The H37 controller (soft sector) allows only the selection of 6, 12, 20, or 30 milliseconds for this parameter. The seek time of the slowest drive is the fastest seek time you can use for all drives on that branch of your system, for example: H17. The HDOS distribution disk is factory-set at a step rate of 30 milliseconds. If you re-set the seek time of your disk drives, choose values in increments of "2" for
the H17 type drives.

Table E
SET TT: OPTION

Option	Description
HELP	Prints the SET options for TT:.
* NOBKS	Tells HDOS that your terminal cannot backspace. If you attempt to delete mistyped characters, HDOS will print the deleted characters between back slashes [\] instead of allowing you to back up over them and erasing them.
BKS	Enables backspacing to correct typing errors. The cursor backs up to the character being deleted, and it is erased from the screen.
* BKM	Causes BACKSPACE (CTRL-H) to be treated as DELETE.
NOBKM	Lets HDOS receive the BACKSPACE character.
* MLI	Maps (changes) lower case input to upper case.
NOMLI	Allows lower case input to HDOS.
* MLO	Maps (changes) lower case output to upper case.
NOMLO	Allows lower case output from HDOS.
* NOTAB	HDOS expands TAB (CTRL-I) into spaces.
TAB	Lets the terminal process TABs faster!

SET (Cont)
(System I/O Configuration)

Table E
SET TT: OPTION (Cont)

* 2SB	Uses two stop bits. (Universal)
1SB	Uses one stop bit. (Normal)
WIDTH nn	Sets the terminal screen width to nn characters. The default value is 80. HDOS starts a new line if more than nn characters are sent to the screen. The value of nn must be SET between 20 and 255. It is best to SET this unit to 255 to accommodate long programming lines.
FILL c n	Sets "c" as the ASCII code for the character which needs "n" null characters inserted after it. Fill is usually needed after a carriage return (13) on some slow, hard-copy terminals. This allows time for the completion of the return motion before the next characters are printed.

With most terminals, you should SET the following options:

```
SET^TT:^1SB             SET^NOMLI
SET^TT:FILL 13 0        SET^NOMLO
SET^BKS                 SET^TAB
SET^WIDTH 225
```

* The asterisk indicates a preset option from the distribution disk.

NOTE: The following tables, F through K, pertain to an array of printer drivers provided by HDOS 3.0 to implement the specified printers. They will work in HDOS 3.0, but probably not in HDOS 2.0. These drivers are available on the HDOS 3.0 Distribution Disks. For details, refer to Table L on page 3-27.

SET (Cont)
(System I/O Configuration)

Table F
AT84.DVD (Serial)
AT85.DVD (Serial)

Options for A Serial Alternate Terminal With H8-4 Interface.

Options for a Serial Alternate Terminal with H8-5 Interface.

Option	Description
HELP	Prints the SET options for AT84.DVD.
HELP	Prints the SET options for AT85.DVD.
1SB	SETs for one stop bit.
2SB	SETs for two stop bits.
MLC	SET for changing lower case to upper case.
NOMLC	SET for providing both upper and lower case.
WIDTH n	SET for setting "n" characters for the width of the printer. If more than "n" characters are sent, a new line is started. The range of "n" is 0 thru 132. If zero is used, the new line feature is disabled. The default is 132.
PAD n	Sends "n" pad characters after a carriage return. Pad is needed on some slow, hard-copy terminals. This allows time for completion of the return before the next characters are printed. Default is zero.
PORT n	SETs the port address to "n." Default value of "n" is 320Q.
BAUD n	SETs the baud rate. You must set only standard rates, such as 1200, 2400, 4800, etc. Default value is 300.

SET (Cont)
(System I/O Configuration

Table G
Options for H1484.DVD (Serial)
Options for H1485.DVD (Serial)
Heath H14 Printer

Option	Description
HELP	Prints the SET options for H1484.DVD (serial).
HELP	Prints the SET options for H1485.DVD (serial.
6LPI	SETs the H14 Line Printer for 6 lines per inch.
8LPI	SETs the H14 Line Printer for 8 lines per inch.
PAGE n	SETs the number of lines per page to "n." If "n" is zero, lines are printed continuously. Default value is 60.
PORT n	SETs the port address for LPH14 to "n." Default value of "n" is 340Q.
WIDTH m,n	SETs the Width Control switch position. "n" is the narrow position, and "m" is wide. The only legal values are 80, 96, and 132. The default setting is "n" = 80, "m" = 132.
BAUD n	SETs the baud rate for the LPH14. You must set only standard baud rates, such as: 1200, 2400, 4800, etc. Default is 4800.

PERIPHERALS

Device Drivers for HDOS 3.0

To facilitate expansion and maintenance of the system, HDOS was designed in a modular fashion; e.g., a number of subprograms that communicate with one another. Each of these subprograms is responsible for a logically distinct task. For example, the subprogram which processes commands is separate from that which processes I/O. When the command processor needs input, it asks the I/O handler for data. Likewise, when the command processor generates output, it passes the data along to the I/O handler. To compare HDOS to a person, one might liken the command processor to the brain, the external peripherals to the sense organs, and the I/O handlers, such as device drivers, to the nerves which translate and transmit data from the sense organs to the brain.

HDOS does not directly communicate with peripherals. Rather, it communicates with peripherals indirectly by means of device drivers. Any device that is to be interfaced to HDOS must therefore be interfaced by means of a device driver. When HDOS writes to device XX:, it merely supplies the bytes to be written, invokes the driver, and relies upon the driver to convert the data into the format required by the specific device. Thus, only device drivers "know how to talk to" peripheral devices. Since device drivers are not inherent parts of the operating system, HDOS I/O is quite flexible.

To manage the various devices in the system, HDOS maintains a table which supports a virtually unlimited number of device drivers. The Device Table size is dynamically determined at boot time. The two mandatory device drivers are SY.DVD and TT.DVD. SY.DVD is used to control the primary disk drive units, while TT.DVD controls the terminal and screen.

TT.DVD is no longer an integral part of HDOS, but is an independent device driver. It transfers to a disk being made bootable at SYSGEN time, as does SY.DVD. In addition to the standard device driver entry points, TT.DVD includes routines to process the following SCALLs: .SCIN, .SCOUT, .PRINT, .CONSL, and .CLRCO. TT.DVD also supports higher baud rates up to 38400 baud.

Whenever the system is booted, HDOS scans the disk directory for entries in the form of "xx.DVD." Then uses these entries to build a device driver table. The size of this device driver table is dependent upon how many such entries are found. For example, a disk freshly SYSGENed would normally only have two device drivers on it: SY.DVD and TT.DVD. Therefore, the device driver table would be composed of only two entries. If you copy 13 more device drivers to that bootable disk, the next time the disk is booted the table size changes to 15 entries.

Once HDOS 3.0 has entered the device drivers in the Device Table during bootup, they remain usable as long as the current system disk is mounted in SY0:, or unless you wish to unload them. In HDOS 3.0, device drivers may be cleanly unloaded. Drivers may also be loaded and/or loaded and locked in memory.

PERIPHERALS (Cont)

Device Drivers for HDOS 3.0 (Cont)

For example, the command:

">"'UNL[oad] xx.DVD<RTN>'

unloads device driver xx.DVD.

Also, the command:

">"'L[oad] xx.DVD<RTN>'

loads device driver xx.DVD,

and the command:

">"'FLO[ad] xx.DVD<RTN>'

both loads and locks device driver xx.DVD in memory.

Device Driver Structure

Device driver filenames must be only two characters long, and the extension must be .DVD. The two-character filename is used to define the device name. For example, the driver of your primary line printer should be called "LP.DVD." You may also have secondary line printer drivers on the disk, such as "EP.DVD" and "UD.DVD." If you see more than one line printer driver on a bootable disk, you must know that each of the drivers have different characteristics.

If you intend to program device drivers in HDOS 3.0 the "SET" part of the driver may be larger than 2 sectors. It may extend to 16 sectors in multiples of 2 sectors.

HDOS 3.0 Supplied Device Drivers

HDOS 3.0 provides a number of general-purpose device drivers on the distribution disks. These device drivers are designed to accommodate both Heath and non-Heath peripherals, and are as follows:

PERIPHERALS (Cont)

TABLE L: DEVICE DRIVERS

Printer Drivers:

AT84.DVD	An alternate terminal configured at port 320Q via an H8-4 card for the H8 computer, or port 320Q for the H89 computer. (Serial)
AT85.DVD	An alternate terminal configured at address 374Q via an H8-5 card for the H8 computer. (Serial)
H1484.DVD	A Heath H14 Printer driver for an H-4 interface. (Serial)
H1485.DVD	A Heath H14 Printer driver for an H8-5 (H8 Computer). (Serial)
H2484.DVD	A Heath H24 Printer (TI-810 equivalent) driver for an H8-4 interface. (Serial)
H2584.DVD	A Heath H25 Printer driver for an H8-4 interface. (Serial)
H4484.DVD	A Diablo H44 Printer driver for an H8-4 interface. (Serial)
MX8084.DVD	An Epson MX-80 printer driver for an H8-4 interface. for Port 340Q. (Serial)
MX8011.DVD	An Epson MX-80 printer driver for an Z89-11 interface. (Parallel)

Other Drivers:

TT.DVD	The console device driver. Also processes all terminal-related SCALLs.
ND.DVD	Null device, often referred to as the "bit bucket."
H17.DVD	H17 device driver. (Hard-sector)
H37.DVD	H37 device driver. (Soft-sector)
H47.DVD	H47 device driver. (8-inch)
IOMEGA.DVD	Bernoulli Box device driver. CAUTION: See source code before using this driver.

PERIPHERALS (Cont)

Incorporating Device Drivers

The following paragraphs outline the procedure you would use to incorporate any of the drivers supplied, except ND.DVD, which requires no software configuration.

1. Decide which device drivers you will need. Keep in mind that the maximum number of user-selectable drivers is nearly unlimited. For instance, assume that an H14 line printer is to be included in your system. In such a case, you would have to select either H1484.DVD (parallel) or H1485.DVD (serial) as your device driver.

2. Configure your hardware to match the specifications of the selected device driver. (Refer to Chapter 2, SET STEP 5, for details.)

3. Copy the device drivers to the appropriate two-letter filename. For example, to utilize H1484.DVD and modify its filename to DB.DVD:

```
'REN[ame] DB.DVD=H1484.DVD<RTN>'
```

4. Enter a BYE command, and then reboot HDOS. It is necessary that you reboot because the HDOS Device Table is built only upon boot up. If you rename H1484.DVD to DB.DVD and then immediately try to use DB: or LP: without rebooting the system, HDOS will not recognize any commands to either of them, since there was no LP: or DB: in the directory when HDOS last built the Device Table. For the same reason you will not be able to use the SET command to configure the new device driver until it has been given a 2-letter name and HDOS has been re-booted. The use of a printer driver such as UD.DVD (configured for HDOS 3.02) is an exception. No boot is necessary.

By way of example, an illustration implementing DB: with a Diablo printer is given below:

```
'REN[ame] DB.DVD=H4484.DVD'
'BYE<RTN>'

"Volume 100, Dismounted from SY0:
Label: System Volume"

"System Down"  To reboot hit SHIFT/RESET<RTN>

"HDOS 3.0
 ISSUE # 50-07-00"

"System has 64k RAM"

"Drivers found: TT: SY: DK: LP: DB:"

"Date <10-Mar-90>?"'16<RTN>'
```

PERIPHERALS (Cont)

Incorporating Device Drivers (Cont)

"Time (00:00:00>?"'15:46:50<RTN>'

"S:"

From now on, the new device will appear as part of the operating system. You may now use the device drivers LP: and DB: throughout HDOS. You can SET options, send data to it for printing, and copy it to other disks. Note: When you copy a device driver, its SETTINGS will be the same on both disks.

****

PATCH

NOTE: An absolute binary program is one with .ABS in its filename. SYSPATCH.ABS is an HDOS utility used to patch absolute binary programs. SYSPATCH may be used to patch user-written programs which have not been write protected. SYSPATCH can modify any Heath-supplied system programs, but it will not patch assembly source programs or BASIC programs. See the comments below.

You can PATCH locations in your program that are not defined within the program, but those locations must follow the current last-word address of your program. Thus, if your program occupied locations 042200 to 050000, you could extend the program by entering PATCHes to locations greater than 050000.

To use SYSPATCH:

1. Run SYSPATCH, using the command format RUN 'DVn:SYSPATCH', or simply 'DVn:SYSPATCH'.

2. SYSPATCH will prompt you for a filename. Enter the device name and the name of the binary file you wish to PATCH. For example:

'SY1:DEMO2.ABS<RTN>'

3. SYSPATCH will now prompt you with "ADDRESS?" Enter the address of the first PATCH as a byte-octal number. For example:

"ADDRESS?"'042200'

4. SYSPATCH will then display an address and byte value, followed by a backslash [\]. You can reply in one of three ways:

   A. Type a 3-digit new value.

   B. Type '<RTN>' to leave this byte unchanged.

   C. Type 'CTRL-D' to bring back the "ADDRESS?" prompt.

5. When you have finished PATCHing your program, type 'CTRL-D' in response to the 'ADDRESS?' prompt. SYSPATCH will then insert the PATCHes into your program.

NOTE: Data to end of this section refer to HDOS 2.0 PATCH.ABS. It was included to provide similar examples for using SYSPATCH.

Though originally thought able to patch only non-Heath programs, after publication of the HDOS 2.0 source by Heath, HUG's software genius, Pat Swayne, discovered some of the tricks to use PATCH on system files and Heath-written device drivers. Details may be found in REMARK #28 (May, 1982), where assembly source for a program for removing ALL flags is presented. However, essential information is also given in other places.

PATCH (Cont)

REMark #19, August 1981, is the first real break in the dam blocking information on the subject. Pat Swayne discusses methods of reducing the size of the system and utilities by eliminating the patch history sector at the end of EDIT, PATCH, INIT, SYSGEN, TEST47, ASM, XREF, DEBUG, PIP, HDOS.SYS, and HDOSOVL1.SYS. He gives assembly source for a program to accomplish this.

And beginning in REMark #27, April 1982, Pat presents a series of patches, including the patch "ID's," "prerequisite codes," and "check codes" necessary for a number of system files. I refer you to this material for specifics, but will include a summary, here, of which, what, and where. If you have eliminated the patch history sectors of these programs with Pat's REDUCE, you shouldn't need the special codes. And most of these are for HDOS 2.0, but a few are for 1.6, so check the references. NOTE: The following list provides examples, but is not applicable to HDOS 3.02!

Program	Purpose of Patch	REMark Issue
ASM.ABS	Fixes bug when making XREF listing	#44 Sept 83, p. 39
BASIC.ABS	FREEZE & UNFREEZE only program	#29 June 82, p. 7
	Permits CHR$(0) & chars > CHR$(127)	#42 July 83, p. 59
DEBUG.ABS	Corrects bug in STEP command	#29 June 82, p. 28
EDIT.ABS	Corrects bug in CTRL-C processing	#29 June 82, p. 28
HDOSOVL0.SYS	Corrects bug in .NAME SCALL processor	#30 July 82, p. 35
LPMX80.DVD	To use GRAFTRAX option	#30 July 82, p. 32
ONECOPY.ABS	Finish copying w/ destination mounted	#45 Oct 83, p. 72
PATCH.ABS	Ignore patch history on W-flagged files	#28 May 82, p. 36
PIP.ABS	List filenames when copying/deleting with wildcards	#27 Apr 82, last ad page
	Similar patch for ver. 1.6	#29 June 82, p. 28

A number of patches for other HUG and Heath software are also bracketed by these issues.

NOTE: SYSPATCH does not PATCH the program until the entire series of PATCHes has been entered and CTRL-D has been typed in response to the ADDRESS? prompt. Until that time, you may use CTRL-C or CTRL-Z CTRL-Z to cancel the patch session and leave your file unchanged.

****

NONESSENTIAL FILES

If you have a computer system with H37 or H47 drives, you will want to keep all of the files on your system disk that you use regularly. We suggest that you delete all of the device driver files except the ones that support your printer and disk drives. For example, in the typical computer system with H37 and H17 drives, essential device drivers are as follows:

SY.DVD DK.DVD TT.DVD
LP.DVD UD.DVD

NONESSENTIAL FILES (Cont)

If you have a computer system with H17 drives only, there may be some files that you want to delete. Do NOT, however, delete any system files, or you may not be able to use your system disk. Files that are candidates for deleting from your system disk copies would include:

OC.ABS
HELP.
SYSHELP.DOC

Since the functions provided by the programs residing on these files will no longer be available on this particular disk, we strongly recommend that you keep at least one master system volume, in addition to the abridged system disk. At any rate, even though HDOS provides built-in safeguards such as write-protection, the effect of incidental common menaces such as dust, extremes of temperature, and power outages, not to mention "operator error," can easily wipe out much tedious labor. For this reason, you should always keep backup copies of your own important files as well as the HDOS system files.

It is too easy to damage a disk. In one careless moment, or if you get distracted, a week's work may be lost. Even if there is an expert in your area who knows the HDOS system sufficiently so that he can apply a disk editor to the disk, even then not all files may be recovered. The only answer is to BACK UP your work -- not daily -- but whenever you complete an important file.

** BACK UP ** BACK UP ** BACK UP ** BACK UP ** BACK UP ** BACK UP **
****

SUMMARY

Your system should now be configured for your terminal and any peripherals, and the drive seek times should be optimized for your drives.

The examples used throughout this procedure are only a small sampling of the possible commands and functions of HDOS. By varying these examples, you will acquire "hands on" experience. Experimentation can cause no damage, thanks to the error-detection and write-protection facilities of HDOS. If you should accidentally delete or damage a file, you can always re-SYSGEN from the distribution disk. Therefore, do not be timid about exploring and enjoying the capabilities of HDOS. In case you type a command wrong, HDOS prints an error message on your screen. This is the worst that will happen. It is recommended that experiments be done on expendable copies.

Refer to Appendix 3-A: for a list of HDOS 3.02 error messages.
****

APPENDIX 3-A: HDOS 3.02 SYSTEM ERROR MESSAGES

This section describes the error messages generated by the HDOS 3.0 Operating System. Error messages fall into two general categories: those which start with ?nn, where nn = two digits, and those which don't. Error messages with no ?nn are produced by the program you are currently running. For example, if you are using the text editor, EDIT, and get a message with no ?nn in it, look in the text editor part of the manual for an explanation. Messages with ?nn in them are produced by some component of the HDOS operating system, and are discussed here. The messages are grouped together according to their ?nn number.

?00 - Bootstrap Errors

Error messages which start with ?00 are generated by the system while it is being booted up.

?00 DISK READ ERROR DURING BOOT

An unrecoverable (hard) disk error occured during the bootstrap process. Try booting again. If the problem persists, either your drive or your disk is bad. Try booting on a different disk drive or with a different bootable disk.

?00 * ERROR * nnn (sector number)

An unrecoverable hard-disk error occured while checksumming the disk. The sector number printed immediately after this message is the one containing the error.

?00 REQUIRED FILE HDOS30.SYS MISSING

The file HDOS30.SYS is not on the volume in SY0:. The disk has not been SYSGENed, or has been SYSGENed incorrectly. Reinitialize it and then SYSGEN it correctly.

?00 THIS DISK HAS NOT BEEN PROPERLY SYSGENED

Some error in the format of the HDOS system files was detected. The disk cannot be booted. The disk must be reinitialized, and then SYSGENed. If it contains valuable data, boot from a different disk and copy the useful data from the defective disk first!

?00 THIS DISK MUST BE INITIALIZED AND THEN SYSGENED BEFORE IT CAN BE USED

This disk must be initialized before you can use it. This message normally appears when you try to boot up a new disk that has not yet been initialized, attempt to mount a CP/M disk under the HDOS Operating System, or attempt to boot a disk that has been destroyed.

APPENDIX 3-A: HDOS 3.02 SYSTEM ERROR MESSAGES (Cont)

?00 - Bootstrap Errors (Cont)

?00 THIS DISK MUST BE SYSGENED BEFORE IT CAN BE BOOTED

This disk has not been SYSGENed, and thus cannot be booted as a system disk. Use SYSGEN to make it a system disk.

?01 - Build Phase Errors

These error messages appear during the second half of the boot process when the HDOS operating system is being built into memory from the system disk. Most of these messages indicate damage to the files on disk. First, try rebooting the system. If the problem persists, then this disk cannot be booted as a system disk. If you own two disk drives, mount the disk in SY1: and copy the files you want to keep onto a different disk. If you own only one disk drive, use ONECOPY (after booting up on some other disk) to copy off your important files. Then, reinitialize the disk and reSYSGEN it.

?01 DISK I/O ERROR DURING BOOT

An unrecoverable (hard) disk error occurred on the system disk, and the boot operation cannot proceed. The disk volume may be bad, or you may have a bad drive. Retry the boot.

?01 DISK STRUCTURE IS CORRUPT

The directory and/or the free space table on this disk is damaged, and HDOS cannot restore the damaged files. CAUTION: Do NOT attempt to contact Heath Technical Services, as this help is no longer available. NOTE: It may be possible to run a program such as CRASH.ABS from Software Wizardry or SUPERZAP from the Software Toolworks to restore the disk to a usable form, or at least recover some of the files.

?01 FORMAT ERROR IN DRIVER FILE

The file does not contain a valid driver program.

?01 HDOS REQUIRES AT LEAST 24K!

Your system does not contain enough RAM to run HDOS, or the RAM is faulty, or it is not addressed correctly. Use a memory diagnostic to make sure that the RAM is working properly, and is jumpered to the correct address.

?01 SYSTEM NOT SYSGENED PROPERLY, OR FILES DAMAGED

A system file is damaged. This can be the result of a software or hardware error.

APPENDIX 3-A: HDOS 3.02 SYSTEM ERROR MESSAGES (Cont)

?01 - Build Phase Error (Cont)

?01 UNABLE TO MOUNT SYSTEM DISK

The system volume from which you are attempting to boot up does not contain the file SY.DVD, the system disk driver.

?02 - Error Messages

These messages are generated by the operating system and may appear at any time. Usually they are in response to some request from the program you are running which, in turn, is usually caused by some command from you. Normally, HDOS looks up these error messages in the file SY0:ERRORMSG.SYS to give an understandable message. If the file SY0:ERRORMSG.SYS is missing, or if the system disk has been dismounted, HDOS will simply type the error message number. The numbers are listed first, followed by the message they represent. Look up the message in the second group for a discussion of its meaning.

Most of the error messages will be meaningless to you. They are generated by HDOS when a program makes a mistake when issuing a request to HDOS. Normally, only users debugging assembly programs will see most of these error messages. The ones that the average user will see are self-explanatory.

?02 SYS ERROR # 000

Heath/Zenith HDOS 3.02

?02 SYS ERROR # 001

End of File.

?02 SYS ERROR # 002

No Free Space on Media.

?02 SYS ERROR # 003

Illegal "SYSCALL" Function Code.

?02 SYS ERROR # 004

Channel Is Already in Use.

?02 SYS ERROR # 005

Device is Not Capable of This Operation.

APPENDIX 3-A: HDOS 3.02 SYSTEM ERROR MESSAGES (Cont)

?02 Error Messages (Cont)

?02 SYS ERROR # 006

Illegal Format for Device Name.

?02 SYS ERROR # 007

Illegal Format for File Name.

?02 SYS ERROR # 008

Not Enough Memory for the Device Driver.

?02 SYS ERROR # 009

Channel is Not Open.

?02 SYS ERROR # 010

Illegal Function Request.

?02 SYS ERROR # 011

File Usage Conflicts.

?02 SYS ERROR # 012

No Such File(s).

?02 SYS ERROR # 013

Unknown Device Name.

?02 SYS ERROR # 014

Illegal Channel Number.

?02 SYS ERROR # 015

The Volume Directory is Full.

?02 SYS ERROR # 016

Illegal File Contents.

?02 SYS ERROR # 017

Not Enough RAM for this Program.

APPENDIX 3-A: HDOS 3.02 SYSTEM ERROR MESSAGES (Cont)

?02 Error Messages (Cont)

?02 SYS ERROR # 018

Read Failure on the Device.

?02 SYS ERROR # 019

Write Failure on the Device.

?02 SYS ERROR # 020

Write-protection Violation.

?02 SYS ERROR # 021

Disk is Write Protected.

?02 SYS ERROR # 022

The File is Already Present.

?02 SYS ERROR # 023

Aborted by Device Driver.

?02 SYS ERROR # 024

File Flags are Locked.

?02 SYS ERROR # 025

A File is Already Open.

?02 SYS ERROR # 026

Unknown Switch Specified.

?02 SYS ERROR # 027

Unknown Unit for this Device.

?02 SYS ERROR # 028

Non-null File Name is Required.

?02 SYS ERROR # 029

Device is Incapable of Write Operations.

APPENDIX 3-A: HDOS 3.02 SYSTEM ERROR MESSAGES (Cont)

?02 Error Messages (Cont)

?02 SYS ERROR # 030

Unit not Available.

?02 SYS ERROR # 031

Illegal Value.

?02 SYS ERROR # 032

Illegal Option.

?02 SYS ERROR # 033

Volume Mounted on the Device.

?02 SYS ERROR # 034

No Volume Mounted on the Device.

?02 SYS ERROR # 035

File Open on the Device.

?02 SYS ERROR # 036

No Provisions Made for Remounting More Disks.

?02 SYS ERROR # 037

This Disk Must be Initialized Before it Can Be Mounted.

?02 SYS ERROR # 038

Unable to Read this Disk.

?02 SYS ERROR #039

Disk Structure Is Corrupt.

?02 SYS ERROR # 040

Wrong Version of HDOS.

?02 SYS ERROR # 041

No Operating System Mounted.

APPENDIX 3-A: HDOS 3.02 SYSTEM ERROR MESSAGES (Cont)

?02 Error Messages (Cont)

?02 SYS ERROR # 042

Illegal Overlay Index.

?02 SYS ERROR # 043

Overlay too Large.

?02 SYS ERROR # 044

File Is Locked Against Deletion.

?02 SYS ERROR # 045

Device Media Is Fixed.

?02 SYS ERROR # 046

Illegal Load Address.

?02 SYS ERROR # 047

Device Not Loaded.

?02 SYS ERROR # 048

Device Not Locked in Memory.

?02 SYS ERROR # 049

Device Is Fixed in Memory.

?02 SYS ERROR # 050

Illegal Date Format.

?02 SYS ERROR # 051

Illegal Time Format.

?02 SYS ERROR # 052

System Clock not Resident.

?02 SYS ERROR # 053

System Disk Is Reset.

APPENDIX 3-A: HDOS 3.02 SYSTEM ERROR MESSAGES (Cont)

?02 Error Messages (Cont)

?02 SYS ERROR # 054

Line Buffer Overflow.

?02 SYS ERROR # 055

Can't Unlink from Interrupt Vector.

?02 SYS ERROR # 056

Permission NOT given.

?02 SYS ERROR # 192

Illegal TASK Function Code.

?02 SYS ERROR # 193

TASK Is Already Active.

?02 SYS ERROR # 194

TASK Is Not Active.

?02 SYS ERROR # 195

TASK Unknown to System.

?02 SYS ERROR # 196

TASK Table Is Full.

?02 SYS ERROR # 197

TASK May Not Be Deactivated.

?02 SYS ERROR # 198

Illegal Task Sequence Number.

?02 SYS ERROR # 199

Task Notification Failed

?02 SYS ERROR # 200

Task Is Too Large. (8k Limit)

APPENDIX 3-A: HDOS 3.02 SYSTEM ERROR MESSAGES (Cont)

?02 Error Messages (Cont)

?02 SYS ERROR # 201

File Is Not Proper TASK format.

HDOS SOFTWARE REFERENCE MANUAL

HDOS DISK OPERATING SYSTEM

VERSION 3.02

CHAPTER 4

SYSCMD/PLUS

HEATH DISK OPERATING SYSTEM

SOFTWARE REFERENCE MANUAL

VERSION 3.02

HDOS was originally copyrighted in 1980 by the Heath Company. Through the years it continued to be improved by successive revisions which included 1.5, 1.6, and finally 2.0. It was entered into public domain on 19 July 1989 per letter by Jim Buszkiewicz, Managing Editor, Heath Users' Group, P.O. Box 217, Benton Harbor, MI 49022-0217 (616)982-3463. A copy of this letter is available for public inspection.

This manual is indicative of further improvements and provides for the latest revision, HDOS 3.0 and HDOS 3.02. Revision 3.0 is detailed in chapters 1, 2, and 3, while chapters 4, 5, 6, 7, 8, and 14, are the kernel of revision 3.02. Chapters 9 through 12, with minor improvements, are essentially picked up from the original HDOS 2.0 manual. Indeed, HDOS is still alive and well!

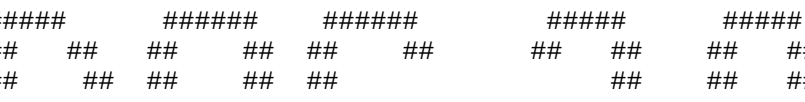
Chapter 4, SYSCMD/Plus, outlines all of the commands available under SYSCMD in HDOS 3.02 and provides examples of their use.

SPECIAL DISCLAIMER: The Heath Company cannot provide consultation on either the HDOS Operating System or user-developed or modified versions of Heath software products designed to operate under the HDOS Operating System. Do not refer to Heath for questions.

Instead, you are invited to direct any questions concerning the Heath Disk Operating System (HDOS) to Mr. Kirk L. Thompson, Editor "Staunch 89/8" Newsletter, P.O. Box 548, #6 West Branch Mobile Home Village, West Branch, IA 52358.

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont)



by W.G. (Bill) Parrott and R.L. Musgrave (a.k.a. Mighty/Soft)

Credits:

with unending gratitude to:

J.G. Letwin, for the ORIGINAL HDOS, a real operating system
G.A. Chandler, for a multitude of changes/enhancements/improvements
David Carroll, for modifications/improvements
Dean Gibson, for a vastly improved assembler, DVD support, etc
Bruce Denton, for D.G. Electronics, Super 89, DG's Utilities
Tom Jorgenson, for Software Wizardry's support of HDOS
Dale Lamm, for MicrOhio Utilities & tasks
Andy Dessler, for the Job Translator & tasks
Dale Wilson, for input to the original HDOS 3.0 team
Dave Kobets, for the best run Heathkit store in the known universe

= = = = = = = = = = = = = = = = = =

Introduction:

HDOS, the Heath Disk Operating System, is more than just an operating system, it is a philosophy. The user is the most important part of the system. The operating system MUST try to protect him at all times. Any applications written for HDOS should try to follow this philosophy.

= = = = = = = = = = = = = = = = = =

General Comments:

As always, when you are booting a freshly sysgened disk for the first time, you MUST type SPACES so that your system can determine the disk's baud rate. If there is NO write protect tab on the boot disk, this baud rate will be written onto it's boot track and you will not need to type SPACES for that disk again.

Note: Disk #1 of the HDOS 3.0 Distribution Disks, the System Disk, is a sysgened disk. If it does not have a write protect tab on it, PUT ONE THERE! In fact, always put a write protect tab on ANY distribution disk you receive. And do it before you use it. Then, when you try to boot it, you will need to type SPACES to determine the baud rate. If you have already booted it and it didn't have the write protect tab in place, DO NOT WORRY! No harm is done; see next note.

If you change your computer's baud rate via a hardware switch or reprogramming the jumper wires, you might discover that some, (or

General Comments (Cont)

all) of your bootable disks just give you garbage on the screen when you try to boot. This is usually caused by a physical change in the baud rate of the hardware. To correct this situation:

FIRST hit the space bar once and notice if any change appears on the screen.

If you do NOT see any new garbage, then hit the space bar a few more times. This should update your new hardware baud rate on the boot track and you should be booting.

If you DO see more garbage, hit the BREAK key ONLY ONCE (this is VERY important). At this point you are back at the software baud rate determination section of the boot code. Just hit the SPACE BAR a couple of times. This should update the baud rate on the boot track allowing the disk to boot.

Also, you must remove the write-protect tab while you do this, or else you will have to go through this procedure every time you boot this particular disk.

During the boot process you are asked for today's date unless a valid date is currently in memory. You no longer need to enter the entire date if certain conditions are true. If today's year is the same as the one you see in the date prompt, you need ONLY enter the day and month (ie: 17-JUN). And if today's month is the same as the one you see in the date prompt, you need ONLY enter the day (ie: 17). And, of course, if the date you see in the date prompt IS today, you need ONLY hit the RETURN key.

Also during the boot process you will see the version, revision and assembly date of the current SYSCMD/Plus followed by the version, revision, and assembly date and time of the HDOS 3.0 you are booting.

Multiple commands may be entered at the SYSCMD prompt. They MUST be separated by a back-slash '\'. For example, "M1\C1". The Command Line Editor (described below) will ONLY retain the last command of the group. If you have a string of commands and one of them is an illegal command or has illegal command syntax, SYSCMD displays that fact like usual, BUT then proceeds with the next command in the list. This means it is possible to foul things up. If the incorrectly entered command was necessary to prevent the next one from doing harmful things to your files, that next command can and will proceed like you knew what you were doing. BE CAREFUL! While multiple commands are being processed, the SYSCMD prompt is NOT visible. It returns when the commands are finished.

The command line will be parsed to see if you want SYSCMD to add device names to selected arguments. The primary device name will be used unless you preceed the command with a ';', in which case the secondary device name will be used. To invoke this feature, use ONLY the device's unit number followed immediately by a ':'. For example, "CO 1:=*.BAT" will copy all files with a .BAT extension from SY0: to SY1:

General Comments (Cont)

and give them the same names. A destination or source file specification given as "DVN:" without an explicit "name.ext" will default to "DVN:*.*".

Any command preceded with a '.' will cause the console screen to clear and enter hold screen mode. This will ONLY work with an H19 type terminal.

'.' as a command by itself will act the same as the CLS command.

HDOS now has the ability to know if the system disk you booted from is in SY0:. It needs to know this so you can continue loading and unloading device drivers as long as the original system disk is present. There is an idiosyncrasy in HDOS that will occasionally make this impossible. If you are copying files to the boot disk in SY0: and you run out of room in the middle of one of the files, HDOS still marks the memory image of the GRT table as changed. What this means is, the next time you dismount or reset SY0: and later mount the disk back onto SY0:, HDOS sees it as a different disk and will prevent loading or unloading of device drivers. The SI command will show 'System Disk NOT Mounted'. This is not a bug. The internal workings of HDOS are such that this operation has to take place. Also, if you change disks in SY0: and copy or delete files on the original boot disk while it is in another drive, the same thing, as described above, will happen.

= = = = = = = = = = = = = = = = = =

Command Line Editor:

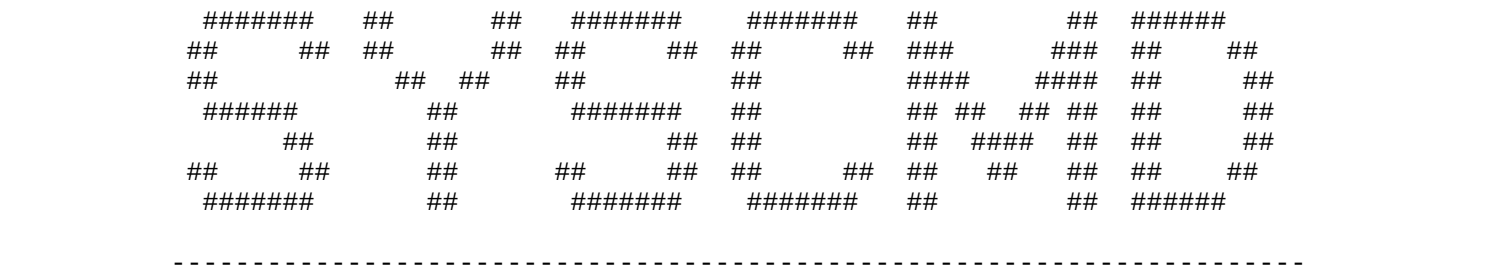
SYSCMD's Command Line Editor is invoked by CTRL-A at the SYSCMD prompt. You will then be able to EDIT the previously executed command string, except when your previous command was a multiple command. In this case, ONLY the last command in the list can be edited.

This editor acts the same as the line editor found in MicroSoft's (tm) MBASIC for HDOS except for the 'C' command. In MBASIC this command changes only one character unless you use it with a count. With a count of, say, 3, it would let you change the next 3 characters. In this implementation, the 'C' command enters overstrike mode and will continue changing characters until you exit the mode.

Command	Description
nSPACE BAR	Advance cursor right by 'n' characters (default =1). Each character is displayed as cursor moves.
nBACK SPACE	Move cursor left by 'n' characters (default =1). Each character is blanked as cursor moves but is NOT removed from buffer.

Command Line Editor (Cont)

Command	Description
nDELETE	Same as BACK SPACE, but characters are ALSO deleted from buffer, and any characters to the right are brought back to fill the space.
A	ABORT current editing, and restart the editor with original buffer contents.
C	Enter CHANGE mode (overstrike). As the user types new characters they replace the buffer contents at the cursor position. Mode is exited with ESC or CR.
nD	DELETE 'n' characters to the right of cursor (default =1). Delimit deleted text, so the user can see if he needs to abort and try again.
H	HACK off rest of buffer and enter INSERT mode.
I	Enter INSERT mode. As the user types new characters, they push the buffer contents to right, so they fit in. The bell rings if the buffer is full. Mode is exited with ESC or CR (below).
xKc	KILL (delete) characters until the 'x'th ocurrence of the character 'c'. Delimits deleted text. If character not found, then uses rest of line.
L	LIST rest of line buffer, and position cursor at start of line.
Q or CTRL-D	QUIT the editor and return to SYSCMD prompt. Original previous command still intact.
xSc	SEARCH for 'x'th occurence of the character 'c' and position the cursor there. All previous characters are displayed.
X	Add EXTRA text at end of the line. Position cursor at the end of the line, and enter INSERT mode.
ESC	ESCAPE from the INSERT or CHANGE mode. Still in editor waiting for next command.
CR	RETURN (or ENTER) KEY. Exit the INSERT or CHANGE mode, if user is in either, and return to SYSCMD with edited command ready for execution.



SYSTEM COMMAND PROCESSOR (SYSCMD)

****
** Internal Commands (75) **
****

Note: Some of the following command names have characters inside of square brackets. It is optional to type these characters.

Example: CF[LAGS] can be entered in the following ways:

CF
CFL
CFLA
CFLAG
CFLAGS

ONLY the characters left of the "[" are required to initiate the command.

= = = = = = = = = = = = = = = = = =

Filename Internal Command <<HDOS 2.0 Command>>

Entering a filename with the extension of .ABS or .BAT at the HDOS prompt will cause the file to be executed if it exists. If the file's extension is .ABS, then it is a machine language program and it will take control. If the file's extension is .BAT, then it is a BATCH file and it's contents will be treated like commands entered by the user.

Notes: BATCH files can use replacable parameters. These are designated %0 through %9. %0 is always the name of your BATCH file. %1 through %9 are the corresponding arguments entered by you on the command line. White space is used as the delimiter between arguments; therefore, each argument can ONLY be a single word. See the SHIFT command for a discussion of how to use more than 9 arguments with a command.

Other useful substitution variables are:

%n = default device name (xx)
%u = default device unit (n)
%x = default extension (ext)
%: = default device (xxn:)
%# = active USER area (0)
%p = active line printer unit # (0)
%k = ASK keystroke (Y)

Filename (Cont) Internal Command <<HDOS 2.0 Command>>

If you want to use a string such as "%1" within BATCH mode ECHO text, you will need to define it this way: "%$@1". The "$@" is the code for sending a NULL. This will separate the "%" from the "1" and prevent BATCH mode from interpreting it as a substitution request.

Synonyms = RUN, BAT

Syntax: FILENAME [optional arguments as required]

= = = = = = = = = = = = = = = = = =

??? or ?[??] Internal Command <<NEW FOR 3.02>>

Display SYSCMD's HELP file on the console.

Synonym = HELP

Syntax: ?

= = = = = = = = = = = = = = = = = =

@ Internal Command <<NEW FOR 3.02>>

Execute the TDU task (Terminal Debug Utility) if it has been started. If it has not, you will get an illegal SCALL error message. This is a very dangerous utility. With it you can poke anything anywhere in RAM. Be VERRRRRRY careful with it.

Syntax: @

= = = = = = = = = = = = = = = = = =

ALT Internal Command <<NEW FOR 3.02>>

Set or display the current system alternate device. The alternate device is used when you include a ';' at the beginning of your command.

If you set the alternate device equal to the primary device and the primary device is the default primary device, SYSCMD will force the alternate device to be the default alternate device. Conversely, if the primary device is the default alternate device, SYSCMD will force the alternate device to be the default primary device. If you have used some other device name for the primary device, then both the primary and alternate device names will remain the same. Not very useful, except for testing something.

Syntax: ALT ..... (display current alternate device)
ALT xx[:] ..... (set alternate device)
ALT : ..... (set to default alternate device)

= = = = = = = = = = = = = = = = = =

ASK or AS[K] BATCH Command <<NEW FOR 3.02>>

Display optional text on the console, wait for user to touch a key, save ASCII value of user's keystroke for later testing/use. See the PROMPT command for a list of the special characters which you can use within the optional text. Testing is done with the IF command. ASK will ONLY function within BATCH mode.

Synonym = (KEY <alpha>)

Syntax: AS ..... (just get keystroke)
AS [optional text] .... (display text & get keystroke)

= = = = = = = = = = = = = = = = = =

BATCH or BAT[CH] Internal Command <<NEW FOR 3.02>>

HDOS normally tries to execute an .ABS file when you enter just a filename and then tries to find a .BAT file if there is no .ABS

file. This command skips the .ABS portion and tries immediately for a .BAT file. This is useful if you have a long path since the search could take a while, and it would go through it twice: first looking for the .ABS file, then looking for the .BAT file.

Synonym = RUN<filename>

Syntax: BAT filename [optional arguments as required]

= = = = = = = = = = = = = = = = = =

BIT BATCH Command <<NEW FOR 3.02>>

Manipulate the 8 bits in the user control byte. Bits are numbered zero (0) through seven (7). This feature could be used to control the flow of a BATCH file. For example, you may want to run the same BATCH file more than once to achieve a different effect. Just set a bit on the first run and test for it the second time. Testing is done with the IF command.

Syntax: BIT ..... (show BIT values)
BIT S ..... (set all 8 bits)
BIT S digit ..... (set one bit, digit = 0..7)
BIT C ..... (clear all eight bits)
BIT C dight ..... (clear one bit, digit = 0..7)
BIT T ..... (toggle all eight bits)
BIT T digit ..... (toggle one bit, digit = 0..7)

= = = = = = = = = = = = = = = = = =

BYE Internal Command <<HDOS 2.0 Command>>

The old, familiar way to exit from HDOS. After using this command, you will have to do a hard reset in order to re-boot.

Synonyms = QUIT, <HALT>

Syntax: BYE

CAT or C[AT] Internal Command <<PIP>>

Display a diskette's directory on the console or send it to a file or device. This command normally uses the primary device. If you preceed it with a ';' it will use the alternate device. If no argument is given it uses the default unit of the default device.

Synonyms = Cn, DIR

Syntax: C ..... (all files)
C filename(s) ..... (selected files)
C DVn: ..... (all files)
C Dvn:filename(s) ..... (selected files)
C DVn:filename.ext= ..... (all files)
C DVn:filename.ext=filename(s) ..... (selected files)
C DVn:filename.ext=dev: ..... (all files)
C DVn:filename.ext=dev:filename(s) .... (selected files)

Cn Internal Command <<PIP>>

Display a diskette's directory on the console. This command uses the given unit of the primary device. If you preceed it with a ';' it will use the given unit of the alternate device.

Synonyms = CAT, DIR

Syntax: Cn (all files)
Cn filename(s) (selected files)

CBUF or CB[UF] BATCH Command <<NEW FOR #.02>>

Clear the console buffer. CBUF will ONLY function within BATCH mode.

Syntax: CB

CFLAGS or CF[LAGS] Internal Command <<PIP>> <<NEW FOR 3.02>>

Clear selected flags from the selected files.

Flags:
- S - System ..... (normally hidden from view)
- L - Lock ..... (can't alter flags unless SYSOP)
- W - Write protect ..... (can't write to file w/o FORCE)
- C - Contiguous ..... (CANNOT be cleared by user)
- A - Archive ..... (presently unsupported)
- B - Bad ..... (file has a bad sector in it)
- D - Delete protect ..... (file can't be deleted w/o FORCE)
- U - User ..... (any meaning the user wishes)
- * - all possible flags
- & - SLWD combination
- @ - clear access date & access count also

Syntax:
- CF DVn: ..... (all files, all flags)
- CF filename(s) ..... (selected files, all flags)
- CF DVn:=flags ..... (all files, selected flags)
- CF filename(s)=flags ..... (selected flags, selected files)

= = = = = = = = = = = = = = = = = =

CHECK or CH[ECK] Internal Command <<PIP>>

Calculate the CRC checksum of the selected files. Display these results in decimal, octal, and hex on the console or send them to a file or device.

Synonym = CRC

Syntax:
- CH DVv: ..... (all files)
- CH filename(s) ..... (selected files)
- CH DVn:filename.ext=dev: ..... (all files)
- CH DVn:filename.ext=dev:filename(s) ... (selected files)

= = = = = = = = = = = = = = = = = =

CLS Internal Command

Clear the console screen. Clear graphics and reverse video modes. Clear the 25th line and turn on the cursor. A new prompt will appear at the top of the screen if ECHO is on. If any argument is used with the CLS command, then ONLY the graphics, and reverse video modes and the 25th line will be cleared and the cursor will be turned on.

Synonym = "." NO command or arguments required.

Syntax:
- CLS ..... (clear screen & modes)
- CLS <any arg> ..... (clear modes ONLY)

= = = = = = = = = = = = = = = = = =

= = = = = = = = = = = = = = = = = =

COPY or CO[PY] Internal Command <<PIP>> <<HDOS 2.0 Command>>

Copy selected source files to selected destination files or to a selected device. If DESTINATION (i.e., DEST) device is a directory device, then *.* is assumed. If '.' is used as DEST then 'xxn:*.*' is assumed with 'xxn' being the default device and unit. If DEST is LP: and you want each file to start on new pages, then use 'LP:*.*', PRINT command does this for you. If DEST is a single file name and more than one source file is specified, then the source files will be concatenated into DESTINATION.

A new feature of HDOS 3.02 is the ability to swap the DEST disk when it is full and continue copying files. When there is not enough free space on your DEST disk to copy the next file in the list, you are given a choice. You may abort the copy operation, skip the current file and proceed, or reset the DEST drive and retry the same file. Whatever you choose, any files copied prior to this point are still intact.

Synonyms = MOVE, TYPE, LIST, PRINT

Syntax: CO destination=source
CO destination=dev: ... (source = all files)
CO .=source ..... (dest = default_dev:*.*)
CO DVn:=source ..... (dest = dev:*.*)

= = = = = = = = = = = = = = = = = =

COUNT or COU[NT] BATCH Command <<NEW FOR 3.02>>

Manipulate user counter byte. This command gives you iteration ability within BATCH files. Testing is done with the IF command.

Syntax: COU ..... (display counter value)
COU value ..... (set counter to value 0..255)
COU = value ..... (set counter to value 0..255)
COU + ... (increment counter)
COU - ..... (decrement counter)

= = = = = = = = = = = = = = = = = =

CRC Internal Command <<PIP>>

Calculate the CRC checksum of the selected files. Display these results in decimal, octal, and hex on the console or send them to a file or device. Synonym = CHECK

Syntax: CRC DVn: ..... (all files)
CRC filename(s) ..... (selected files)
CRC DVn:filename.ext=dev: ..... (all files)
CRC DVn:filename.ext=dev:filename(s) .. (selected files)

= = = = = = = = = = = = = = = = = =

DATE or DA[TE] — Internal Command — <<HDOS 2.0 Command>>

Set or display the current system date. You are no longer required to enter the complete date string. If your desired date is in the current system date's year, the year may be omitted. If your desired date is in the current system date's month and year, both the month and year may be omitted.

```
Syntax: DA              (display current system date)
        DA dd           (set system date's day ONLY)
        DA dd-mmm       (set system date's day & month ONLY)
        DA dd-mmm-yy          (set system date)
        DA no-date      (clear system date to <NO-DATE>)
```

DEFAULT or DEF[AULT] — Internal Command — <<NEW FOR 3.02>>

Set or display the current system default device. You may also set the default extension if you wish. PIP commands use this default to build source file lists unless you specify a device.

If you set the primary device equal to the alternate device and the alternate device is the default alternate device, SYSCMD will force the primary device to be the default primary device. Conversely, if the alternate device is the default primary device, SYSCMD will force the primary device to be the default alternate device. If you have used some other device name for the alternate device, then both the primary and alternate device names will remain equal. Not very useful, except for testing something.

If the alternate device is equal to the default primary device, using ";" with the appropriate commands will now affect the default primary device.

```
Syntax: DEF ..... (display current system default block)
        DEF :  ..... (set default default)
        DEF xx  ..... (set default device, same unit number)
        DEF xx:  ..... (set default device, unit = 0)
        DEF xxn[:] ..... (set default device and unit number)
        DEF xxnext ..... (set whole default block)
        DEF 0  ..... (set default extension to nulls)
        DEF ~  ..... (set whole default block to nulls)
```

DELETE or DEL[ETE] — Internal Command — <<PIP>> <<HDOS 2.0 Command>>

Delete selected files. If you select, *.* you will be asked for confirmation. If you are NOT in user area 0, you will be asked for confirmation. It is possible for you to be asked both times if you select *.* AND are outside of user area 0.

Synonym = ERASE

Syntax: DEL DVn: (all files)
DEL filename(s) (selected files)

DEVICES or DEV[ICES] — Internal Command — <<NEW FOR 3.02>>

Display current status of all known device drivers in system. Also indicate mounted units and their free space. The following symbols are used to indicate the driver's status:

= in memory, locked, fixed
** = in memory, locked
++ = in memory (temporary)
-- = in memory BUT unload is pending
= NOT in memory (blank symbol)

If a given unit is not mounted, then display its capability. The following symbols are used to do this:

D = directory device
R = capable of read
W = capable of write
U = capable of update (random)
C = capable of character mode
F = media is fixed (hard disk)
? = media is pre-3.0
N = driver requires unload notification

The last three symbols above are ONLY present with device drivers supplied with the HDOS 3.0 distribution disks. Third party drivers would have to be recompiled to comply with this standard. The trailing colon in the device name is optional.

Syntax: DEV (display all known devices)
DEV xx[:] (display specific device ONLY)

DIR Internal Command <<PIP>> <<HDOS 2.0 Command>>

Display a diskette directory on the console or send it to a file or device. This command normally uses the primary device. If you preceed it with a ';', it will use the alternate device. if no argument is given, it uses the default unit of the default device.

Synonyms = CAT, Cn

Syntax: DIR ..... (all files)
DIR filename(s) ..... (selected files)
DIR DVn: ..... (all files)
DIR DVn:filename(s) ..... (selected files)
DIR DVn:filename.ext= ..... (all files)
DIR DVn:filename.ext=filename(s) ..... (selected files)
DIR DVn:filename.ext=dev: ..... (all files)
DIR DVn:filename.ext=dev:filename(s) .. (selected files)

DISMOUNT or D[ISMOUNT] Internal Command <<HDOS 2.0 Command>>

Dismount the specified unit of the specified device. This command normally uses the primary device. If you preceed it with a ';', it will use the alternate device. If no argument is given, it will dismount the default unit of the default device.

Synonym = Dn

Syntax: D ..... (dismount default unit of default drive)
D xx: ..... (dismount unit 0 of specified drive)
D xxn: ..... (dismount specified unit of specified drive)

Dn Internal Command <<NEW FOR 3.02>>

Dismount the specified unit of the primary device. If you preceed it with a ';', it will use the alternate device.

Synonym = DISMOUNT

Syntax: Dn (n = 0..7)

= = = = = = = = = = = = = = = = = =

DMM or DM[M] — Internal Command <<NEW FOR 3.02>>

Display Main Memory: This command tabulates briefly the memory allocation currently in use. The following information is presented:

```
Total Memory:     nnnnn Bytes (nn.00 k)
HDOS Locked:      nnnnn Bytes (nn.nn k)
HDOS Reserved:    nnnnn Bytes (nn.nn k)
User Memory:      nnnnn Bytes (nn.nn k)
HDOS System:      nnnnn Bytes (nn.nn k)
```

Total Memory is just that -- the total amount of RAM that is found in your computer.

HDOS Locked is the amount of RAM used by HDOS drivers and tables and system scratch area. It occupies the top of memory.

HDOS Reserved is the amount of RAM used by HDOS drivers NOT locked in memory. It is just below HDOS Locked RAM. This will normally be 0.

User Memory is the RAM you may use for your applications. It starts at USERFWA (2280H or 042.200A) and goes up to HDOS Reserved RAM.

HDOS System is the RAM starting at the bottom of memory which contains the Base Page, HDOS 3.0 itself, H17 ROM routines, HDOS 3.0 buffers, the RAM work areas used by HDOS, and finally, the system STACK. It runs up to USERFWA.

Syntax: DM

= = = = = = = = = = = = = = = = = =

ECHO or EC[HO] — BATCH Command <<NEW FOR 3.02>>

Set or display the status of ECHO. This determines whether or not you will see commands on the console as they are entered, usually by the BATCH facility. This command also sends text to the screen even while ECHO is off. See the PROMPT command for a list of the special characters which you can use within the ECHO text. The second 'F' in OFF is optional. If you want to start the ECHO text with the word 'ON' or 'OF ,' you MUST use '$ON' or '$OF' to bypass command parsing. While ECHO is OFF, the DOS prompt is NOT shown between commands.

```
Syntax: EC ..................... (display status of ECHO state)
        EC ON  ................. (turn ECHO on)
        EC OF[F]  .............. (turn ECHO off)
        EC text to display .... (display text on console)
```

= = = = = = = = = = = = = = = = = =

END BATCH Command <<NEW FOR 3.02>>

Use this command to prematurely exit from BATCH mode. This saves the time and trouble of jumping to a label at the end of the BATCH file. Using a "C" argument will cause the CLS command to also be executed. Using any other argument will cause the optional CLS command to be executed; that is, it will clear modes. See the CLS command for an explanation of what this means.

Syntax: END ..... (exit BATCH mode)
END C ..... (exit & clear screen through CLS)
END <any arg> ..... (exit & clear modes through CLS x)

ERASE or ERA[SE] Internal Command <<PIP>>

Erase selected files. If you select *.*, you will be asked for confirmation. If you are NOT in user area 0, you will be asked for confirmation. It is possible for you to be asked both times if you select *.* AND are outside of user area 0.

Synonym = DELETE

Syntax: ERA DVn: (all files)
ERA filename(s) (selected files)

FLOAD or FLO[AD] Internal Command <<NEW FOR 3.02>>

Load a device driver into RAM, lock AND fix it there. This places the driver within HDOS Locked RAM (see DMM). Under NO circumstances can you UNLOAD this driver or any drivers or tasks above it in memory. The trailing colon is optional with this command. The system TT: and SY: drivers are fixed in memory by HDOS at boot time. This is indicated by the '##' symbol in the DEVICES command.

Synonym = LOAD

Syntax: FLO xx[:] (load, lock, and fix the xx: device driver)

GOTO or GO[TO] — BATCH Command — <<NEW FOR 3.02>>

Branch to the selected label within a BATCH file. The format of the label is ":label" (without the quotes). The first character is a colon. The search for this label starts at the first line of the BATCH file. If you know your label is after the GOTO then you should use JUMP instead, since it start it's search at the current line of the BATCH file. GOTO will ONLY function within BATCH mode.

Synonym = JUMP

Syntax: GO label

HALT or HA[LT] — Internal Command — <<NEW FOR 3.02>>

The special way to exit from HDOS. After using this command you will have to do a hard reset in order to re-boot. This exit is special in that HDOS will try to execute SHUTDOWN.ABS if it can find it on SY0:. If it can't find this file, then it tries to run SHUTDOWN.BAT from SY0:. This would be useful if you always run backups or you need to park a hard disk.

Synonyms = BYE, QUIT

Syntax: HA

HELP or H[ELP] — Internal Command — <<PIP>> <<HDOS 2.0 Command>>

Display SYSCMD's HELP file on the console.

Synonym = ???

Syntax: H

ID — Internal Command — <<NEW FOR 3.02>>

Display current version information, date compiled and assembly options. Also show FWA, LWA and buffer address.

Synonym = VERSION

Syntax: ID

IF BATCH Command <<NEW FOR 3.02>>

Conditional branching and program control within BATCH files. The command following the condition is executed if the condition yields a TRUE result. If NOT is used before the condition test, then a FALSE result will execute the command following the condition.

Available <<conditions>> are:

```
BIT digit ......... (is specified bit on, digit = 0..7)
EXI[ST] filename .. (does file exist)
COU[NT] = value ... (does counter = 8 bit unsigned value)
ERR[OR] = value ... (does error code = 8 bit unsigned value)
CRC = value ....... (does .CRCSUM = 16 bit unsigned value)
KEY = value ....... (does keystroke = 8 bit value)
string = string ... (does string 1 = string 2)
```

The value for CRC is taken from the RAM cell used by the CRC command. This value is preserved until the next COPY or MOVE command.

The value for KEY can also be an ASCII character inside of single quotes, for example, 'A'.

The (string = string) function uses white space and/or the equal sign as delimiters. Therefore, ONLY single words can be used here.

Syntax: IF <<condition>> <<command>> ..... (if TRUE)
IF NOT <<condition>> <<command>> ..... (if FALSE)

JUMP or JU[MP] BATCH Command <<NEW FOR 3.02>>

Branch to the selected label within a BATCH file. The format of the label is ":label" (without the quotes). The first character is a colon. The search for this label starts at the current line of the BATCH file. If you know your label is before the JUMP then you MUST use GOTO instead since it start it's search at the first line of the BATCH file. JUMP will ONLY function within BATCH mode.

Synonym = GOTO

Syntax: JU label

KEY BATCH command <<NEW FOR 3.02>>

Preset the ASK keystroke value. ONLY alpha characters and the special cases shown below are valid here. Some control characters will work here also, as they do with ASK.

Synonym = (ASK)

Syntax: KEY ..... (display the current value in decimal)
KEY ?<return> ..... (set it to NULL)
KEY ?<space> ..... (set it to SPACE)
KEY ?<tab> ..... (set it to TAB)
KEY ?? ..... (set it to '?')
KEY alpha ..... (set it to <<alpha>> character)

LIST or LI[ST] Internal Command <<PIP>>

Display files on the system console. If the file is NOT ASCII, you will be informed of that fact, and the command will cycle to the next filename in your list. If you use the /FORCE switch, even non-ASCII files will be displayed. Be advised that this will probably put garbage on the screen.

Synonyms = TYPE, PRINT

Syntax: LI DVn: ..... (all files)
LI filename(s) ..... (selected files)

LOAD or L[OAD] Internal Command <<HDOS 2.0 Command>>

Load a device driver into RAM and lock it there. This places the driver within HDOS Locked RAM (see DMM). Under certain circumstances, you can UNLOAD the driver later if you find you no longer need it or you need the extra space. The trailing colon is optional with this command.

Synonym = FLOAD

Syntax: L xx[:] (load & lock the xx: device driver)

= = = = = = = = = = = = = = = = = =

LOG Internal Command <<NEW FOR 3.02>>

Toggle the state of the ECHO task. This is NOT the same as the ECHO command.

Syntax: LOG ..... (turn logging on)
LOG ON ..... (turn logging on)
LOG OF[F] ..... (turn logging off)

= = = = = = = = = = = = = = = = = =

MD Internal Command <<NEW FOR 3.02>>

Multiple dismount all mounted units of specified device except SY0:. This command normally uses the primary device. If you precede the command with a ';', it will use the secondary device. If an argument is given, then dismount the device. The trailing colon in the device name is optional.

Synonym = QD

Syntax: MD ..... (Multi-dismount default device)
MD DVn: DVn: ..... (Multi-dismount specified device)

= = = = = = = = = = = = = = = = = =

MM Internal Command <<NEW FOR 3.02>>

Multiple mount all available units of specified device, if the device is ready -- that is, has a diskette in it and the door is closed. This command normally uses the primary device. If you precede the command with a ';', it will use the secondary device. If an argument is given, then mount that device. The trailing colon in the device name is optional.

Synonym = QM

Syntax: MM ..... (multi-mount default device)
MM DVn: DVn: ..... (multi-mount specified device)

= = = = = = = = = = = = = = = = = =

= = = = = = = = = = = = = = = = = =

MOUNT or M[OUNT] Internal Command <<HDOS 2.0 Command>>

Mount the specified unit of the specified device. This command normally uses the primary device. If you precede the command with a ';', it will use the alternate device. If no argument is given, it will mount the default unit of the default device.

Synonym = Mn

Syntax: M ..... (mount default unit of default drive)
M Dv: ..... (mount unit 0 of specified drive)
M DVn: ..... (mount specified unit of specified drive)

= = = = = = = = = = = = = = = = = =

Mn Internal Command <<NEW FOR 3.02>>

Mount the specified unit of the primary device. If you preceed the command with a ';' it will use the alternate device.

Synonym = MOUNT

Syntax: Mn (n = 0..7)

= = = = = = = = = = = = = = = = = =

MOVE or MOV[E] Internal Command <<PIP>> <<NEW FOR 3.02>>

Copy selected source files to selected destination files or to a selected directory device, and then delete the source file if the copy is good (tested by VERIFY). If the DESTINATION (i.e., DEST) device is NOT a directory device, then this command is same as COPY, and source files are NOT deleted. If '.' is used as DEST, then 'xxn:*.*' is assumed, with 'xxn' being the default device and unit. VERIFY is automatically enabled by the MOVE command. This command CANNOT be used to concatenate files.

Synonym = COPY/DSF (/DSF = delete source file)

Syntax: MOV destination=source
MOV .=source (dest = default_DVn:*.*)
MOV DVn:=source (dest = DVn:*.*)

= = = = = = = = = = = = = = = = = =

= = = = = = = = = = = = = = = = = =

Pn Internal Command <<NEW FOR 3.02>>

Set active list device unit number. This is used with LP: drivers like UD: which have more than one possible unit. The valid choices for the unit number are 0 through 7. A check is made to insure the unit number is known to the system. If not, the current unit number is NOT changed. This command affects PRINT and PCn. It WILL NOT make all programs default to the specified printer unit number (unless the individual programs fetch the unit number from low memory and use it themselves).

Synonym = PRN

Syntax: Pn (set unit to "n")

= = = = = = = = = = = = = = = = = =

PATH or PA[TH] Internal Command <<NEW FOR 3.02>>

Set, display or clear the system path string. No syntax checking is performed by this command. Any errors will not show up until the path is accessed by SYSCMD. It will issue the phrase 'Check Path', show you the offending characters, and give the appropriate error message.

You may define your PATH string in several ways. The delimiters are the SPACE, the TAB, the COMMA, the COLON, the SEMICOLON, or nothing. The following examples are all equivalent:

```
SY0 SY1 SY2
SY0   SY1   SY2   (tab or multiple spaces)
SY0,SY1,SY2
SY0:SY1:SY2
SY0;SY1;SY2
SY0SY1SY2
```

You may also use 'xx:' where 'xx' is a valid directory device. This will yield 'xx0'.

```
Syntax: PA ..................... (display current path string)
        PA text ............... (set new path string)
        PA ~ .................. (clear path string)
```

= = = = = = = = = = = = = = = = = =

PAUSE or PAU[SE] BATCH Command <<NEW FOR 3.02>>

Display optional text, print "Touch a key when ready," and wait for a keystroke. The keystroke is NOT saved for future testing/use. See PROMPT command for a list of special characters you can use in the optional text. PAUSE will ONLY function within BATCH mode.

Synonym = (WAIT, with NO argument)

Syntax: PAU [optional text]

PCn Internal Command <<PIP>> <<NEW FOR 3.02>>

Display a diskette's directory on the system list device, usually the line printer. This command uses the specified unit of the primary device. If you precede the command with a ';', it will use the specified unit of the alternate device. See Pn command to change printer unit number. See PRN command to change printer device name.

Syntax: PCn ..... (all files)
PCn filename(s) ..... (selected files)

PIP Internal Command <<PIP>> <<HDOS 2.0 Command>>

Execute PIP in prompted mode or execute PIP with a command. Prompted mode stays in PIP until you touch Control-D at it's prompt. If you include the command for PIP to execute, it will do the command, if possible, and then return to where you called it from. Normally this will be SYSCMD, but could also be MegaPIP (i.e., MP.ABS) ((c) Mighty/Soft) or a shell program of your own choosing. A third party shell should NOT be used unless specifically advertised for use with HDOS 3.0, or your system could hang and you could lose data.

Syntax: PIP ..... (enter PIP and prompt user)
PIP command ..... (enter PIP, do command, exit)

= = = = = = = = = = = = = = = = = =

PRINT or PR[IBNT] Internal Command <<PIP>> <<NEW FOR 3.02>>

Display files on the system list device, normally the line printer. If the file is NOT ASCII, you will get garbage on the printer so you are responsible for knowing what you are printing. The command sent to PIP will be something like 'LP0:*.*=filename(s)'. This will allow each file to start on a new page. See Pn command to change printer unit number. See PRN command to change printer device name.

Synonym = LIST, TYPE

Syntax: PR DVn: ..... (all files)
PR filename(s) ..... (selected files)

= = = = = = = = = = = = = = = = = =

PRN Internal Command <<NEW FOR 3.02>>

Set or display active list device name. Setting the list device name also resets its unit number to zero. This command lets you play around with the name of your printer. If the only argument is a colon (:), the list device name is reset to its default value, usually LP:. This command affects PRINT and PCn. It WILL NOT make all programs default to the specified printer name (unless the individual programs fetch the device name from low memory and use it themselves).

Synonym = (Pn)

Syntax: PRN ..... (display current name & unit)
PRN xx[:] ..... (set name to "xx")
PRN : ..... (set name to default)

= = = = = = = = = = = = = = = = = =

PROMPT or PRO[MPT] Internal Command <<NEW FOR 3.02>>

Set, display, or clear the system prompt string. No syntax checking is performed by this command. What you see is what you get when SYSCMD issues the new prompt.

Note: You have several special characters available to you for the PROMPT, ECHO, ASK, and PAUSE strings. They are as follows:

$d = system date (dd-mmm-yy)
$t = system time (hh:mm:ss)
$v = version number (3.02)
$n = default device name (xx)

(Continued on Next Page.)

PROMPT or PRO[MPT] Internal Command (Cont)

```
        $u = default device unit (n)
        $x = default extension (ext)
        $: = default device (xxn:)
        $# = active USER area (0)
        $p = active line printer unit # (0)
        $k = the ASK keystroke
        $h = back space + space + backspace (back up & erase)
        $> = default system prompt
        $@ = the null character
        $b = the bell character
        $< = the back space character (alone)
        $, = the tab character
        $_ = the new line character
        $^ = the form feed character
        $= = the carriage return character
        $' = the click character (^R, ONLY if you have an Ultra ROM)
        $e = the escape character
            $s = the space character
            $$ = the dollar sign character
            $~ = the tilde character (for use as first char. in text)

            You  MUST  put  $_ at the end of each physical line of your
            text  if you don't  want the cursor to remain at the end of
            the text, wherever it may be on the screen.

    Syntax: PRO ..................... (display current prompt string)
            PRO text ............... (set new prompt string)
            PRO ~  ................. (clear prompt string)
```

= = = = = = = = = = = = = = = = = = =

PUSER or PU[SER] Internal Command <<PIP>> <<NEW FOR 3.02>>

```
  Put selected files into the selected user areas.

  Users:      0 - User area 0       (CANNOT be set by you)
        1 through 7 - User area 1 through User area 7
        * - all possible user areas
            ! - put ONLY into selected areas, remove from others
                  (except 0)

    Syntax: PU dev:=users ............. (all files, selected users)
            PU filename(s)=users ...... (selected files,
                                         selected users)
```

= = = = = = = = = = = = = = = = = = =

= = = = = = = = = = = = = = = = = =

QD Internal Command <<NEW FOR 3.02>>

Quiet multiple dismount all mounted units of specified device without issuing the dismount message. This command normally uses the primary device. If you preceed it with a ';' it will use the secondary device. If an argument is given, then dismount that device. The trailing colon in the device name is optional.

Synonym = MD

Syntax: QD ..... (quiet-dismount default device)
QD xx[:] ..... (quiet-dismount specified device)

= = = = = = = = = = = = = = = = = =

QM Internal Command <<NEW FOR 3.02>>

Quiet multiple mount all available units of specified device if the device is ready, that is, has a diskette in it and the door is closed, without issuing the mount message. This command normally uses the primary device. If you preceed it with a ';' it will use the secondary device. If an argument is given, then mount that device. The trailing colon in the device name is optional.

Synonym = MM

Syntax: QM ..... (quick-mount default device)
QM xx[:] ..... (quick-mount specified device)

= = = = = = = = = = = = = = = = = =

QUIT or Internal Command
Q[UIT]

The normal way to exit from HDOS. After using this command you will have to do a hard reset in order to re-boot.

Synonym = BYE, (HALT)

Syntax: Q

= = = = = = = = = = = = = = = = = =

REM BATCH Command <<NEW FOR 3.02>>

Insert a comment into your BATCH file. You may also use a single tick mark (').

Syntax: REM [optional text]
' [optional text]

= = = = = = = = = = = = = = = = = =

= = = = = = = = = = = = = = = = = =

RENAME or REN[AME] Internal Command <<PIP>> <<HDOS 2.0 Command>>

Rename files. The device is only required on the destination name. If the source name has a lock flag, you will have to use the /FORCE switch to rename it. If you want the newly named file to retain it's original flags, you will also need to use the /KEEP switch.

Syntax: REN destination=source ..... (use default device)
REN DVn:destination=source ..... (use specified device)

= = = = = = = = = = = = = = = = = =

RESET or R[ESET] Internal Command <<HDOS 2.0 Command>>

Reset the specified unit of the specified device. This command normally uses the primary device. If you preceed it with a ';', it will use the alternate device. If no argument is given, it will reset the default unit of the default device.

Synonym = Rn

Syntax: R ..... (reset default unit of default drive)
R xx: ..... (reset unit 0 of specified drive)
R xxn: ..... (reset specified unit of specified drive)

= = = = = = = = = = = = = = = = = =

Rn Internal Command <<NEW FOR 3.02>>

Reset the specified unit of the primary device. If you precede it with a ';' it will use the alternate device.

Synonym = RESET

Syntax: Rn (n = 0..7)

= = = = = = = = = = = = = = = = = =

RUN Internal Command <<HDOS 2.0 Command>>

Execute the selected file if it exists. If the file's extension is .ABS, then it is a machine language program and it will take control. If the file's extension is .BAT, then it is a BATCH file and it's contents will be treated like commands entered by the user at the keyboard.

Synonyms = BAT, <filename>

Syntax: RUN filename [optional arguments as required]

= = = = = = = = = = = = = = = = = =

= = = = = = = = = = = = = = = = = =

RUSER or RU[SER] — Internal Command — <<PIP>> <<NEW FOR 3.02>>

Remove selected files from the selected user areas.

Users: 0 - User area 0 (CANNOT be removed by you)
1 through 7 - User area 1 through User area 7
* - all possible user areas

Syntax: RU DVn: ..... (all files, current user)
RU filename(s) ..... (selected files, current user)
RU DVn:=users ..... (all files, selected users)
RU filename(s)=users ..... (selected files, selected users)

= = = = = = = = = = = = = = = = = =

SFLAGS or SF[LAGS] — Internal Command — <<PIP>> <<NEW FOR 3.02>>

Set selected flags on the selected files.

Flags: S - System ..... (normally hidden from view)
L - Lock ..... (can't alter flags unless SYSOP)
W - Write protect ..... (can't write to file w/o FORCE)
C - Contiguous ..... (CANNOT be set by user)
A - Archive ..... (presently unsupported)
B - Bad ..... (file has a bad sector in it)
D - Delete protect ..... (file can't be deleted w/o FORCE)
U - User ..... (any meaning the user wishes)
* - all possible flags
! - set ONLY selected flags, clear all others (except C)
& - set S L W and D flags

Syntax: SF DVn:=flags (all files, selected flags)
SF filename(s)=flags (selected files, selected flags)

= = = = = = = = = = = = = = = = = =

SHIFT or SH[IFT] — BATCH Command — <<NEW FOR 3.02>>

Shift command line arguments left one position. In this way you can have more than 9 replaceable substitution parameters. The substitution variables are %0 through %9. %0 is always the name of your BATCH file. This is useful if you want your BATCH file to re-run itself. %1 through %9 are replaced by the first nine command line arguments. Each time you use the SHIFT command you discard the first argument and add the next one to the list. That is %1 now contains what used to be %2, etc. %9 becomes the next available argument if it is present on your command line.

(Continued on Next Page.)

SHIFT or SH[IFT] BATCH COMMAND (Cont)

Note: See the implicit run command for other useful substitution variables.

Syntax: SH

= = = = = = = = = = = = = = = = = =

SI Internal Command <<NEW FOR 3.02>>

Display System Information, which includes the following:

PIP Resident / PIP NOT Resident

Tells you if PIP is currently in memory or not.

System Disk Mounted / System Disk NOT Mounted

Tells you whether the disk you booted from is still in SY0: or not. If it is, then you can still load and unload devices.

System Clock Resident / System Clock NOT Resident

Tells you whether your clock task is running or not. If it is, then you can use WAIT & TIME / commands.

User Clock Vector Disabled / User Clock Vector Enabled

Tells you if the hardware control byte is allowing user clock interrupts. The H17 and H37 device drivers make use of clock interrupts so you should expect to find this feature enabled whenever either of these drivers are in memory.

Following the User Clock Vector status is the contents of the byte at ".MFLAG". The bits have the following meanings:

00000001 - Enable user clock vector
00000010 - Disable display update (H8)
00111100 - Not used at the present time
01000000 - NO refresh of front panel (H8)
10000000 - Disable HALT processing

H8 users take note: Since HDOS 3.0 occupies low memory, your front panel monitor is gone. Therefore, the two references to (H8) above are meaningless. Any TASK or DVD or other software that played around with the L.E.D.'s on your front

(Continued on Next Page.)

SI Internal Command (Cont)

panel will probably NOT function unless they provide their own refresh routines tied into the system tic counter.

System Flags = 00000000 00000000

The first 8 bits are from S.FLAG and have the following meanings:

00000001 - SYSCMD.SYS is in memory
00000010 - VERIFY mode is ON
00000100 - ECHO mode is OFF
00001000 - BATCH mode is ON
00010000 - Display exit code on re-entry
00100000 - Break off current operation
01000000 - Type-ahaead buffer stuffed by user
10000000 - SYSCMD Initialization has been done

The second 8 bits are from S.XFLAG and have the following meanings:

00000001 - HALT command in progress
00000010 - Show syntax of PIP command
00111100 - Not used at the present time
01000000 - Permission to use /DSF in PIP
10000000 - Ultra ROM is present

Syntax: SI

= = = = = = = = = = = = = = = = = =

START or ST[ART] Internal Command <<NEW FOR 3.02>>

Start a background task. Each task has it's own particular quirks and questions so just follow their own directions.

Syntax: ST taskname

= = = = = = = = = = = = = = = = = =

TIME or TI[ME] Internal Command <<NEW FOR 3.02>>

Set or display the current system time. The hour and minutes are required, but the seconds are optional. If you use '/' as the argument AND the system clock task is resident, you will get a continuous time display at the cursor. This will continue until you touch a key. At that time you are returned to SYSCMD.

Note: If you have Bill Parrott's Ultra ROM installed, you will hear the clock ticking during continuous display.

Syntax: TI[ME] ..... (display current system time)
TI[ME] hh:mm[:ss] ..... (set system time)
TI[ME] / ..... (display time continuously)

= = = = = = = = = = = = = = = = = =

TRAP or TR[AP] — BATCH Command — <<NEW FOR 3.02>>

Grab a keystroke on the fly within a BATCH file. If a keystroke is waiting, it is poked into the ASK keystroke storage location. It can be tested for with "IF NOT KEY=0" command. You should use the "KEY $" command earlier in the BATCH file to reset the KEY value if you intend to use this test. If you know what keys to expect you can use "IF KEY='X'" command without resetting the KEY value first. TRAP will ONLY function within BATCH mode.

Synonym = (ASK)

Syntax: TR

TYPE or T[YPE] — Internal Command — <<PIP>> <<HDOS 2.0 Command>>

Display files on the system console. If the file is NOT ASCII, you will be informed of that fact, and the command will cycle to the next file name in your list. If you use the /FORCE switch, even non-ASCII files will be displayed. Be advised that this will probably put garbage on the screen.

Synonyms = LIST, PRINT

Syntax: T DVn: (all files)
T filename(s) (selected files)

UNLOAD or UNL[OAD] — Internal Command — <<NEW FOR 3.02>>

Unload the selected device driver. If the driver has a mounted unit, then just flag it for unload and wait until the unit gets dismounted. At that time the unload will occur. Unload pending is indicated by the symbol '--' in the DEVICES command. The trailing colon in the device name is optional. If the argument is an asterisk '*', then all possible drivers will be unloaded or flagged for unload pending.

Syntax: UNL xx[:] (unload specified driver)
UNL * (unload all possible drivers)

= = = = = = = = = = = = = = = = = =

USER or U[SER] Internal Command <<NEW FOR 3.02>>

Set or display the current active USER area.

Synonym = Un

Syntax: U ..... (display active USER area)
U n ..... (set active USER area, n = 0..7)

= = = = = = = = = = = = = = = = = =

Un Internal Command <<NEW FOR 3.02>>

Set the current active USER area.

Synonym = USER

Syntax: Un (set active USER area, n = 0..7

= = = = = = = = = = = = = = = = = =

VERIFY or VERI[FY] Internal Command <<NEW FOR 3.02>>

Set or display the default state of the VERIFY flag for use in copying files. When VERIFY is ON, the source file is CRC'ed, and

the destination file is CRC'ed. If they match, then it is assumed that you have a good copy which is also readable by HDOS. If you are concatenating files, a running total of the source files CRC's is kept, and upon completion of the copy command the destination file is CRC'ed with the same end result. The second 'F' in 'OFF' is optional.

Syntax: VERI (display default state)
VERI ON (turn VERIFY on)
VERI OF[F] (turn VERIFY off)

= = = = = = = = = = = = = = = = = =

VERSION or VER[SION] Internal Command <<HDOS 2.0 Command>>

Display current version information, date compiled, and assembly options.

Synonym = ID

Syntax: VER

= = = = = = = = = = = = = = = = = =

= = = = = = = = = = = = = = = = = =

WAIT BATCH Command <<NEW FOR 3.02>>

Wait a predetermined number of seconds if the system clock task is resident, otherwise do nothing but return to SYSCMD. If no argument is given, WAIT will pause until you touch a key. The key value is NOT saved, so any keystroke from ASK or TRAP is NOT lost. In this way it is similar to PAUSE but with NO messages.

Synonym = (PAUSE)

Syntax: WAIT ..... (wait until keystroke)
WAIT n ..... (wait n seconds)
WAIT 0 ..... (don't wait)

= = = = = = = = = = = = = = = = = =

XYZZY or Internal Command <<NEW FOR 3.02>>

XYZ[ZY]

Toggle display of system exit code upon re-entry into SYSCMD. This command also toggles PIP's announcement of when it is loaded into memory.

Using any argument with this command will toggle display of the command syntax being sent to PIP.

Syntax: XYZ ..... (system exit code toggle)
XYZ <any arg> ..... (PIP command syntax toggle)

= = = = = = = = = = = = = = = = = =

HDOS SOFTWARE REFERENCE MANUAL

HDOS DISK OPERATING SYSTEM

VERSION 3.02

CHAPTER 5

PIP/PLUS

HEATH DISK OPERATING SYSTEM

SOFTWARE REFERENCE MANUAL

VERSION 3.02

 Through the years it continued to be improved by successive revisions which included 1.5, 1.6, and finally 2.0. It was entered into public domain on 19 July 1989 per letter by Jim Buszkiewicz, Managing Editor, Heath Users' Group, P.O. Box 217, Benton Harbor, MI 49022-0217 (616)982-3463. A copy of this letter is available for public inspection.

This manual is indicative of further improvements and provides for the latest revision, HDOS 3.0 and HDOS 3.02. Revision 3.0 is detailed in chapters 1, 2, and 3, while chapters 4, 4, 5, 6, 7, 8, and 14, are the kernel of revision 3.02. Chapters 9 through 12, with minor improvements, are essentially picked up from the original HDOS 2.0 manual. Indeed, HDOS is still alive and well!

Chapter 5, PIP/Plus, lists all of the commands available under PIP, and provides examples of their use.

SPECIAL DISCLAIMER: The Heath Company cannot provide consultation on either the HDOS Operating System or user-developed or modified versions of Heath software products designed to operate under the HDOS Operating System. Do not refer to Heath for questions.

Instead, you are invited to direct any questions concerning the Heath Disk Operating System (HDOS) to Mr. Kirk L. Thompson, Editor "Staunch 89/8" Newsletter, #6 West Branch Mobile Home Village, West Branch, IA 52358.

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont)

Modifier Switches: (Cont)

```
#########   ######  #########
##      ##    ##    ##      ##
##      ##    ##    ##      ##
#########     ##    #########
##            ##    ##
##            ##    ##
##          ######  ##
```

PERIPHERAL INTERCHANGE PROGRAM

```
*********************************
*                               *
** PIP/Plus Verb Switches (24) **
*                               *
*********************************
```

= = = = = = = = = = = = = = = = = = = =

/BRIEF or /B[RIEF] <<HDOS 2.0 Switch>>

List the contents of the directory entry of the specified files. A null name or extension in the SOURCE is taken as the '*' wildcard. Omitting the DEST will cause the destination to default to TT:. Wildcards are NOT permitted in the destination file name. This form only lists the file name and extension in column format. The default number of columns is 5.

Syntax: [DEST=]SOURCE[,...]/B

The form of the output is:

```
FILENAME.EXT  FILENAME.EXT  FILENAME.EXT  FILENAME.EXT  FILENAME.EXT
FILENAME.EXT  FILENAME.EXT  FILENAME.EXT  FILENAME.EXT  FILENAME.EXT
FILENAME.EXT  FILENAME.EXT  FILENAME.EXT  FILENAME.EXT  FILENAME.EXT

NNN Files (YYYYY Free)
```

= = = = = = = = = = = = = = = = = = = =

/CLRFLAG or /CLR[FLAG]

Clear flags on specified files. At least one source file MUST be specified. Wildcards can be used in the source file name or extension. The 'C' flag is invalid here. It can ONLY be set by HDOS directly during its .CLOSE operation.

Valid flags are:

```
        S - system ............ normally hidden from view
        L - lock .............. can't alter flags unless SYSOP
        W - write protect ..... can't write to file w/o FORCE
        C - contiguous ........ CANNOT be cleared by you
        A - archive ........... presently unsupported
        B - bad ............... file has a bad sector in it
        D - delete protect .... file can't be deleted w/o FORCE
        U - user .............. any meaning you want
```

/CLR[FLAG] or
/CLR[FLAG] (Cont)

If no flag list is given, all flags except 'C' will be cleared.
If '*' is included in flag list, all flags except 'C' will be cleared.
If '&' is included in flag list, 'SLWD' flags will be cleared.
If '@' is included in flag list, the access date & count are cleared.

Syntax: SOURCE[,...]/CLR[:flags]

= = = = = = = = = = = = = = = = = =

/CRC

Perform a CRC checksum on the specified files. Wildcards can be used in the source file name or extension. Omitting the DEST will cause the destination to default to TT:. Wildcards are NOT permitted in the destination file name.

Syntax: [DEST=]SOURCE[,...]/CRC

The form of the output is:

```
Name    .Ext    CRC Dec  CRC Oct   CRC Hex   User: u    Date: DD-MMM-YY

FILENAME.EXT    NNNNN    OOO.OOO   HHHH
FILENAME.EXT    NNNNN    OOO.OOO   HHHH
FILENAME.EXT    NNNNN    OOO.OOO   HHHH
```

= = = = = = = = = = = = = = = = = =

/DELETE or <<HDOS 2.0 Switch>>
/DEL[ETE]

Delete specified files. At least one source file MUST be specified. Wildcards can be used in the source file name or extension.

If *.* is specified, DELETE asks:

```
        !?! Delete ALL files on DEV:? Y/<N>/Q
```

NO is the default response here. A 'Q' response will set QUERY mode.

If NOT in user area 0, DELETE asks:

```
        You're in USER n, Ok? Y/<N>
```

NO is the default response here. This feature serves as ONLY a double check on your decision. It is here to remind you that you are only seeing a portion of your directory unless you use the override (/.) switch. If you answer 'Y' then the delete command will proceed.

Syntax: SOURCE[,...]/DEL

= = = = = = = = = = = = = = = = = =

= = = = = = = = = = = = = = = = = =

/DISMOUNT or <<HDOS 2.0 Switch>>
/DIS[MOUNT]

Dismount the current disk from the specified unit of the specified device. If the device name is ONLY xx:, then the unit defaults to 0.

Syntax: DEV:/DIS

= = = = = = = = = = = = = = = = = =

/FUBAR

Fouled Up Beyond All Recognition. This is a verb switch which isn't implemented. It is available for you to use to control some future patch you may dream up. It currently takes you through the PIP CRASH routine.

Syntax: [whatever]/FUBAR

= = = = = = = = = = = = = = = = = =

/FULL or
/F[ULL]

List the contents of the directory entry of the specified files. A null name or extension in the SOURCE is taken as the '*' wildcard. Omitting the DEST will cause the destination to default to TT:. Wildcards are NOT permitted in the destination file name. This is a more complete interpretation of the HDOS directory entry.

Additional information given:

Alloc ..... allocated size of the file
Users--- ..... user areas in which the file occurs
Accessed ..... the date when the file was last accessed
A/C ..... number of times the file has been accessed

Syntax: [DEST=]SOURCE[,...]/F

= = = = = = = = = = = = = = = = = =

/GROUPS or
/G[ROUPS]

List the contents of the directory entry of the specified files. A null name or extension in the SOURCE is taken as the '*' wildcard. Omitting the DEST will cause the destination to default to TT:. Wildcards are NOT permitted in the destination file name. This is a

special interpretation of the HDOS directory entry.

Additional information given:

(Continued on Next Page.)

/GROUPS or
/G[ROUPS] (Cont)

```
        Alloc ......... allocated size of the file
        FGN  .......... first group number
        LGN  .......... last group number
        LSI  .......... last sector index
```

Syntax: [DEST=]SOURCE[,...]/G

The form of the output is:

```
Name    .Ext  Alloc    Decimal       Octal     Hexidecimal  DD-MMM-YY
                     FGN LGN LSI  FGN LGN LSI  FGN LGN LSI   User: u

FILENAME.EXT  NNNNN  DDD DDD DDD  OOO OOO OOO   HH  HH  HH
FILENAME.EXT  NNNNN  DDD DDD DDD  OOO OOO OOO   HH  HH  HH
FILENAME.EXT  NNNNN  DDD DDD DDD  OOO OOO OOO   HH  HH  HH

NNN Files, Using MMMMM Sectors (YYYYY Free)
```

= = = = = = = = = = = = = = = = = =

/ID

Display version information about PIP/Plus, including version, revision, date assembled, H19 flag, SYSOP flag, Z80 flag, and whether or not user areas are supported. In addition, show FWA, LWA, buffer address, and buffer size in sectors. Source or destination file names are illegal here.

Syntax: /ID

= = = = = = = = = = = = = = = = = =

/LIST or <<HDOS 2.0 Switch>>
/L[IST]

List the contents of the directory entry of the specified files. A null name or extension in the SOURCE is taken as the '*' wildcard. Omitting the DEST will cause the destination to default to TT:. Wildcards are NOT permitted in the destination file name. This is the traditional HDOS directory format with the addition of a time stamp if the diskette was initialized with HDOS 3.0 or later.

Syntax: [DEST=]SOURCE[,...]/L

The form of the output is:

```
Name    .Ext   Size   Created    Time   Flags--  User: u Date: DD-MMM-YY
FILENAME.EXT  NNNNN  DD-MMM-YY  HH:MMz  SLWCABDU
FILENAME.EXT  NNNNN  DD-MMM-YY  HH:MMz  SLWCABDU
FILENAME.EXT  NNNNN  DD-MMM-YY  HH:MMz  SLWCABDU

NNN Files, Using MMMMM Sectors (YYYYY Free)
```

= = = = = = = = = = = = = = = = = =

/MINIMUM or
/M[INIMUM]

List the contents of the directory entry of the specified files. A null name or extension in the SOURCE is taken as the '*' wildcard. Omitting the DEST will cause the destination to default to TT:. Wildcards are NOT permitted in the destination file name. This form only lists the file name and extension in a single column. There is NO white space included to pad the name. This allows the /MINIMUM switch to create raw data for use in building BATCH files which can manipulate lists of files.

Syntax: [DEST=]SOURCE[,...]/M

The form of the output is:

```
  FILENAME.EXT
  FNAME.EXT
  FILNAM.EXT
```

/NOP

Do absolutely NOTHING. This command was added so that a SHELL program such as MegaPIP (i.e., MP.ABS) could load PIP without PIP trying to perform any operation.

Syntax: /NOP

/PUTUSER or
/PUT[USER]

Put specified files in specified user areas. At least one source file MUST be specified. Wildcards can be used in the source file name or extension. User area 0 is invalid here. All files always reside in user area 0. One or more user areas MUST be specified.

Valid user areas are 1 through 7.

If '!' is included in user list, ONLY specified user areas and 0 will be set; all others will be cleared.
If '*' is included in user list, all user areas will be set.

Syntax: SOURCE[,...]/PUT:users

= = = = = = = = = = = = = = = = = =

/REMUSER or
/REM[USER]

Remove specified files from specified user areas. At least one source file MUST be specified. Wildcards can be used in the source file name or extension. User area 0 is invalid here. All files always reside in user area 0.

Valid user areas are 1 through 7.

If no user list is given, all user areas except 0 will be cleared.
If '*' is included in user list, all user areas except 0 will be cleared.

Syntax: SOURCE[,...]/REM[:users]

= = = = = = = = = = = = = = = = = =

/RENAME or <<HDOS 2.0 Switch>>
/R[ENAME]

Rename specified files. At least one source file MUST be specified.

Wildcards can be used in the destination file name or extension as well as in the source file name or extension. The wildcard patterns MUST be compatible.

Syntax: DEST=SOURCE[,...]/R

= = = = = = = = = = = = = = = = = =

/RESET or <<HDOS 2.0 Switch>>
/RES[ET]

Reset the specified unit of the specified device. This command dismounts the unit first, if there is currently a disk mounted, tells you to replace the disk in drive DVn:, and then mounts that unit. If the device name is ONLY DV: then the unit defaults to 0.

Syntax: DVn:/RES

= = = = = = = = = = = = = = = = = =

/SETFLAG or
/SET[FLAG]

Set flags on specified files. At least one source file MUST be specified. Wildcards can be used in the source file name or extension. The 'C' flag is invalid here. It can ONLY be set by HDOS directly during it's .CLOSE operation. One or more flags MUST be specified.

(Continued on Next Page.)

/SETFLAG or (Cont)
/SET[FLAG]

Valid flags are:

```
S - system ............ normally hidden from view
L - lock .............. can't alter flags unless SYSOP
W - write protect ..... can't write to file w/o FORCE
C - contiguous ........ CANNOT be set by you
A - archive ........... presently unsupported
B - bad ............... file has a bad sector in it
D - delete protect .... file can't be deleted w/o FORCE
U - user .............. any meaning you want
```

If '!' is included in flag list, ONLY specified flags will be set,
all others will be cleared except 'C'.
If '*' is included in flag list, all flags except 'C' will be set.
If '&' is included in flag list, 'SLWD' flags will be set.

Syntax: SOURCE[,...]/SET:flags

= = = = = = = = = = = = = = = = = =

/SNAFU

Situation Normal: All Fouled Up. This is a verb switch which isn't implemented. It is available for you to use to control some future patch you may dream up. It currently takes you through the PIP CRASH routine.

Syntax: [whatever]/SNAFU

= = = = = = = = = = = = = = = = = =

/TABLE or
/TAB[LE]

Build a source list in the managed name table. At least one source file MUST be specified. Wildcards can be used in the source file name or extension. This command is mainly for use by SHELLS such as MegaPIP so they can build source file lists using the code found in PIP/Plus.

The following hooks are within PIP/Plus for your use:

```
PIP warm start +3  bytes = signature word
        first byte is version number
        second byte is revision number
PIP warm start +5  bytes = a jump to PIP sort routine
PIP warm start +8  bytes = a jump to PIP *CAD* routine
PIP warm start +11 bytes = a jump to PIP *REN* routine
PIP warm start +14 bytes = an unused jump
PIP warm start +17 bytes = a pointer to NAMTAB
PIP warm start +19 bytes = name table length (NAMTLEN)
```

(Continued on Next Page.)

/TABLE or (Cont)
/TAB[LE]

PIP warm start +21 bytes = max. size of name table (NAMTMAX)
PIP warm start +23 bytes = a pointer to PIO.DEV
PIP warm start +25 bytes = PIP's most recent ERROR code

Syntax: SOURCE[,...]/TAB

= = = = = = = = = = = = = = = = = =

/USR

Set active user area.

Valid user areas are 0 through 7. User area 0 is the default if no argument is given.

Syntax: /USR[:user]

= = = = = = = = = = = = = = = = = =

/VERSION or <<HDOS 2.0 Switch>>
/VERS[ION]

Display version information about PIP/Plus including version, revision, date assembled, H19 flag, SYSOP flag, Z80 flag, and whether or not user areas are supported. Source or destination file names are illegal here.

Syntax: /VERS

= = = = = = = = = = = = = = = = = =

/WIDE or
/W[IDE]

Synonym = /BRIEF

Syntax: [DEST=]SOURCE[,...]/W

= = = = = = = = = = = = = = = = = =

/??? or
/?[??]

Display PIP/Plus HELP file on the console.

Syntax: /?

****

PIP/Plus Modifier Switches (36)

= = = = = = = = = = = = = = = = = = =

/ACCESS or
/AC[CESS]

During date processing such as /BEFORE, /CURRENT, /TODAY, or /AFTER the file's creation date is used. By specifying the /ACCESS switch, you will be using the file's access date instead.

Syntax: [whatever]/AC

= = = = = = = = = = = = = = = = = = =

/AFTER or
/AFT[ER]

While building the source file list, include files whose date is after the specified date. A date MUST be specified.

Syntax: [whatever]/AFT:dd[-mmm[-yy]]

= = = = = = = = = = = = = = = = = = =

/AGE

While building the source file list, include files that are greater than or equal to the specified number of days old. An age MUST be specified.

Syntax: [whatever]/AGE:nnn

= = = = = = = = = = = = = = = = = = =

/ALLOCATE or <<HDOS 2.0 Switch>>
/ALL[OCATE]

During directory listing operations, use the file's allocated size

rather then its actual size. Since disk space is allocated in clusters, this will give you a better idea about how much disk space a file really takes up. See the standard disk format chart in Chapter 8, for information about cluster sizes for the various HDOS 3.0 disk drivers.

Syntax: [whatever]/ALL

= = = = = = = = = = = = = = = = = = =

/ATTRIB or
/ATT[RIB]

Set the specified flags on the destination file(s) during the COPY command. If no flags are given, then use the flags from the source file, assuming the file is from a directory device.

Valid flags are:

S - system	B - bad
L - lock	D - delete protect
W - write protect	U - user
C CANNOT be set by you	* - all but 'C'
A - archive	& - 'S','L','W' & 'D'

Syntax: [whatever]/ATT[:flags]

/BEFORE or
/BEF[ORE]

While building the source file list, include files whose date is before the specified date. If no date is given, then it defaults to the current system date.

Syntax: [whatever]/BEF[:dd[-mmm[-yy]]]

/CLS

Clear the console screen if you have an H19.

Syntax: [whatever]/CLS

/COL

During BRIEF directory listings, the files are presented in 5 columns. With this switch, you can specify a different number of columns. If you specify more columns than the target can handle, you will get wrap around if it is enabled. A number MUST be specified.

Syntax: [whatever]/COL:n

/CONTIG or
/C[ONTIG]

Make the destination file(s), during a COPY command, contiguous. That is, the file's sectors are sequential on the disk.

Syntax: [whatever]/C

/COUNT or
/COU[NT]

While building the source file list, include files which have been accessed the specified number of times or more. If no count is given then it defaults to 1. In this case you are asking for files which HAVE been accessed, even if only once.

Syntax: [whatever]/COU[:nnn]

/CURRENT or
/CUR[RENT]

While building the source file list, include files whose date is equal to the specified date. If no date is given, then it defaults to the current system date.

Syntax: [whatever]/CUR[:dd[-mmm[-yy]]]

/DATE

Use the current system date on the destination file(s) during the COPY command. The default condition is to use the source file's date, if it is a file on a directory device.

Syntax: [whatever]/D

/DSF

After successfully copying a file, delete the source file. The /VERIFY option is automatically set when you use the /DSF switch. This is how we find out if we have a successful copy.

To use this switch, the application program must have permission. The auxilliary system flag byte has a bit which shows whether or not PIP has permission to use /DSF. SYSCMD has permission; MegaPIP does NOT.

Syntax: [whatever]/DSF

/FLAG or
/FL[AG]

While building the source file list, include files with the specified flags set. One or more flags MUST be specified.

Valid flags are:

S - system	B - bad
L - lock	D - delete protect
W - write protect	U - user
C - contiguous	* - all but 'C'
A - archive	& - 'S','L','W' & 'D'

Syntax: [whatever]/FL:flags

/FORCE or
/FOR[CE]

During COPY, if the destination already exists and is flagged with the 'W' flag, copy it anyway. If you are NOT in SYSOP mode and the 'L' flag is set, the copy attempt will fail. If you ARE in SYSOP mode, copy the file.

During RENAME, if the destination is flagged with the 'W' flag, rename it anyway. If you are NOT in SYSOP mode and the 'L' flag is set, the rename attempt will fail. If you ARE in SYSOP mode, rename the file.

During DELETE, if the target file is flagged with the 'W' or 'D' flags, delete it anyway. If you are NOT in SYSOP mode and the 'L' flag is set, the delete attempt will fail. If you ARE in SYSOP mode, delete the file.

Syntax: [whatever]/FOR

/HOLD or
/H[OLD]

Set hold screen mode if you have an H89 or H19.

Syntax: [whatever]/H

/KEEP
/K[EEP]

Retain the original flags on the destination file when you need to use /FORCE to complete the command. This applies to both COPY and RENAME.

Syntax: [whatever]/FOR/K

= = = = = = = = = = = = = = = = = =

/NOCOUNT or
/NOC[OUNT]

While building the source file list, include files which have been accessed less than the specified number of times. If no count is given, then it defaults to 1. In this case you are asking for files which have NEVER been accessed.

Syntax: [whatever]/NOC[:nnn]

= = = = = = = = = = = = = = = = = =

/NOFLAG or
/NOF[LAG]

While building the source file list, include files with the specified flags cleared. If no flags are specified, then include ONLY files which have NO flags set.

Valid flags are:

S - system	B - bad
L - lock	D - delete protect
W - write protect	U - user
C - contiguous	* - all but 'C'
A - archive	& - 'S','L','W' & 'D'

Syntax: [whatever]/NOF[:flags]

= = = = = = = = = = = = = = = = = =

/NOUSER or
/NOU[SER]

While building the source file list, include files with the specified user areas cleared. If no user areas are specified, then include ONLY files which are just in user area 0.

Valid user areas are 1 through 7 and *. 0 is omitted here since all files are in user area 0.

Syntax: [whatever]/NOU[:users]

= = = = = = = = = = = = = = = = = =

/PAGE or
/P[AGE]

Paginate directory listings to the line printer. The specified number will be the number of files listed per page. If no number is given it defaults to 55.

If the directory listing is going to the console, it will be formatted into screen pages. A header line will appear on each screen page. No number should be given for this usage. This switch also affects the /CRC switch in a similar fashion.

Syntax: [whatever]/P[:nn]

= = = = = = = = = = = = = = = = = =

/QUERY or
/Q[UERY]

While building the source file list, you will be given each eligible file name one at a time. You are asked for permission to include that file in the source file list.

Valid responses to the '?':

Y - Yes, do include this file.
N - No, skip over this file.
*Y - Yes, include this file and ALL the rest. Then exit query mode.
*N - No, skip this file. Then exit query mode.

Syntax: [whatever]/Q

= = = = = = = = = = = = = = = = = =

/REVERSE or
/REV[ERSE]

Reverse the order of the source file name sort.

Syntax: [whatever]/SO[:sub-switches]/REV

= = = = = = = = = = = = = = = = = =

/SAFE or
/SA[FE]

If the destination file exists during a COPY command, you will be informed of this and asked what to do. You may either skip the file, replace it, or stop the copy operation.

Syntax: [whatever]/SA

= = = = = = = = = = = = = = = = = =

= = = = = = = = = = = = = = = = = =

/SORT or
/SO[RT]

Sort the source file name list before acting upon it. If no sub-switches are given, then the sort defaults to NAME or EXTENSION. Once you use a given sub-switch, you cannot use it again within the current sub-switch list.

Valid sub-switches are:

N - filename ascending	NR - filename descending
E - file extension ascending	ER - file extension descending
D - creation date ascending	DR - creation date descending
T - creation time ascending	TR - creation time descending
A - access date ascending	AR - access date descending
C - access count ascending	CR - access count descending

Syntax: [whatever]/SO[:sub-switches]

= = = = = = = = = = = = = = = = = =

/SUPPRESS or <<HDOS 2.0 Switch>>
/SU[PPRESS]

Supress various messages and audit trails within PIP/Plus. If no sub-switches are given, then it defaults to T & C.

Valid sub-switches are:

A - audit trails	(...Something)
H - header lines	(Name .Ext ...)
T - trailing messages	(original use of switch)
S - status line	(the 25th line)
C - files selected message	(count)
2 - unsupported	
1 - unsupported	
0 - unsupported	
* - all possible sub-switches	(AHTSC210)

Syntax: [whatever]/SU[:sub-switches]

= = = = = = = = = = = = = = = = = =

/SYSTEM or <<HDOS 2.0 Switch>>
/S[YSTEM]

While building the source file list, include files with the SYSTEM flag set.

Syntax: [whatever]/S

= = = = = = = = = = = = = = = = = =

/TODAY or
/T[ODAY]

While building the source file list, include files whose date is equal to the current system date.

Syntax: [whatever]/T

/UAREAS or
/UA[REAS]

Set the specified user areas on the destination file(s) during the COPY command. If no user areas are given, then use the user areas from the source file, assuming the file is from a directory device.

Valid user areas are 1 through 7 and *. All files reside in user area 0.

Syntax: [whatever]/UA[:users]

/USER or
/US[ER]

While building the source file list, include files within the specified user areas. One or more user areas MUST be specified.

Valid user areas are 0 through 7 and *, although 0 is always true.

Syntax: [whatever]/US:users

/VERIFY or
/V[ERIFY]

Verify the destination file during a COPY command. This is done by keeping track of the source file's CRC and then, after closing the destination file, determining the destination file's CRC. The two values are compared and, if NOT equal, it is assumed that you have a bad copy, and the copy operation aborts. If they ARE equal, the copy operation continues.

Syntax: [whatever]/V

/XXX

This is a modifier switch which isn't implemented. It is available for you to use to control some future patch you may dream up. It currently sets a flag byte at *XUSR* equal to 1 when it is invoked.

Syntax: [whatever]/XXX

= = = = = = = = = = = = = = = = = =

/YYY

This is a modifier switch which isn't implemented. It is available for you to use to control some future patch you may dream up. It currently sets a flag byte at *YUSR* equal to 1 when it is invoked.

Syntax: [whatever]/YYY

= = = = = = = = = = = = = = = = = =

/ZZZ

This is a modifier switch which isn't implemented. It is available for you to use to control some future patch you may dream up. It currently sets a flag byte at *ZUSR* equal to 1 when it is invoked.

Syntax: [whatever]/ZZZ

= = = = = = = = = = = = = = = = = =

/.

Override the automatic setting of the /USER:user switch to the active user area. This lets you perform operations in user area 0, even though you may be in another user area: user 1 through 7. User area 0, of course, means all files are included. When using this switch, the user indicator on directory and CRC lists includes an asterisk.

Syntax:[whatever]/.

= = = = = = = = = = = = = = = = = =

/-

Reverse the effect of a wildcard specification in a source file name. You MUST be careful not to specify more than one source file name (usually with a wildcard) or you will double up on the lines to be acted upon.

Syntax: SOURCE/<VERB switch>/-

= = = = = = = = = = = = = = = = = =

SOFTWARE REFERENCE
MANUAL

HDOS DISK OPERATING SYSTEM

VERSION 3.02

CHAPTER 6

HDOS 3.02 COOKBOOK

HEATH DISK OPERATING SYSTEM

SOFTWARE REFERENCE MANUAL

VERSION 3.02

 Through the years it continued to be improved by successive revisions which included 1.5, 1.6, and finally 2.0. It was entered into public domain on 19 July 1989 per letter by Jim Buszkiewicz, Managing Editor, Heath Users' Group, P.O. Box 217, Benton Harbor, MI 49022-0217 (616)982-3463. A copy of this letter is available for public inspection.

This manual is indicative of further improvements and provides for the latest revision, HDOS 3.0 and HDOS 3.02. Revision 3.0 is detailed in chapters 1, 2, and 3, while chapters 4, 5, 6, 7, 8, and 14, are the kernel of revision 3.02. Chapters 9 through 12, with minor improvements, are essentially picked up from the original HDOS 2.0 manual. Indeed, HDOS is still alive and well!

Chapter 6, HDOS 3.02 Cookbook, alphabetizes and groups first all similiar SYSCMD commands and then all of the PIP commands. This provides a panoramic view of the commands that are available, and provides you with the ability to choose the best command to accomplish your objectives.

SPECIAL DISCLAIMER: The Heath Company cannot provide consultation on either the HDOS Operating System or user-developed or modified versions of Heath software products designed to operate under the HDOS Operating System. Do not refer to Heath for questions.

Instead, you are invited to direct any questions concerning the Heath Disk Operating System (HDOS) to Mr. Kirk L. Thompson, Editor "Staunch 89/8" Newsletter, P. O. Box 548, #6 West Branch Mobile Home Village, West Branch, IA 52358.

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont)

INTRODUCTION TO THE CHAPTER

There are many new commands in HDOS 3.02 that will delight the user. This operating system is so far advanced over the original HDOS version 2.0. that once you start using it, you may never return to HDOS 2.0.

This chapter will display the commands of both Syscmd/Plus and Pip/Plus in an alphabetical/functional way, so that all of the commands that perform a certain function will be clustered together. For example, all the CAT commands. This will not only help you to decide which command is the best to use to help solve a certain problem, but also will help you to learn the HDOS 3.02 Operating System faster.

****

INTRODUCTION TO THE "COMMAND MODE" COOKBOOK

The terms "Command Mode" indicate that the commands associated are located in the file "SYSCMD.SYS." The heart of the HDOS 3.02 modification is to be found in two main disk files: (1) SYSCMD.SYS and (2) PIP.ABS. In the documentation, these correspond to "SYSCMD/Plus" and "PIP/Plus."

SYSCMD/Plus and PIP/Plus remain co-resident in memory whenever possible. This eliminates the need for slowdowns due to repeated loading and unloading of PIP/Plus, as was the case in HDOS version 2.0.

The following paragraphs comprise a list of HDOS 3.02 commands with brief explanations and examples that will introduce you to the new power and versatility of the HDOS 3.0 Operating System, Version 3.02.

The following conventions apply:

(1) To address a secondary device precede the command with a semicolon. Example: ;PC0<RTN>.

(2) Any command may be preceded by a period [.], which will clear the screen and set hold-screen mode. To advance one line in hold-screen mode, hit the SCROLL key. To advance one screen, hit SHIFT SCROLL.

(3) The command line will be parsed to see if user wants SYSCMD to add device names as arguments. The current default will be used. For example: 1:X.X becomes SY1:X.X.

(4) Multiple commands may be entered on a single command line. Separate them with a backslash. For example: M1\.C1\D1.

(5) The [^] symbol means to type a space. This is a critical matter.

(6) The commands used in SYSCMD/Plus and PIP/Plus may be abbreviated. For example: T[ype]. The characters inside the brackets [ ] do not need to be typed.

****

SYSCMD/Plus

The following is a list of SYSCMD/Plus commands for HDOS 3.02:

BYE	Two primary methods of exiting HDOS.
CAT	Provide a disk directory on the screen.
CHECK	Calculate the CRC checksum for selected files.
CLS	Clear the screen.
COPY	Copy files from one directory device to another.
CRC	Same as CHECK.
DATE	Display or set the system date.
DEFAULT	Display or set the current system default drive.
DELETE	Delete files from a disk.
DEVICES	Display current status of all known device drivers.
DIR	Same as CAT, including file sizes and other data.
DISMOUNT	Dismount a disk from a disk drive.
ERASE	Same as DELETE.
FLAGS	Set or clear flags from disk files.
FLOAD	Load a device driver into RAM, lock, and fix it in memory.
ID	Display the current version information.
LIST	Display an ASCII file on the screen.
LOAD	Load a device driver into RAM and lock it there.
LOG	A TASKing file to keep track of time expended.
MOUNT	Mount a disk on a disk drive.
MOVE	Copy selected files to a destination device, and then erase the source file.
Pn	Set the active list device driver unit number.
PATH	A route for the system to follow to find sub-directories and commands within other user areas.
PIP	Brief method of performing commands.
PRINT	Send a file to the printer.
PRN	Set or display active list device name.
PROMPT	Set, clear, or display your system prompt.
PUSER	Put a file into a user area.
QUIT	Another method for exiting HDOS.
RENAME	Rename a file.
RESET	Dismount old disk; remount new disk.
RUN	Execute a selected file.
RUSER	Remove a file from a user area.
SFLAGS	Set specific flags.
START	Start a TASK file.
TIME	Set or display the current system time.
TYPE	Display an ASCII file on the screen.
UNLOAD	Unload a selected device driver that is loaded.
USER	Set or display the current active user areas.
VERIFY	Used during the COPY process to verify files.
VERSION	Display current SYSCMD/Plus information.

Refer to disk file SYSHELP.DOC for additional commands not covered by this listing.

****

See Also:
HA[lt] or
Q[uit]

BYE
(Terminating the Session)

The command HA[lt] requires only the letters HA to run. The letters inside the brackets are optional. Do not type the brackets.

The command Q[uit] requires only the letter Q to run. The letters inside the brackets are optional. do not type the brackets.

The BYE command is similar to DISMOUNT, except that BYE automatically dismounts ALL mounted devices and prepares the system for a new Bootstrap. For example:

```
'BYE<RTN>'
"Volume 202, Dismounted from SY2:
 Label: TAX SCHEDULES 1986"
"Volume 100, Dismounted from SY1:
 Label: WORKING DISK"
"Volume 001, Dismounted from SY0:
 Label: SYSTEM VOLUME"

"System Down..."
```

You may also use the commands:

HALT<RTN> (HA[lt]<RTN>) or QUIT<RTN> (Q[uit]<RTN>)

to dismount all mounted devices and prepare the system for a new bootstrap. There is a caveat on using HALT; it first runs SHUTDOWN.ABS, or SHUTDOWN.BAT, if present. A hard reset is required to reboot.

.....

See Also:
.C
.C1
.C2
DIR

CAT or C[at]
(Cataloging Non-System Files)

This command requires only the letter C to run. The letters inside the brackets are optional. Do not type the brackets.

The C[AT] command produces a directory listing of information about a file or group of files. You may use it with or without the device specifications 1, 2, ;0, ;1, or ;2. But only one device may be specified at one time. If you do not specify a device, HDOS assumes that you mean SY0:. As before, the period [.] causes the list of files to stop after 24 lines have been printed on the screen.

If you have a disk with only BASIC.ABS on it, type C<RTN>, and the following list will be printed on the console:

CAT or C[at]
(Cataloging Non-System Files)(Cont)

```
Name    .Ext   Size   Created   Time   Flags--- User 0  Date: 17-Nov-88

BASIC   .ABS     42  17-Nov-80  12:00a    C
   1 File, Using 42 Sectors (946 Free)
```

This listing provides information about nonessential files on the disk mounted in SY0:.

To obtain a listing from a disk in another drive, type:

'C1<RTN>' or '.C1 for drive SY1:'
'C2<RTN>' or '.C2 for drive SY2:'

You can also list information about individual .DOC files on SY1: by using the C[at] command. The general format of this command is:

'C1*.DOC<RTN>' or '.C1*.DOC<RTN>'

You may use the C[at] command to print a catalog listing of files on a configured line printer (see the "Peripherals" section of this chapter to configure your line printer). The formats for this use of the C[at] command are:

'C[at] LP:=DVn:<RTN>'
'C[at] LP:=DVn:FNAME.EXT<RTN>'

Note that the command DIR is a synonym for C[at] and works in exactly the same way in all instances.

.....

See Also: C[at]/S or .C[at]/S

CAT/S or C[at]/S
(Cataloging System and Non-System Files)

The C[at]/S command produces a listing of all the files, both system and non-system, on the disk. This list will not be alphabetized. The /S modifier informs HDOS that you wish to display files, the listing of which would normally be suppressed because the "S" flag is SET on them. Type: 'C[at]/S<RTN>' and a list similar to the following list will be printed:

CAT/S or C[at]/S

(Cataloging System and Non-System Files)(Cont)

```
Name    .Ext   Size   Created   Time   Flags--- User 0  Date: 27-Jun-89

HDOS30  .SYS     40  20-May-89  12:20a   SLW C
TT      .DVD     13  20-May-89  12:20a   SLW C
SYSCMD  .SYS     40  20-May-89  12:20a   SLW C
PIP     .ABS     49  20-May-89  12:20a   SLW C
SY      .DVD     20  20-May-89  12:20a   SLW C
AUTOEXEC.BAT      1  28-Apr-89   7:44a
CLOCK   .TAS      3  28-Apr-89   7:44a
ERRORMSG.SYS      8  28-Apr-89   7:44a
SET     .ABS     12  28-Apr-89   7:44a
SYSHELP .19      43  20-May-89  12:20a
HELP    ,19      23  20-May-89  12:20a
LP      .DVD     15  17-Nov-80  12:16a
RGT     .SYS     1   20-May-89  12:16a   SLW C D
GRT     .SYS     1   20-May-89  12:16a   SLW C D
DIRECT  .SYS     24  20-May-89  12:16a   SLW C D

   15 Files, Using 421 Sectors (525 Free)
```

.....

See Also: .C[at]F or C[at]/S/F or .C[at]/S/F

CAT/F or C[at]/F

(Determining File Sector Allocation)

HDOS assigns sectors in groups, or clusters, in order to facilitate the process of extending a file. For details, see "Theory of Operation" in Chapter 6. Thus, the number of sectors HDOS assigns to a file may or may not correspond to the number of sectors that it takes to store the data in the file. The C[at] and C[at]/S commands produce listings in which the size of the file is the number of sectors that it takes to store the data in the file. When appended to the C[at] and C[at]/S commands, the/F switch will produce a listing in which the size of the file reflects the actual number of sectors that have been allocated to the file. The general format for using the /F switch is:

'C[at] DVn:/F<RTN>'

or, more commonly:

'C[at] DVn:/S/F<RTN>'

or '.C/S/F<RTN>'

for SY0:.

CAT/F or C[at]/F
(Determining File Sector Allocation)(Cont)

The following list appears on the screen:

```
Type: Boot     INIT      20-May-89 by HDOS 3.0   User 0  Date:28-Jun-89


Name    .Ext  Size Alloc  Created   Time  Flags   User 0  Accessed  A/C

HDOS30  .SYS   40   42   20-May-89 12:20a SLWC         0  28-Jun-89   2
TT      .DVD   13   18   20-May-89 12:20a SL C D       0  28-Jun-89   2
SYSCMD  .SYS   40   42   20-May-89 12:20a SLWC         0  28-Jun-89  95
PIP     .ABS   49   72   20-May-89 12:20a SL C D       0  28-Jun-89  72
SY      .DVD   20   24   20-May-89 12:20a SL C D       0  28-Jun-89  10
AUTOEXEC.BAT    1    6   28-Apr-89  7:44a C           0  28-Jun-89  20
CLOCK   .TAS    3    6   28-Apr-89  7:44a             0  28-Jun-89  95
DK      .DVD   18   20   20-May-89  7:44a             0   3-Jun-89  13
ERRORMSG.SYS    8   12   28-Apr-89  7:44a             0   3-Jun-89   1
INIT    .ABS   29   32   20-May-89  7:44a             0  28-Jun-89  32
SET     .ABS    8   12   20-Apr-89  7:44a S WC        0  28-Jun-89   4
SYSGEN  .ABS   20   24   28-Apr-89  7:44a             0  28-Jun-89   6
UD      .DVD   15   18   20-May-89 12:20a             0  28-Jun-89  84
RGT     .SYS    1    1   20-May-89 12:20a      01234567  28-Jun-89  25
GRT     .SYS    1    1   20-May-89 12:20a      01234567  28-Jun-89  25
DIRECT  .SYS   24   28   20-May-89 12:20a      01234567  28-Jun-89  25

  16 Files, Using 309 Sectors (637 Free)
```

See Also: CLS, . (Keyboard Dot)

CLS
(Clear Console Screen)

The CLS command clears the console screen. For example, if you obtain a disk directory, and want to continue with the next step in your plan, but the screen is cluttered, you can remove the clutter by simply typing CLS<RTN>. The screen clears, and the cursor moves to the "HOME" position. The alternate command is to just type a dot .<RTN>.

See Also: COPY or CO[py], VERI[fy] or MOV[e]

COPY or CO[py]
(Duplicating Files)

You may wish to have an extra copy of a file for the purposes of modification or safekeeping. Use the COPY command for such purposes. In general, many commands are a form of the COPY command. When you list the contents of a file, you are actually "copying" the file to the system console. When you run a program, you are actually "copying" the

COPY or CO[py]
(Duplicating Files)(Cont)

contents of a file into the memory of the computer and then telling the computer to "jump" to the memory location. This concept will be discussed in more detail in the "Peripheral Interchange" section.

The general format for the COPY command is:

'CO[py] DVn:DESTINATION.EXT=DVn:SOURCE.EXT<RTN>'

The destination and source may be either a filename or a device (such as LP: or TT:) or a combination of the two. You can omit the DVn: portion of both filenames if the source file is on SY0:, and you want the destination file stored there as well. If either of the files is not stored on SY0:, it is good practice to include the DVn: portion with both filenames.

HDOS 3.0 has a unique method of copying files. The general command to copy one file from one drive to another is as follows:

'CO[py] SY1:LP.DVD=SY0:LP.DVD<RTN>'

However, if you want to copy all of the files from one drive to another drive, the simplest command is:

'CO[py] SY1:=SY0:<RTN>' or 'CO[py] SY1:*.*=SY0:*.*<RTN>'

For now, you may copy one of the HDOS files such as BASIC.ABS by typing 'CO[py] TEMP.ABS=BASIC.ABS<RTN>'. The output generated will be as follows:

'CO[py] TEMP.ABS=BASIC.ABS<RTN>'

"1 FILE COPIED"

You have created an exact duplicate on the system volume of the file containing the program BASIC.ABS. The file is executable by means of any of the following commands:

'TEMP<RTN>'
'RUN TEMP<RTN>'
'TEMP.ABS<RTN>'
'RUN TEMP.ABS<RTN>'
'RUN SY0:TEMP.ABS<RTN>'

A copy of a system file, such as HDOS30.SYS, is not at all useful. System files may be copied in a useable form only by means of the program, SYSGEN, as explained in "SYSGEN" in chapters 2 and 3.

COPY or CO[py]
(Duplicating Files)(Cont)

To copy a file to a peripheral, simply specify the peripheral device name in the destination portion of the COPY command. HDOS will treat the device name as it it were a file. For example, to copy a file to the terminal, type:

```
'CO[py] TT:=SYSHELP.DOC<RTN>'
```

Or to copy this file to a printer, type:

```
'CO[py] LPn:=DVn:SYSHELP.DOC<RTN>'
```

It is also possible to copy a file from TT: to disk, as is demonstrated in the following example:

```
'CO[py] TESTFILE.DOC=TT:<RTN>'
'THIS IS A TEST.<RTN>'
'TYPE CTRL-D'
"1 FILE COPIED"
```

If you type C[at]<RTN> after performing this example, the file SY0: TESTFILE.DOC will be included in the catalog listing.

It is also possible to copy from one non-disk device to another. For example:

```
'CO[py] LP:=TT:<RTN>'
'This is another test.<RTN>'
'TYPE CTRL-L and CTRL-D'
"1 FILE COPIED"
```

When you finish entering data for the keyboard, (which was signaled by the CTRL-D command), HDOS will transmit what you typed to the system printer (LPn:).

VERIFY

Within HDOS 3.02, you can have the system verify your file(s) by using the command:

```
'CO SY1:LP.DVD=SY0:LP.DVD^/V[ERIFY]'
```

Or you can turn the VERI[fy] feature on before you copy by using the command:

```
'V[ERIfy] ON<RTN>'  or  'V[ERIfy] OFF<RTN>'
```

Finally, VERIFY may be included in the AUTOEXEC.BAT file.

COPY or CO[py]
(Duplicating Files)(Cont)

MOVE

Also, when you are planning to delete the source file after it has been copied, use the command:

'MOV[e] SY1:LP.DVD=SY0:LP.DVD<RTN>'

.....

See Also:
CH[eck]

CRC
(Provides File Checksum)

The command 'CRC^DVn:FILENAME.EXT<RTN>' provides a CRC (cyclic redundancancy check) for any file. The CRC is a multiple-digit number. On the same file, the CRC should always be the same. If it is different, something bad has happened to the copy. Usually, this indicates that somewhere along the line it has picked up a bad sector.

A similar command is 'CH[eck]^DVn:FILENAME.EXT<RTN>'. This command requires only the letters CH to run. The letters inside the brackets are optional. Do not type the brackets.

.....

DATE or DA[te]
(Manipulates System Date)

This command can do three tasks:

(1) Type 'DA[te]<RTN>', and the system shows you the current system date:

(2) Type 'DA[te]^NO-DATE<RTN>', and the system clears the system date to <NO-DATE>.

(3) Type 'DA[te]^dd-mmm-yy', and the system date is set to the new date that you have typed in.

****

See Also:
ERA[se]

DELETE or DEL[ETE]
(Deleting Files)

This command requires only the letters DEL to run. The letters inside the brackets are optional. Do not type the brackets.

From time to time you may decide that you have too many files on your disk. You can get rid of extraneous files by using the DEL[ete] command. Be forewarned, however, that there is no easy way to recover a file that has been deleted, except to copy it from a "back up" disk, such as the distribution disk. It is for this reason that the HDOS distribution disk and system files are write-protected. Write protection insures that essential system files will not be inadvertently destroyed. As a "safe" example of this command, you may delete the files that you copied in the previous sections by typing:

'DEL[ete] TEST1.DOC,TEMP.ABS<RTN>'

As a more "daring" example, you may want to delete all the files that go by the name "FARM," and reside on SY1:. A word of caution at this time. For purposes of safety, it is a good practice to always obtain a disk directory of the disk files you propose to delete in order to prevent deleting the wrong files. For example, type:

'C[at]^SY1:FARM.*<RTN>'

to see exactly what would be deleted. If all is well, then type:

'DEL[ete]SY1:FARM.*<RTN>'

.....

DEVICES or DEV[ices]
(Provides System Device Data)

This command can do two tasks:

(1) Type 'DEV[ices]', and the system shows device information for all known devices. This is a simple, one-column display.

(2) Type 'DEV[ices] DVn:' and the system shows device information for DVn:.

.....

DISMOUNT
(Dismounting Disks)

When you are finished using the disk mounted in SY1:, SY2:, DK0:, DK1:, or DK2:, etc. you must use the DISMOUNT command to instruct HDOS to restore the directory information from memory to the disk. In HDOS 3.02 there are several methods to dismount a disk:

TABLE 2: Options for Dismounting Disks in HDOS 3.02

DISMOUNTING OPTION	PRIMARY DRIVES	SECONDARY DRIVES
Old Style of Dismounting	DISMOUNT SYn:<RTN>	DISMOUNT DKn:<RTN>
HDOS 3.02 New Style:	**********	**********
Single-Dismount:	Dn<RTN>	;Dn<RTN>
Multiple-Dismount:	MD<RTN>	;MD<RTN>
Quiet-DMount Many Disks:	QD<RTN>	;QD<RTN>

Having dismounted DVn:, you can replace the dismounted disk with another. DO NOT remove the disk before it has been dismounted, or files may be lost.

The mounting of SY0: is automatically accomplished during Bootstrap. You cannot normally use the MOUNT command with SY0:, but you can use the DISMOUNT command.

.....

COMMAND LINE EDITOR

CTRL-A invokes the command-line editor at the system prompt.

The commands are as follows:

A	Abort and Restart	CTRL-D	Quit Editor
C	Change Mode	nSc	Search for Character
nD	Delete character(s)	X	Extra (Insert Mode)
H	Hack and Insert	ESC	Exit Insert or Change Mode
I	Insert Mode	nSPACE	Move Cursor Right
nKc	Kill character(s)	nBKSP	Move Cursor Left
L	List rest of line	nDEL	Move Left & Delete Chars
Q	quit editor	RET	Accept Command Line

.....

FLAGS

CFLAGS or CF[lags] to Clear: SFLAGS or SF[lags] to Set

(Write Protection)

You may decide to write-protect your files to prevent them from being inadvertently deleted or modified. You can do this by means of the FLAGS program. In HDOS 3.0, the flags available are as follows:

```
A ARCHIVE ......... Not yet supported.
B BAD ............. File has a bad sector in it.
C CONTIGUOUS ...... File recorded continuously on disk.
D DELETE PROTECT ... Cannot delete unless use PIP/FORCE.
L LOCK ............ Unalterable.
S SYSTEM .......... Identifies system files.
W WRITE PROTECT .... Prevents writing to file.
```

TO SET A FLAG:

COMMAND: 'SF^DVn:FILENAME.EXT=n' sets flag n.
COMMAND: 'SF^Filename.Ext=&' sets S, L, W, and D flags at one fell swoop.
COMMAND: 'SF^Filename.Ext=*' sets all possible flags.
COMMAND: 'SF^Filename.Ext=!' sets only selected flags: clears all others, except C.

TO DELETE A FLAG:

COMMAND: 'CF^DVn:' clears all flags on all files, except C, L, and S.
COMMAND: 'CF^DVn:FILENAME.EXT' clears all flags on one file.
COMMAND: 'CF^DVn=n' clears n flags on all files.
COMMAND: 'CF^DVn:FILENAME(S)=n' clears selected flags.

USE OF FLAGS:

Flags are used to control access to files or to identify files.

TO CONTROL ACCESS:

(1) Set the D flag to keep a file from being inadvertently deleted.
(2) Set the W flag to keep a file from being written to.
(3) Set the L flag to keep a file from being altered in any way.
NOTE: The L flag is the most powerful flag of all, since it cannot be deleted except by using the /FORCE command under PIP/Plus.

TO IDENTIFY:

(1) The B flag may be set by the user. It indicates that bad sector(s) have been found in that file.
(2) The C flag is automatically set by HDOS. It indicates that this file has been copied contiguously to the disk. The system files all carry this flag. It occurs automatically during INIT or SYSGEN. Also the C flag may be administered to selected files.
(3) The S flag is automatically set by HDOS. This indicates that the file is a system file.

FLAGS
CFLAGS or CF[lags] to SET: SFLAGS or SF[lags] to Clear
(Write Protection)(Cont)

EXAMPLES:

Setting a flag:

'SF^SY1:OPUS.DOC=D<RTN>'

Clearing a flag:

'CF^SY1:OPUS.DOC=D<RTN>'

See Also:
FL[oad] nn

LOAD or L[oad]
(Loads a Specific Device Driver)

The L[oad] command will load, and lock in memory a specific device driver. It will not fix it in memory like FLOAD will. For example: LOAD LP:<RTN>.

The FL[oad] command will load, lock, and fix in memory a specific device driver. It will not fix it in memory. Example: FL[oad] LP:<RTN>.

See Also:
LOG ON
LOG OFF

LOG
(Enables Logging Tasks)

The command 'LOG ON' enables a logging task (i.e., turns it on.)

The command 'LOG OFF' disables a logging task (i.e., turns it off.)

MOUNT
(Mounting Disks)

The disk drive units are known as directory devices. This means that HDOS maintains a directory for the disks that are mounted on the drives. The operating system also uses a table which "maps" the location of every file on the disk. For the sake of efficiency, parts of the directory and map tables (GRT.SYS) are kept in memory while HDOS is running. When a disk is removed from the system, or dismounted, these directory and table segments must be written from memory back onto the disk. If you add or delete files, you must dismount in order to reflect the most recent changes in the status of various files. But even if you change nothing on the disk, the directory and table segments must be written back to the disk (GRT.SYS) from memory.

MOUNT

(Mounting Disks)(Cont)

CAUTION

If you remove a disk from a drive without first dismounting it, and attempt to install a different disk in that drive, the second disk will NOT BE DESTROYED by HDOS 3.02, as it would be with HDOS 2.0. Instead, an error message will be issued: "? 02 A volume is presently MOUNted on the device." The operating system will refuse to mount your new disk. You may be inconvenienced, but you cannot help but respect the built-in protection provided by 3.02.

In order to repair a disk that has been damaged in this fashion, it requires not only a suitable program, but experience in the details of the HDOS disk structure. Therefore, to save a lot of grief, you MUST use the MOUNT and DISMOUNT commands when you insert and remove the disks.

Use the MOUNT command when you install a disk. Only initialized disks may be mounted. The basic requirements are that there be a disk in the drive, and that the drive door is closed. In HDOS 3.02, there are several methods to mount a disk:

TABLE 1: Options for Mounting Disks Under HDOS 3.02

MOUNTING OPTION	PRIMARY DRIVES	SECONDARY DRIVES
HDOS 2.0 Style of Mounting	MOUNT SY1:<RTN>	MOUNT DK0:<RTN>
HDOS 3.02 New Style	**********	**********
Single-Mount:	M1<RTN>	;M0<RTN>
Multiple-Mount:	MM<RTN>	;MM<RTN>
Quiet-Mount Many Disks:	QM<RTN>	;QM<RTN>

These commands inform HDOS that there is a disk installed in one or more drives. HDOS reads the table and directory segments (GRT.SYS) from the disk(s) into memory in preparation for your file manipulation commands. HDOS will not recognize any commands dealing with the drive(s) until the disk(s) are properly mounted.

See Also:
PA[th] text
PA[th]~

PATH or PA[th]

The PATH command is designed for computer systems with a hard drive. It will also work with computer systems that have high capacity disk drives such as the H37 and H47.

The PATH command is used to Set, display, or clear the system path string. No syntax checking is performed by this command. Any errors will not show up until the path is accessed by SYSCMD. It will issue the phrase, "Check Path," show you the offending characters, and give the appropriate error message.

You may define your PATH string in several ways. The delimiters are the SPACE, the TAB, the COMMA, the COLON, or the SEMICOLON. Therefore, the following examples are all equivalent:

```
SY0: SY1: SY2:   SY0, SY1, SY2   SY0SY1SY2
SY0; SY1; SY2;   SY0 SY1 SY2
```

The command 'PA[th]...TEXT<RTN>' sets the system path. NOTE: The PATH command is executed just like any other command.

To clear the system path, type: 'PA[th]~<RTN>'. (NOTE: The punctuation mark between "PA[th]" and "<RTN>" is a tilde.)

.....

See Also:
PRN LP:
PRN LPn:
PR[int]^Filename.Ext
PCn

PRN (Manipulates Printer)

The command 'PRN<RTN>' shows the current printer driver name and unit.

The command 'PRN LPn:' sets the current loaded printer driver to the default unit number. For example: LP0:.

The command 'PRN LPn:' sets the current loaded printer driver to the desired unit number.

The command 'PR[int]^filename.ext' sends the specified file to the line printer, using the current loaded printer driver.

The command 'PCn' prints the disk catalog of primary device n. If the command is preceded by a semicolon (;), it prints a disk catalog of secondary device n.

.....

See Also:
PRO[mpt] text
PRO[mpt]~

PROMPT or PRO[mpt]

The command 'PRO[mpt]<RTN>' shows the system prompt.

The command 'PRO[mpt]TEXT<RTN>' sets the system prompt. For example: when used within the AUTOEXEC.BAT file, type: PRO JOEY+>

The command 'PRO[mpt]~<RTN>' clears the system prompt.
NOTE: The punctuation mark after PRO[mpt]~ is a tilde.

RENAME or REN
(Renaming Files)

This command requires only the letters REN to run. The letters inside the brackets are optional. Do not type the brackets.

The 'REN[ame]' command is used to change the name of any file except essential system files and other protected files. System files cannot be renamed because they have the W and L flags set. For an explanation of flags, see the "FLAGS" section of this chapter. The general format for the REN[ame] command is:

'REN[ame] DVn:NEWNAME.EXT=DVn:OLDNAME.EXT<RTN>'

The DVn: portion of both filenames must be the same, as in the following example:

'REN[ame] SY1:TAX.DAT=SY1:INCOM79.DAT<RTN>'

You may omit the DVn: portion of both file names if the file you want to rename is stored on the disk in SY0:.

As an example, you can rename the file that you created in the previous section, TESTFILE.DOC, by typing:

'REN[ame] TEST1.DOC=TESTFILE.DOC<RTN>'

RESET or R^DVn:

This command requires only the letter R to run, (i.e., R^[eset]). The letters inside the brackets are optional. Do not type the brackets.

The R[eset] command (i.e., R^[eset]DVn:) first dismounts the disk that is in the drive, instructs you to remove the disk from the drive, and then instructs you to insert the new disk. There is one caution with respect to this command: before you remove the disk, insure that the

RESET or R DVn: (Cont)

RESET command has been given. It is easy if you are in a hurry to skip the command and pull the disk. HDOS 3.02 will not allow you to insert a new disk. Instead, it will give you the following error message:

?02 A volume is presently mounted on the device.

Example of using the RESET command: 'R1^<RTN>'

This command may be used to reset SY1: or SY2:

and ';R^<RTN>'

resets the secondary drives, DK0: or DK1: or DK2:.

.....

See Also: Arguments RUN FNAME[.ABS]

(Running Programs)

The format of the RUN command is:

'RUN DVn:FNAME.EXT<RTN>'

When you initialized your disks in Chapter One, "System Set-Up Procedures," you were instructed to type "INIT." Had you desired, you could have typed:

'RUN SY0:INIT.ABS<RTN>'

HDOS recognizes the contents of any file with the .ABS extension as an executable machine-code program. If you type only the FNAME portion of the filename while in the command mode, HDOS assumes that you mean "RUN SY0:FNAME.EXT." Thus, to run BASIC, simply type:

'BASIC<RTN>' for Benton Harbor BASIC
'MBASIC<RTN>' for Microsoft BASIC

If you tried either preceding example, first type BYE<RTN>. To exit B.H. BASIC, type:'Y<RTN>' when B.H. BASIC prints "Are you sure?" To exit from MBASIC, type 'SYSTEM<RTN>'. This will exit you from BASIC to the HDOS command mode.

NOTE: In calling up a program, you do not need to use the command, RUN. In HDOS the RUN is understood for any .ABS program.

Refer to the appropriate section of your software manual for more information about system resources such as BASIC.

If you wish to run a program with the .ABS extension from a disk mounted on a drive other than SY0:, you would type:

RUN (Cont)

```
'RUN DVn:FNAME<RTN>'
```

or

```
'DVn:FNAME<RTN>'
```

Thus, if you wanted to run BASIC from the disk in drive SY2:, you would enter:

```
'SY2:BASIC<RTN>'
```

All of the following formats are valid for running a program that has the .ABS extension:

```
'FNAME<RTN>'
'FNAME.ABS<RTN>'
'RUN DVn:FNAME<RTN>'
'RUN DVn:FNAME.ABS<RTN>'
```

```
'BASIC<RTN>'              \
'BASIC.ABS<RTN>'           \  All of these commands
'RUN BASIC<RTN>'            |  are equivalent.
'RUN SY0:BASIC<RTN>'       /
'RUN SY0:BASIC.ABS<RTN>' /
```

Some programs are not "interactive," (i.e., they are not designed to ask you questions and wait for your responses to act). This type of program wants all the information it needs to be supplied at the time you run it. This is done by adding "arguments" after the program name. The program "SET.ABS" is an example of such a program. If you just give the command "SET<RTN>," or "RUN SET.ABS<RTN>," the SET program will complain that you didn't provide the data it needs. The following examples are equivalent:

```
'SET^SY:^STEP 20<RTN>'
'SET.ABS^SY:^STEP 20<RTN>'
'RUN^SET^SY:^STEP 20<RTN>'
'RUN^SY0:SET^SY:^STEP 20<RTN>'
'RUN^SY0:SET.ABS^SY:^STEP 20<RTN>'
```

Refer to the SET section of chapters 2 and 3 for more information about the SET command and SET options.

See Also: TIME or TI[me]
TI[me]^/ --------------
TI[me] hh:mmm:ss (Manipulates System Time)

The command 'TI[me]' shows system time.
The command 'TI[me]^/' shows system time continuously.
The command 'TI[me]' hh:mmm:ss allows the user to set the system time.

See Also: TYPE or T[ype]
.TYPE or .T[ype] --------------
LIST or L[ist] (Listing the Contents of a File)

The command requires only the letter T to run. The letters inside the brackets are optional. Do not type the brackets.

This most basic system command allows you to examine the contents of a file on the console terminal. Some files contain text in ASCII, which is meaningful when listed. Such files usually have a .BAS or .DOC extension. Other files are written in binary code, and have no meaning when listed on the console. These files have .ABS or .DVD extensions.

Another feature is provided when using "TYPE." If you first type a . (keyboard period) followed by the T[ype] command, the screen fills 24 lines with the file and then stops to allow you to read the contents. To advance the file one line, just hit the SCROLL key. To advance the file one screen, hit SHIFT-SCROLL.

Example: '.T^SY1:QUERY.DOC<RTN>'.

In HDOS 3.02 there are two files that contain meaningful information. These files are: SYSHELP.DOC, which provides help for the SYSCMD commands, and HELP., which provides help for the PIP commands. To obtain information on any command, just TYPE either of these files, and then control them by the use of the SCROLL key.

See Also: UNLOAD or UNL[oad]
UNL[oad]^* ------------------
(Unloads a specific device driver)

The UNL[oad] command will unload a specific device driver. For example: 'UNL[oad] LP:<RTN>'.

The 'UNL[oad]^*' command will unload all device drivers on disk. Alternately, you may unload the device driver(s) of your choice. If the system has too many device drivers, this command will free up disk space to allow you to run application programs. HDOS 3.02 provides for having nearly unlimited numbers of device drivers on disk, while HDOS 2.0 is limited to only 5 at one time.

See Also:
U[ser]n
PU[ser]FN.EXT=n
RU[ser]FN.EXT
RU[ser]FN.EXT=n

USER or U[ser]
(Refers to User Areas)

(1) The command 'U[ser]' displays the active user areas.

(2) The command 'U[ser]n' sets the active user to be user area n. The term "n" can stand for any number between 0 and 7.

(3) The command 'PU[user]FN.EXT=n' puts selected files into user areas.

(4) The command 'RU[ser]Filename.Extension' removes all of the files from user areas and places them into user area zero.

(5) The command 'RU[ser]Filename.Extension=n' removes certain files from specified user areas.

.....

See Also:
ID

VERSION or VER[sion]
(Shows Version Data)

The VER[sion] command shows brief SYSCMD version data. Example: 'VER[sion]<RTN>'.

The ID command shows extended version data. Example: 'ID<RTN>'.

.....

INTRODUCTION TO THE PIP/PLUS COOKBOOK

HDOS 3.02 uses a set of system programs called PIP/Plus. PIP is an acronym for Peripheral Interchange Program. Since the file in which PIP/Plus resides has the .ABS extension, you may assume by convention that it contains an executable machine-code program. You can therefore enter PIP/Plus by simply typing 'PIP<RTN>' from the command mode. The result will be a prompt as follows:

```
'PIP<RTN>'
"P:"
```

The P: prompt will be displayed at the left margin of the system console whenever the PIP/Plus program is awaiting input.

To exit PIP/Plus, type: 'CTRL-D'.

The legal PIP/Plus commands are somewhat different from "normal" system commands. The general form is the COPY command, where a "destination" is followed by an equal sign, which is then followed by one or more "source" specifications:

```
"P:"'DVn:DESTINATION.EXT=DVn:SOURCE.EXT<RTN>'
```

As an example:

```
"P:"'SY1:TEMP2.ABS=SY2:BASIC.ABS<RTN>'
"1 FILE COPIED"
```

This example has the same effect as the COPY command. In this case, the destination is SY1:TEMP2.ABS, and the source is SY2:BASIC.ABS.

If you do not specify a destination file, PIP/Plus will assume that you refer to TT: and will copy the contents of the file onto the terminal. Each of the following commands has exactly the same result:

```
"P:"'BASIC.ABS"
"P:"'TT:=BASIC.ABS'
"P:"'TT:=SY0:BASIC.ABS'
'TYPE BASIC.ABS'
'COPY TT:=SY0:BASIC.ABS'
```

If you attempt any of these examples, the result will be a listing of binary "garbage." Hit 'CTRL-C' to cease output to the terminal. SY0:BASIC.ABS contains a machine-code program, rather than text written in ASCII.

It is possible to catalog, rename, and delete files within PIP/Plus. These functions are accomplished by means of a "switch," which is either typed after a filename or names, typed after a disk drive name, or typed by itself in response to the P: prompt.

Within the PIP/Plus section of HDOS 3.02 Cookbook, related commands are grouped together in as near alphabetical order as possible. This will help the user to select the most command.

PIP/PLUS
Peripheral Interchange Program

[1] The following PIP/Plus switches provide a variety of disk directory listings: [NOTE: letters inside the brackets may be optionally typed.]

/ALL[ocate]	Gives a catalog listing of disk allocation; not filesize.
/B[rief]	Gives a "brief" catalog listing of non-system files. This listing has multiple columns. Also see "W[ide].
/F[ull]	Gives a catalog listing of non-system files. Also provides the number of sectors allocated to files and other details.
/FL[ag]:f	Gives a catalog listing with specified flags.
/NOF[lag]:f	Gives a catalog listing without specified flags.
/G[roup]	Gives a catalog listing of specified files.
/L[ist]	Gives a catalog listing of non-system files.
/M[inimum]	Gives a minimum directory listing: one-column list.
/P:nn	Paginates directory listing (default=55 files per page).
/REV[erse]	Sorts in descending order (default=ascending order).
/S[ystem]	Gives a catalog listing including system files.
/W[ide]	Same as brief.

[2] The following PIP/Plus switches may be used for copying, deleting, renaming or to cataloging files: [Note: Letters within the brackets may be optionally typed.]

/AC[cess]	Use access date instead of creation date.
/AFT[er]	Includes files created after dd-mmm-yy.
/AGE:n	Includes files n days old or older.
/ATT:f	Set DESTINATION flags on Copy (default=source flags).
/BEF[ore]	Includes files created before dd-mm-yy (default today).
/C[ontig]	Copy files in the Contiguous mode.
/CLR	Clears flags on specified files.
/COU:n	Includes files with access count <=n (default=1).
/NOC:n	Includes files with access count <=n (default=1).
/CRC	Performs a checksum on the specified files.
/CUR[rent]	Includes files created on dd-mmm-yy (default today).
/D[ate]	Uses today's date on copy.
/DEL[ete]	Deletes files.
/DIS[mount]	Dismounts disks.
/FOR[ce]	Overrides W and D flags.
/ID	Displays data about PIP/Plus.
/K[eep]	Keeps DEST file flags on copy (use with FOR[ce]).
/PUT	Puts specified files into user areas.
/Q[uery]	Include ONLY user-selected files.
/R[ename]	Renames specified filenames.
/REM[ove]	Removes specified files from user areas.
/RES[et]	Resets a specified disk drive.
/SA[fe]	Ask before overwriting an existing file.
/SET	Sets flags on specified files.
/SO[rt]	Sort files for destination usage (default=NE).
/SU[press]	Suppresses trailing messages, headers, and status line.
/USR	Opens an active user area.
/V[erify]	Compares CRC of SOURCE FILES and DESTINATION files.
/VERS[ion]	Displays information about PIP/Plus.

[Disk Directories with Date Data]

/ACCESS ..... or ..... /AC[cess]
/AFTER dd-mmm-yy ..... or ..... /AFT[er]
/BEFORE dd-mmm-yy ..... or ..... /BEF[ore]
/CURRENT dd-mmm-yy ..... or ..... /CUR[rent]
/AGE:n
/COUNT nn: ..... or ..... /COU[nt]
/NOCOUNT:nn ..... or ..... /NOCOUNT

(1) The switch /ACCESS may be used to obtain a disk directory which includes access dates, instead of creation dates.

(2) The switch /AFTER dd-mmm-yy may be used to obtain a disk directory which includes files created after dd-mm-yy.

(3) The switch /BEFORE dd-mmm-yy includes files created before dd-mmm-yy. The default is today.

(4) The switch /AGE:n may be used to obtain a disk directory which includes files n days old or older.

(5) The switch /COUNT nn: includes files that have been accessed the number of times specified or more. In this case, you are asking for files that have been accessed at least once.

(6) The switch /NOCOUNT nn: includes files that have been accessed less than the number of times specified. In this case, you are asking for files that have never been accessed.

NOTE: These switches may also be used with COPY, DELETE, AND RENAME.

.....

[Detailed Disk Directories]

/ALLOCATE ..... or ..... /ALL[ocate]
/FULL ..... or ..... /F[ull]

(1) The /ALLOCATE switch provides a disk directory listing which displays the disk filespace allocation, not the actual file size.

(2) The /FULL switch provides disk filespace allocation as well as the last date accessed, and the number of times accessed.

.....

[Brief Disk Directories]

/BRIEF ..... or ..... /B[rief]
or ..... /B/S
/WIDE ..... or ..... /W[ide]
/COL:nn

(1) The /BRIEF and /WIDE switches provides a wide directory. These commands produce a 5-column directory of all non-system files.

(2) The /COL:nn switch is used with /B to specify how many columns are shown on a page.

.....

[Controls Directory Flags]

/FLAG:f ..... or ..... /FL[ag]:f
/NOFLAG:f ..... or ..... /NOFL[ag]:f

You can obtain a disk directory either with or without the specified flags. The command P:SY1:/FL:W<RTN> yields a directory of all of the files on disk drive SY1: which bear the W flag.

[Catalogs Non-System Files]

/L

The /L switch produces a directory listing of non-system files. The /L switch is the same as the CAT command in SYSCMD/Plus. The examples that follow are valid uses of the /L switch:

```
"P:"'SY1:/L<RTN>
"P:"'SY1:EDIT.ABS/L<RTN>'
```

NOTE: If you want a catalog of non-system files from a disk mounted in a drive other than SY0:, you must specify a device name before you type /LIST.

[Catalogs System and Non-System Files]

/L/S

The L/S switch enables you to list all the files in the system exactly like the CAT/S command in SYSCMD/Plus. The /S in both the CAT/S command and the /L/S switch is a modifier which causes files which have the 'S' flag set to be included in the listing, along with those files which you have copied or created. /S is used in PIP/Plus with /L just as it is used in the command mode with CAT.

For example:

```
"P:"'AT:=HDOS30.SYS/L/S<RTN>
```

Note that a destination, AT:, was specified, so that this listing was printed on the alternate terminal, instead of the console terminal.

[Lists Minimum Catalog]

/M

The /M switch provides a 'quick and dirty' catalog listing with all the non-system files listed in one-column. The advantage of this is quick access to file information.

[Paginates the Directory Listing]

/P:nn

The /P:nn switch paginates the directory listing. The nn stands for the number of files you want to see on one page. This is useful for those with either hard-drives or 80-track drives. The default is 55 files per page.

.....

[Controls the Files One At A Time]

/QUERY ..... or ..... /Q[uery]

The /Q[uery] switch can be used when you perform the following tasks on a number of files: CAT, COPY, DELETE, RENAME, or SYSGEN *.*/Q. It will only allow the computer to print one file, and when that file comes up on the screen, you are expected to take the necessary action. Your response to the computer is to type a Y for yes and a N for no. After you type your response, the computer goes to the next file.

.....

[Determines FGN, LGN, and LSI]

/GROUP ..... or ..... /G[roup]

This switch permits an advanced HDOS user to determine the physical location on the surface of the disk where one or more files reside. This data includes: FGN -- first group number, LGN -- last group number, and and LSI -- last sector index. This may be particularly helpful when modifying a file using a direct track and sector access method (e.g., Crash, Superzap, UDump, etc).

.....

[Using LP: with PIP/Plus]

You can print out catalog listings produced by /ALL /B, /BS, /F, /L, /L/S, and /W on a line printer by means of PIP. For example, to obtain a /L/S listing for drive SY0:, type:

"P:"'LP:=/L/S<RTN>'

Or, to obtain the same listing for a disk drive other than SY0:, type:

"P:"'LP:=DVn:/L/S<RTN>"

.....

[Copying Files Using PIP/Plus]

There is no special command for copying files in PIP/Plus; however the procedure is as follows:

"P:"DVn:Destination.Ext=DVn:Source.Ext<RTN>'

or

"P:"'1:Filter.ABS=2:Filter.ABS<RTN>'

Many people shy away from using PIP to copy files to a given device. However, as has been demonstrated, the procedure is relatively simple.

.....

[Sets the Destination Flags on a Copy]

/ATT:f

This switch sets the DESTINATION flags on a copy. The default is Set Source Flags. The lower case f indicates which flags you wish to set.

.....

[Copies Files in the Contiguous Mode]

/CONTIG ..... or ..... /C[ontig]

This switch copies files in the contiguous mode. All of the HDOS 3.02 system files are contiguous files. Contiguous means to copy files with one sector lined up one after another, instead of copying at random. You could use this flag to obtain faster disk access for your programs.

.....

[Uses Today's Date When Copying Files]

/DATE ..... or ..... /D[ate]

Uses today's date when copying files.

.....

[Manipulates Flags]

/CLR
/SET

(1) The switch /SET sets flags on specified files. At least one source file must be specified. Wildcards may be used in the source filename or extension. One or more flags must be set. The C flag is set automatically by HDOS during the SYSGEN operation. Example to set a flag:

"P:"'SY1:QUERY.DOC/SET:W<RTN>"

This sets the W flag to write-protect the file, QUERY.DOC.

[Manipulates Flags](Cont)

/CLR
/SET

(2) The switch /CLR clears flags on specified files. At least one source file must be specified. Wildcards may be used in the source filename or extension. Exception: the C flag, which is set automatically by HDOS during the SYSGEN operation. Example to clear a flag:

```
"P:"'SY1:QUERY.DOC/CLR:W<RTN>'
```

[Overrides the D, L, and W Flags]

/FORCE ..... or ..... /FOR[ce]
/KEEP ..... or ..... /K[eep]

(1) The /FORCE switch overrides the D, L, and W flags when performing COPY, DELETE, and RENAME. This is the only way to overwrite, delete, or rename the files to which these flags are set.
(2) The /KEEP switch keeps DESTINATION flags on the copy.

NOTE: Use /KEEP only with /FORCE.

[Cyclic Redundancy Check]

/CRC

The switch /CRC performs a CRC checksum on specified files. Wildcards may be used in the source filename or extension. Omitting the destination will cause the destination to default to TT:. Wildcards are not authorized in the destination filename.

[Deletes Files]

/DEL (Deleting Files)

The /DEL switch, like the DELETE command can be dangerous if you misuse it. It is difficult to recover a file that is deleted with this command, since you have to utilize a disk editor to look at the binary code of the disk itself. The format is as follows:

```
"P:"'NEWFILE.ABS/DEL<RTN>'
```

You may want to delete some files which have the S flag set, such as unnecessary device drivers. To do this, you will have to add the /S switch after the filename. This switch makes the files "visible" to PIP/Plus, which usually ignores any file that has the S flag set. For example:

```
"P:"'ATH85.DVD/S/DEL<RTN>'
```

[Displays Version Data about PIP/Plus]

/ID
/VERSION ..... or ..... /VERS[ion]

Displays version information about PIP/Plus including version, revision, date assembled, H19 flag, SYSOP flag, Z80 flag, and whether or not user areas are supported. In addition, it shows FWA, LWA, buffer address, and buffer size in sectors.
.....

[Mounting and Dismounting Disks]

/MOUNT ..... or ..... /MOU
/DISMOUNT ..... or ..... /DIS

The /MOU and /DIS switches are used in the same manner as MOUNT and DISMOUNT commands in SYSCMD. They allow you to change the disks in the drives. You MUST specify which device you want mounted or dismounted, even if you want to mount or dismount SY0:. For example:

```
"P:"'SY1:/DIS<RTN>'
```

results in:

```
"Volume 090, Dismounted from SY1:
 Label: BASIC Data Files"
```

Remounting SY1:

```
"P:"'SY1:/MOU<RTN>'
"Volume 082, Mounted on SY1:
 Label: Assembly Programs"
```

If you plan to DISMOUNT the system volume using /DIS, you will have to LOAD any devices you wish to use before executing PIP/Plus.

A sample of the LOAD command used in this context is as follows:

```
'L[oad] LP:<RTN>'

'L[oad] DK:<RTN>'

'PIP:<RTN>'

"P:"'SY1:/MOU<RTN>'
"Volume 090, Mounted on SY1:
 Label: BASIC Data Files"
```

[Mounting and Dismounting Disks](Cont)

```
"P:"'SY0:/DIS<RTN>'
"Volume 200: Dismounted from SY0:
 Label: Games Disk"

"P:"'LP:=SY1:/L/S<RTN>'

"P:"'TT:=DK1:/L/S<RTN>'
```

You do not have to LOAD devices SY: or TT:. Within HDOS 3.0, TT: they are already loaded, since they now are system files. Also, if the alternate device, DKn: has a disk mounted in any of its units, then it won't be necessary to load DK: because HDOS has already put the DK.DVD device driver into memory so that it would know how to go about mounting DK0:, DK1:, or DK2:.

.....

[Renames Files]

/RENAME ..... or ..... /REN[ame]

The /RENAME switch is used in the same manner as the RENAME command in the command mode. For example:

```
"P:"'NEWFILE.ABS=TEMP2.ABS/REN<RTN>'
```

The /R[ename] switch cannot be used to rename essential system files, or any file that has the "W" flag set, such as HDOS30.SYS. It will not work if the source file does not exist, or if the destination file is already present. However, if you do try to rename an essential system file, or try to specify a nonexistent source file or a pre-existing destination file, nothing will be damaged. HDOS will simply print an error message and await further input.

In case you want to RENAME a text or binary file that has the W flag set, you should use the switch /FOR[ce]. For details, refer to the description provided.

.....

[Switching Disks]

/RES ..... or ..... /RESET

The /RES switch will both mount and dismount a disk. For example, if You want to change the disk in drive DK1:

```
"P:"'DK1:/RES<RTN>'
"VOLUME 010, DISMOUNTED FROM DK1:
 LABEL: WORKING DISK"

"Please Replace the Diskette in Drive DK1:"
```

[Switching Disks](Cont)

When the message "Please Replace the Diskette in Drive DK1:" is displayed, remove the disk that is currently in the drive and replace it with the disk you want mounted. The /RES switch will automatically continue the mounting operation when you close the drive door.

You can also use the /RES switch to reset SY0:. This has the same effect as using /DIS to dismount SY0: and then /MOU to mount a new disk in SY0:. As with SY0:/DK0:, you are using PIP as a stand-alone program and you are therefore making HDOS inactive. Again, you must load any devices you want to use before resetting SY0:. When you exit PIP after using SY0:/RES, you will normally enter the boot routine. If you issue the /RES command for a drive that has no disk MOUNTed in it, HDOS will "know" that there is no need to "DISMOUNT" any disk first. In such a case, /RES will have exactly the same effect as /MOU, normally to mount a disk in that drive.

.....

[Queries the Operator Prior to Overwriting A File]

/SAFE ..... or ..... /SA[fe]

The /SA[fe] switch queries the computer user before overwriting an existing file.

.....

[Sorts Files in Alphabetical Order]

/SORT ..... or ...../SO[rt]

The /SO[rt] switch sorts files for DESTINATION use. The default is NAME or EXTENSION.

.....

[Supresses]

/SU

The /SU switch supresses trailing messages, headers, and status line.

.....

[Manipulates Files in User Areas]

/USR
/PUTUSER ..... or ..... /PUT[user]
/REMUSER ..... or ..... /REM[user]

HDOS 3.02 has the attribute of being able to utilize eight user areas, (i.e., 0 through 7). Each user area resembles a discrete disk to the operating system. Thus, user areas enable related files to be listed together, so as to make storing and file-handling easier. User area zero is the default if no argument is given.

[Manipulates Files in User Areas](Cont)

(1) The /USR switch sets the active user area. User area zero is the default if no argument is given. For example:

"P:"'SY1:USR:1<RTN>"

Opens user area 1 on SY1: to receive files. No files are sent.

(2) The /PUT switch enables one to send specified files into user areas from one disk to another. It is also possible to send a file into a different user area on the source disk.

At least one source file must be specified. Wildcards may be used in the source filename or extension. User area zero is invalid, as all files reside in user area zero. One or more user areas must be specified. For example:

"P:"'SY1:QUERY.DOC/PUT:1<RTN>'

Puts the file, QUERY.DOC into SY1:, user area 1.

(3) The /REM[ove] switch enables one to remove specified files from specified user areas. At least one source file must be specified. Wildcards may be used in the source filename or extension. User area zero is invalid since all files normally reside in user area zero. Example to remove a file from a user area:

"P:"'SY1:QUERY.DOC/REM:1<RTN>'

This removes the file, QUERY.DOC from SY1:, user area 1.

CAUTION: When working with user areas, be careful not to duplicate file names on the same disk, as the older version will be deleted if this situation occurs.

Queries User Before Over-writing An Existing File

****

PIP/Plus - SUMMARY

After you have become accustomed to PIP/Plus, you will probably find its "shorthand" notation more convenient than the command mode. To further expedite your operations with PIP/Plus shorthand, you can type PIP in the command mode, followed by a PIP/Plus switch or series of switches. Thus:

'PIP^/L/S<RTN>'

PIP/Plus - SUMMARY (Cont)

has the same effect as

```
'PIP<RTN>'
"P:"'/L/S'
```

When you type PIP at the command mode level followed by a command line, PIP/Plus will execute your command line and then return you to HDOS. You will therefore not be able to use the PIP/Plus command SY0:/RES except within PIP/Plus. The command:

```
'PIP^SY0:/RES<RTN>'
```

will cause PIP/Plus to reset SY0: and then immediately exit PIP/Plus.

The various file functions of copying, renaming, and so on are not actually duplicated between PIP/Plus and the system command mode, as it may seem. When you type a command, the system first decodes it, using a program which resides in the file SY0:SYSCMD.SYS. SYSCMD.SYS contains certain "built-in" commands, and if the command you type is one of these, such as STAT, VER, and DATE, SYSCMD.SYS executes it. Otherwise, SYSCMD.SYS checks to ascertain whether the command is a "transient" command - that is, a command which is a program residing in a file, such as SET.ABS and ONECOPY.ABS. All other transient commands, such as COPY, RENAME, etc, reside in PIP.ABS.

If the command you have typed is neither a built-in command nor a transient command, SYSCMD.SYS prints an error message, which it finds on SY0:ERRORMSG.SYS. If your command is a transient command, then SYSCMD.SYS passes it on to PIP for execution. PIP/Plus normally resides in a file called SY0:PIP.ABS. In order to execute any transient command, HDOS reads SY0:PIP.ABS into your system's memory. The command is then passed on to PIP/Plus, which uses other system resources, such as device drivers, to execute the command. Thus, even though you type COPY in the command mode level of HDOS, it is PIP/Plus that actually performs the copy operation.

If you have only one or two file operations to perform, you will probably find it more convenient to use the command mode forms of the commands. For more extensive file manipulation, it will be faster to run PIP.ABS and command PIP/Plus directly.

Remember that you can exit from PIP back to the command mode by typing 'CTRL-D'. You can also obtain a listing of PIP commands by typing HELP. from the command mode. This causes the file HELP. to be listed on your terminal.

****

APPENDIX 6-A: MOST USED HDOS 3.02 SYSTEM COMMANDS

CF filename(s)=flags	Clear selected flags from listed files
CO DVn:=source	Copy source DVn: to destination DVn:. (SY0: is destination in this example)
CO SY2:READ.DOC=SY1:READ.DOC .	Copy to SY2: from SY1: a file, READ.DOC
DEL DVn: or ..... DEL SY1:*.*	Delete all files on DVn:. Gives a verification query first
DEL SY1:JILL.001	Deletes file JILL.001 on drive SY1:
Dn	Dismount unit n of primary drive chain
D1	Dismount unit SY1:
HA (Halt)	Runs "Shutdown.BAT" first. Then shuts down system same as BYE.
MD	Multi-Dismount primary drives except for SY0:
Mn	Mount primary device DVn:
M1	Mount drive SY1:
MM	Multi-Mount all primary drives except for SY0:
MOV DVn:=Source	Copy source filename(s) to destination; then delete source filenames if CRC okay
REN	Rename destination=source
REN	SY1:JILL.DOC=JANE.DOC
RDVn:	Reset drive DVn:
R1	Reset drive SY1:
SF filename(s)=flags	Set selected flags of listed filenames
SF JANE.DOC=C	Sets contiguous flag on file JANE.DOC
ST taskname	Start a task
T filename	Type a filename to the screen

APPENDIX 6-A: MOST USED HDOS 3.02 SYSTEM COMMANDS (Cont)

SY1:.T KAREN.LTR..... Type a file by the name of KAREN.LTR to the screen. Then place the contents in a "hold screen" configuration. This file resides on SY1:.

Un ..... Set active user (0 thru 7)

U2 ..... Set user 2

PU filename(s)=user ..... Put selected files into the selected user area you designated previously with the command: Un<RTN>

RU filename(s)=user ..... Remove selected files from the designated user area when you typed Un<RTN>

NOTES:

1. The term n, such as Un stands for a number. In this case, the number may be from 1 thru 7. Another example is: DVn:, where the expression stands for a disk drive from SY0: thru SY3:.

2. DVn: stands for a disk drive. In this case, the number may be from 1 thru 3.

3. A User is a discrete disk area where certain files may be made to reside. Each of these discrete disk areas is "stand-alone." With the the use of "user" you are permitted to copy the same filename on the disk several times, provided the files are sent to different user areas.

HDOS SOFTWARE REFERENCE MANUAL

HDOS DISK OPERATING SYSTEM

VERSION 3.02

CHAPTER 7

ADVANCED TECHNIQUES

HEATH DISK OPERATING SYSTEM

SOFTWARE REFERENCE MANUAL

VERSION 3.02

HDOS was originally copyrighted in 1980 by the Heath Company. Through the years it continued to be improved by successive revisions which included 1.5, 1.6, and finally 2.0. It was entered into public domain on 19 July 1989 per letter by Jim Buszkiewicz, Managing Editor, Heath Users' Group, P.O. Box 217, Benton Harbor, MI 49022-0217 (616)982-3463. A copy of this letter is available for public inspection.

This manual is indicative of further improvements and provides for the latest revision, HDOS 3.0 and HDOS 3.02. Revision 3.0 is detailed in chapters 1, 2, and 3, while chapters 4 ,5, 6, 7, 8, 13 and 14, are related to revision 3.02. Chapters 9 through 12, with minor improvements, are essentially picked up from the original HDOS 2.0 manual. Indeed, HDOS is still alive and well!

Chapter 7, Advanced Techniques, lists and describes the native utilities that are a part of HDOS 3.02. Further, it lists the associated utilities that make using this version of HDOS so pleasant. It provides new techniques etc., which have been ported from MS-DOS. This chapter makes available many new facilities that are not only challenging but quite practical to daily computer use.

SPECIAL DISCLAIMER: The Heath Company cannot provide consultation on either the HDOS Operating System or user-developed or modified versions of Heath software products designed to operate under the HDOS Operating System. Do not refer to Heath for questions.

Instead, you are invited to direct any questions concerning the Heath Disk Operating System (HDOS) to Mr. Kirk L. Thompson, Editor "Staunch 89/8" Newsletter, P.O. Box 548, #6 West Branch Mobile Home Village, West Branch, IA 52358.

TABLE OF CONTENTS

INTRODUCTION

HDOS 3.02 has several new utilities, advanced techniques, and test programs. Some of the utilities are available with the distribution disks, and some of them are associated utilities that are written by D-G Electronics, and which work well with HDOS 3.02. Each of these new features will provide the user with new tools and facilities to make his computer activities easier and more fun.

SECTION 1: HDOS 3.02 Utilities

New utilities are available as a companion set for use with HDOS 3.02. These utilities include the following:

- Archive
- Bad
- DS - Directory Sort
- Dskcpy
- MegaPip
- Prodump
- Verify

These utilities are described within this chapter. Any or all of them would be a fine addition to your computer program collection. Refer to the Table of Contents to determine their location.

Fortunately, HDOS 3.02 associated utilities may be purchased for a mere pittance from Kirk L. Thompson editor of the Staunch'8/89 Newsletter, Lot #6 - Route 1, West Branch Mobile Home Village, West Branch, IA 52358.

SECTION 2: Advanced Techniques

The Advanced Techniques were imported from the MS-DOS Operating System. Once you learn them, you will find that using them is almost like entering a new dimension. Chances are, you will elect to add them to your personal bag of computer tricks.

SECTION 3: System Analysis

The test programs provide a comprehensive means for testing certain aspects of your HDOS 3.02 system. Selected programs are displayed for close inspection. You will find that they are more detailed and more useful than the programs heretofore available within HDOS version 2.0, with the possible exception of TEST17/37/47, which is only available with HDOS 2.0.

****

SECTION 1: HDOS 3.02 UTILITIES

= = = = = = = = = = = = = = = = = =

ARCHIVE -- Disk Archive Utility (From D.G. Electronics)

ARCHIVE (ARC.ABS) is a disk back-up program for use with more than two drives. With this utility, you can store, retrieve, and directory the contents of a complete diskette into a single save file. Utilizing the fact that files are set up by groups, this utility allows the user to waste at most one group on the destination device instead of one group per source file. For example, the H47, utilizing double-sided, double-density disks, has a group size of 16 sectors. If the user backed up a 400-sector H17 disk, he may be using more than 500 sectors on the H47 due to the group allocation. By packing an entire disk into a single file, the user also avoids file-naming conflicts, since HDOS never knows about the directory structure of the archived disk. This utility may be used with HDOS V 3.0 and up.

[A] SAVING A DISK TO AN ARCHIVE FILE :

ARC savefile=inputdevice: [/switches]

/Q - Prompt the user whether to include the named file in the archive file. If the user doesn't want to save the file, simply respond with 'N' to the prompt.

/S - Don't include 'S' flagged files in the archive file.

Note: If both the '/S' and '/Q' switches are used, ARCHIVE will only prompt for those files without the 'S' flag.

[B] RESTORING A DISK FROM AN ARCHIVE FILE :

ARC ouputdevice:=archivefile [/switches]

/Q - This switch controls the deleting of files on the destination device. Each file will be prompted for, and if the user wishes not to delete the file, he should respond with an 'N' to the prompt.

/S - This switch keeps all files with the 'S' flag on the destination device. This can be helpful when converting from one version of an operating system to another.

/K - This switch allows the user simply to add the files of the savefile to the destination disk without changing volume control information or deleting any files on the destination.

SECTION 1: HDOS 3.02 UTILITIES (Cont)

ARCHIVE - Disk Archive Utility (Cont)

[C] TO DIRECTORY AN ARCHIVE FILE:

```
ARC [listfile=]archivefile /LI
```

[D] TO RESTORE A SINGLE FILE FROM AN ARCHIVE FILE:

```
ARC dev:filename=archivefile
```

= = = = = = = = = = = = = = = = = =

BAD -- Bad Block Correction Utility (From D.G. Electronics)

BAD (BAD.ABS) is a utility to find and remove bad blocks from an HDOS recognized directory device. In this way the disk may be patched to remove bad groups from use without reinitialization.

These bad sectors are normally detected through the "TEST" routine that destroys all data contained on the disk. Through the use of BAD, the user may remove bad sectors without reinitialization. By telling HDOS that the group is bad, the user will save the frustration of having HDOS detect a bad sector and return a "Write Failure" error message while writing out a data file that just took two hours to edit. These bad blocks may be due to wear, excessive heat, poor quality recording surfaces, misaligned heads, etc.

Through the use of BAD, the user may keep any disk "up-to-par." By reading each group on the disk, the user may remove sectors from a file which has bad sectors, then save most of the data contained in the file without getting a "Read Failure" error message while copying the files. The user will also avoid getting write errors on the disk, since all bad sectors are accounted for and blocked from being used. Also, by using BAD to examine a disk after initialization, the user can tell INIT the bad sectors on the disk without running TEST. It should be noted that any program file, i.e. those files with an extension of .ABS or .SYS, may NOT be used after a group has been removed from the file. In these cases, the file should be deleted, and the file restored from a distribution diskette.

The basic format for the bad command is:

```
BAD [listfile]=input device: [/switches]
```

SECTION 1: HDOS 3.02 UTILITIES (Cont)

BAD - Bad Block Correction Utility (Cont)

The "listfile" is optional, but if specified, the default device will be "SY0:," and the default extension will be ".BAD", "Input device" is the name of the device to be checked, and switches are one of the following;

/LI -- To only show the bad blocks currently on the device.

/MAN -- To allow the user to check individual groups on the specified device.

= = = = = = = = = = = = = = = = = = =

DS -- A Directory Sort Utility

Usage: DS [DVn:] [/sort keys] [/X]

	ascending	descending
Sort Keys:	N = name	NR = name
	E = ext	ER = ext
	D = date	DR = date
	T = time	TR = time
	A = acc. date	AR = acc. date
	C = count	CR = count

DVn: is any valid HDOS device (SY1:, DK0:, etc.)

The /X switch will produce an extended audit of the sort operation.

The directory on the specified disk will be sorted.

= = = = = = = = = = = = = = = = = = =

DSKCPY -- Universal Disk Copy (From D.G. Electronics)

DSKCPY (CPY.ABS) is a disk duplication utility for use with HDOS version 3.0 or higher. Its main purpose is to eliminate the need for device-dependent copy programs, such as "FTCOPY" or "COPYER". By using the operating system device values, DSKCPY can determine all needed device parameters for complete device initialization and copying. Since this is a 'universal' disk copy utility, optimum speed cannot be obtained by using particular device-dependent copy algorithms. However, as the disk is copied one full track at a time, the speed of this program will be comparable to device-dependent programs.

SECTION 1: HDOS 3.02 UTILITIES (Cont)

DSKCPY -- Universal Disk Copy (Cont)

For example, benchmark tests on the H17 comparing DSKCPY with "COPYER" show that DSKCPY is 70% as fast as "COPYER". Since no other device-dependent copy routines exist for other devices, comparisons cannot be made.

The general form of the command is as follows:

```
    CPY [destination device:=source device:]

        *** O P T I O N S ***

  V - Copy a disk with verification
  C - Copy a disk without verification
  I - Initialize and copy a non-HDOS volume
  Q - Quit, returning to HDOS
```

V: Verify a Disk

The "V" command is used to copy and verify a disk. UNDER NO CONDITIONS SHOULD A USER INITIALIZE THE DESTINATION VOLUME! This option will copy all tracks from the source to destination. This is the most extensive command allowed under DSKCPY. During this command, the user may change both the volume ID and volume label. The user should note that the volume ID may ONLY be changed if the destination device is initialized.

C: Copy a Disk without Verification

The "C" command is used to copy a disk WITHOUT verification. Verification requires that the source track be read, written to the destination, re-read from the destination and compared to the source track that was read. By eliminating this extra read, the speed at which the disk is copied is increased by 50%. However, this should be used only if the user is sure of the hardware and diskettes. All options described in the "V" command will apply: i.e. volume ID's and labels may be altered during the copy.

I: Copy a Non-HDOS Disk

The "I" command is used to copy non-HDOS volumes from source to destination. This may be used when copying CP/M disks or other non-HDOS volumes where the disk must not be altered. This copy was set up with CP/M disks in mind, i.e. the volume number on the destination will be set to zero. Naturally, the user will not be prompted to change either the volume label or ID. The destination device MUST be initialized to determine the device size to be copied. Therefore, the user must be sure that the source device is completely compatible with the destination. That is, don't try to copy a

SECTION 1: HDOS 3.02 UTILITIES (Cont)

DSKCOPY -- Universal Disk Copy (Cont)

I: Copy a Non-HDOS Disk (Cont)

single-sided 8" disk on the H47 to a double-sided disk. It is the responsibility of the user to insure that the devices are compatible.

Q: Quit and Return to HDOS

The "Q" command will return the user to HDOS. If the device copied was SY0:, or the destination device was a Tandon driver, the user must re-boot. The Tandon driver's INIT code was not designed to return to HDOS. Optionally the user may type <CTRL-D> to return to HDOS.

= = = = = = = = = = = = = = = = = =

MEGAPIP -- File Handling Utility

MegaPip is an excellent file-handling utility for HDOS 3.02. This is the first "shell" program developed for the HDOS Operating system.

When MegaPip is called up (i.e., MP<RTN>), a graphic rectangle will be "painted" on the screen and it will take up the entire screen. The question mark key [?] will call up the "Help Files" without requiring the pressing of the <RTN>. The "Help Files" provide the following displays:

HELP SCREEN 1

```
     -- Tagging Functions --            -- File Functions --

     T = Tag file                       C = Copy files
     U = Untag file                     R = Rename files
     W = Wild tag/untag                 D = Delete files
                                        I = File info
                                        N = File CRC
     -- Misc. Functions --              F = Alter file(s) Flags
                                        A = File(s) user Areas
     L   = new Login
     S   = free Space
     E   = Edit file                    -- Viewing Functions --
     X   = eXecute file
     +/- = next/previous screen         V = View file(s)
     ?   = Help                         P = Print file(s)
     Q   = Quit MegaPip                 H = Hex Dump
```

SECTION 1: HDOS 3.02 UTILITIES (Cont)

MEGAPIP - File-Handling Utility (Cont)

```
HELP Screen 2:
                          f1 = Run PIP
                     f2 = Sort file table
                     f4 = active user area
                   f5 = mount/dismount/reset

                      arrows move cursor
                      home -> first file
                      white -> last file

                      esc = abort at key
                  ^D (CTRL-D) = abort at text

                         blue = refresh
                          red = quit
```

How to Use MegaPip:

BACKGROUND:
When MegaPip is called, it will bring up the list of files from SY0:. If you want to bring up the file list of SY1: or SY2:, just type 'L' at the cursor. You will be asked for an argument. In this case, just type 'SY1:', or the desired drive name, and press '<RTN>'. MegaPip will then log onto SY1: and bring up the files found there.

You can move the shadow bar with the use of the arrow keys. To move from screen to screen, place the shadow bar on the last column of files to the right and then place it on the last file in the column. Then type '<RTN>'.

HOW TO USE THE UTILITY:
Probably one of the most often-used tasks is copying files from one drive to another. To copy files, first you must tag them. Place the shadow bar on the first file that you want to copy. Type 'T'. (Ignore the apostrophes.) Continue to move the shadow bar to the file(s) that you want to copy and type 'T' for each one. When you are done with tagging the files to be copied, type 'C' at the cursor.

The program will ask: "COPY - <T>agged, <U>ntagged, <F>ile." Type 'T.' The program will ask: "Copy TAGGED to?" Type 'SY1:'. The program goes to a plain screen and prints the following:

"PIP SY1:=SY0:Filename.Ext/S/SU:CST"

"SY0:Filename.Ext --> SY1:Filename.Ext ... Copied"

SECTION 1: HDOS 3.02 UTILITIES (Cont)

MEGAPIP - File-Handling Utility (Cont)

This expression is made for each file that you want copied. When it is done, the program instructs: "Touch Any Key .... " When you touch any key, the screen shifts, and you find yourself back into MegaPip's main screen.

When you want to exit the program, type "Q" at the cursor. The program will ask: "Are You Sure ? Y/N." Just type 'Y,' and you will be back at the HDOS system prompt (for example, DAN+>).

This is but one example of the versatility of this fine program. For example, when you are in MegaPip you can perform any of the day-to-day tasks that you would normally do, only with more convenience. The tasks that you can select are limited by the options provided.

CAUTION

MegaPip is a fine program, but it is recommended that you approach it carefully, since it can cause you to lose files! The first time one uses it, you can inadvertently delete all of your files on the disk being displayed. To prevent deleting all of your files on disk when using the Delete command, FIRST TAG the specific files that you want deleted. THEN tell the program to delete only the tagged ones.

= = = = = = = = = = = = = = = = = =

PRODUMP -- Disk Dump Utility (From D.G. Electronics)

PRODUMP (DMP.ABS) is an intelligent disk dump for HDOS. By using device drivers and small library routines, PRODUMP will be able to dump from any device created for the H8/H89/Z89 running under version 3.0 or higher.

At this time, two devices are fully supported. These are the H17 (5.25") and the H47 (8"). There are also two other devices receiving limited support through library routines. These are the TANDON 80 track drives, and the ST506 five-megabyte hard disk. Through the use of the configure command, new devices may be added if the device driver exists. Otherwise, there is no method to access the device.

MAKING A PARAMETER FILE TO DEFINE NEW DEVICES:

This section will describe how to build and edit a parameter file for use with new devices. During start-up, the file "SY0:PARAM.DMP" will be accessed. If found, the needed information will be copied and placed in the device tables. This file will contain the same basic formats as used in the configure command. Note that once these defaults are read in, the user may still alter the device type by use of the configure command. The format for this file will be:

SECTION 1: HDOS 3.02 UTILITIES (Cont)

PRODUMP -- Disk Dump Utility (Cont)

DVn:,sides,tracks,sectors/track

where "sides" is the number of sides/unit, "tracks" is the number of tracks per side, and "sectors/track" is the number of 256-byte sectors per track. An example input for the H17 and H47 devices is shown below:

H17 as device SY:

```
SY:,1,40,10     As a single-sided, 48 TPI drive
```

H47 as device DK:

```
DK:,1,76,13     As a single-sided, single-density drive

DK:,2,76,26     As a double-sided, double-density drive
```

These entries are allowed, one per line for as many devices as needed. The user should note that the H17 and H47 examples are for understanding the format. A user using only these two device types does NOT need to create a parameter file for PRODUMP. Only those users using a new device driver need the ability to create a permanent table for their new devices.

.....

COMMAND SUMMARY REFERENCE GUIDE

Command	Description	Required	Valid Mode	Arguments
Address	Find specified address in .ABS or PIC file		File	Split-octal
Configure	Configure device or terminal characteristics		Any	Terminal/Device
Device	Select a specified device for track/sector reading		Any	Device name
Edit	Alter bytes as read from file or device		File/ Device	None
File	Select a specified file to be dumped		Any	File Name

SECTION 1: HDOS 3.02 UTILITIES (Cont)

PRODUMP -- Disk Dump Utility (Cont)

Command	Description	Required Valid Mode	Arguments
Group	Calculate and read device physical group number	Device	Decimal Group
Hex	Change output mode to HEX and print the buffer again	Any	None
List	Print the current sector again	Any	None
Mode	Set the output mode to either Octal, Hex, or Decimal	Any	O,D,H
Octal	Change output mode to OCTAL and print the buffer again	Any	None
Print	Show the sector as valid ASCII or EBCDIC without a header	Any	None
Quit	Exit PRODUMP and return to HDOS	Any	None
Rewind	Rewind to beginning of file and read/show the first sector	File	None
Sector	Position to virtual/physical sector number and read it	File/ Device	Decimal Sector #
Track	Get device track and sector numbers for reading	Device	TT/SS number
Write	Rewrite the current sector to disk	File/ Device	<CR> or "X"
X	Show current PRODUMP status	Any	None
<CR>	Read the next virtual/physical sector and show it	File/ Device	None
<ESC>	Continue previous search function, for the same string	File/ Device	None
<CTRL-A>	Start hardcopy listing file Toggle listing on/off	Any Any	Listing File None

SECTION 1: HDOS 3.02 UTILITIES (Cont)

PRODUMP -- Disk Dump Utility (Cont)

Command	Description	Required Valid Mode	Arguments
<CTRL-B>	Set-up/continue search function on file/device	File/Device	None
<CTRL-C>	Reset PRODUMP, and return to "<DMP>" prompt	Any	None
<CTRL-D>	Exit PRODUMP and return to HDOS	Any	None

= = = = = = = = = = = = = = = = = = =

VERIFY -- Disk Verify Utility (From D.G. Electronics)

VERIFY (VFY.ABS) is a utility for use with HDOS format disks with a format compatable with HDOS version 2.0. This includes all disks in the range from version 1.0, through version 2.0. Later format types may be different, and as such are unsupported. The verification process involves a multi-stage evaluation of the disk structure, followed by a report for the user. The evaluation involves both format and integrity checks, including bad group detection, analysis of directory entries, analysis of the reserved group table (RGT), the group reservation table (GRT), and volume control block (VCB).

THE BASIC COMMAND FORMAT:

```
vfy [listfile]=input device:[/switches]
```

file, which if not specified will default to "TT0:". The "input device:" specification must be a valid and known HDOS directory device name, and "/switches" are one of the switches specified in the section below:

/B[AD]

The "/BAD" switch is used to request verify to check the device for bad groups detected within the volume. No action is taken to remove the bad groups, but the groups are reported to the listing file showing the group number, the physical sectors involved, and the file(s) associated with the bad group.

SECTION 1: HDOS 3.02 UTILITIES (Cont)

VERIFY -- Disk Verify Utility (Cont)

/C[ORRUPT]

The "/CORRUPT" switch is used to show the user which groups of the device are corrupt. Corrupt includes any group which is orphanned, multiply allocated, or circullarly linked. This is shown in a map representation indicating which groups have specific attributes. Following the mapping, each corrupt group is analyzed to determine which files are allocated, and how each file joins into the corrupt group. Following the file analysis, each orphanned group is analyzed to determine if any of the known files start or end in the specified orphaned group. In this way, orphans may be reclaimed by their parent files.

/D[IRECTORY]

The directory switch will allow the user to see individual file mappings. It will first give a condensed directory, including file name size, flags, first groups number (FGN), last group number (LGN), and the last sector index (LSI). The flags field will contain the standard HDOS flags as well as four flags created by verify. These flags are:

```
S       - The file is suppressed.
L   - The file is locked against further HDOS flag changes.
W   - The file is write-protected.
C       - The file's allocation is contiguous on the device.
E   - The file is past the volume's end-of-direct mark.
F   - The file's directory entry has a format error.
B   - The program "BAD" has removed sectors from it.
*   - The file is found corrupt by Verify.
```

Following the condensed directory, each file will be mapped to the listing file, and the file's attributes shown, including if it is a .ABS or .PIC file. The size calculations given by the file header may not accurately reflect the true file size.

/F[ULL]

This switch will request verify to dump all critical regions to the output file. These include each directory entry, the GRT, RGT, and VCB. All directory entries, including empty one's will be printed in octal, decimal, and ASCII.

/G[RT]

This switch will allow the user to see all files allocated to any specific group. The section will print each group, its linkages, owners, and backward links. It will search all possibilities for owners to allow caching, multiply, allocated, and other corrupt groups.

SECTION 1: HDOS 3.02 UTILITIES (Cont)

VERIFY -- Disk Verify Utility (Cont)

/R[GT]

This switch will allow the checking of groups flagged as reserved or bad. A cross reference will be made between the GRT and RGT for correspondence. This will aid in the identification of completely overlaid GRT files.

/S[UMMARY]

This section will print out a complete compilation summary, including the number of directory searches, I/O reads, and CPU time used, as well as the device's errors.

/AL[L]

This switch will cause VERIFY to include all of the above options.

/-switch

All the above switches may be preceeded by an "-" character to indicate its negation. For example, /B/-B would negate the bad block scan. The negation is position sensitive, therefore all negations should follow, i.e. /ALL/-B would perform all functions, except the bad block scan.

/FIX

This routine will go through the disk on a file-by-file basis determining who is linked to any part of the list. This routine will concentrate ONLY on those groups determined to be corrupt by the corrupt routine. This operation will attempt to correct the GRT. It will not affect the directory or the RGT files. If any of these files have an inconsistency, this operation MUST not be performed.

/ID

This switch will display the version of VFY.ABS.

/EXP[ENSE]

This will display costs for the whole operation.

SECTION 2: ADVANCED TECHNIQUES

PATH

The PATH command is most useful for those who have a computer system with multiple drives. You may set, display, or clear the system path string. No system checking is performed by this command. Any errors will show up when the PATH is accessed by SYSCMD.SYS. In case the system detects errors in the PATH command, it will issue the phrase 'Check Path,' show you the offending characters, and give the appropriate error message.

You may define your PATH string in several ways. The delimiters are the SPACE, the TAB, the COMMA, the COLON, the SEMICOLON, or nothing. The following examples are all equivalent:

```
SY0 SY1 SY2     SY0       SY1       SY2  (succession of tabs)
SY0,SY1,SY2     SY0:SY1:SY2:
SY0;SY1;SY2     SY0SY1SY2
```

To set a new PATH string: PA text [Example: PA SY0;SY1;SY2]
To display a PATH string: PA<RTN>
To clear a PATH string: PA ~<RTN>

NOTE: [~] Indicates the Tilde sign.

****

TASK

HDOS 3.02 provides a function that is called "TASKing."

A "TASK" is a position-independent program, much like a device driver, which loads and executes below HDOS. Typically, tasks are used to process interrupts, and gain control under the supervision of the Task Manager.

The Task Manager (TMG) provides to the individual tasks complete interrupt control and dispatching, as well as task communication and control services. TMG is itself a task, and must be started (via the START command) before any of its facilities may be made available to the user tasks.

The most obvious function of the Task Manager is that of task identification. Each task started must call upon the Task Manager for identification before any other services may be requested. The fact that a task has called for identification is evidenced by a message issued by TMG stating the task's name, version, and a special "task sequence number," or TSN. It is with the TSN that tasks may be most

SECTION 2: ADVANCED TECHNIQUES (Cont)

TASK (Cont)

easily manipulated by other tasks or programs.

The following task programs are available on your distribution disks:

```
TMG.TAS ..... Task Manager.  MUST be started before tasks will work.
BATCH.TAS ... Task used with JTRA.ABS.
CHAN.TAS .... Shows I/O channel activity on the 25th line.
CLOCK.TAS ... Standard software clock.
CRASH.TAS ... Touch BREAK key to CRASH system.  For curiosity.
ECHO.TAS .... Send screen output to LP.DVD at same time.  Requires
                you to load LP: first.
KEYS.TAS .... Program all 8 function keys.
SCALL.TAS ... All SCALL activity is directed to LP:.
SYSMON.TAS .. Monitors STACK for overflow and S.FASER calls.
TDU.TAS ..... Terminal Debug Utility.
```

For more details on tasking, refer to Appendix 7-A, "Task Manager."

BATCH FILES

To avoid repetitive typing of repeated command sequences HDOS 3.02 has provided a better way of accomplishing this. It allows you to take a series of commands and store them in a special kind of file called a "batch file." This file can be repeatedly used, and it will always have the same result.

A "batch file" is a text file that contains a series of commands. It is easy to create with a standard text editor, such as "Pie" or "Edit19." Batch file names always end in the extension of "BAT." The batch file must always be in ASCII form.

A typical "batch file" is 'AUTOEXEC.BAT,' but you may create other batch files with different file names that end in .BAT. The contents of a typical AUTOEXEC.BAT file is as follows:

```
START CLOCK
PROMPT DAN+>
MOUNT SY1:
```

In the case of AUTOEXEC.BAT, when the disk is booted, after searching for any prologue.sys files and running them, the HDOS 3.02 system then runs AUTOEXEC.BAT. This provides a convenient place to list your typical start-up commands, such as START CLOCK, PROMPT NAME+>, etc. so that they would start up automatically upon boot.

SECTION 2: ADVANCED TECHNIQUES (Cont)

BATCH FILES (Cont)

To execute "batch files" other than AUTOEXEC.BAT, simply type the filename of the batch file, but not the extension, BAT. and then press <RTN>. If HDOS 3.02 does not find the "batch file" in the current directory of the current drive or hard disk partition, it will carry the search to the other drives or hard disk partitions previously specified by the PATH command.

When HDOS 3.02 locates the "batch file," the file's contents are loaded into memory, the first command in the file is displayed, and then the command is executed. When the first command has been completed, HDOS 3.02 shows the system prompt, and then checks to see whether there is a second prompt listed in the "batch file." If there is a second command listed, HDOS 3.02 displays it and then executes it. This process continues until all of the commands are run.

If an executable file named in a batch file requires user input, for example, SPELL.ABS, upon completion of the program, the batch file will automatically execute the next listed command.

```
BLANK.BAT ..... Blanks the screen.  Touch any key to restore.
BLINK.BAT ..... Clears the screen the hard way.
MDRC.BAT ...... Tool for looking at several disks.  Example:
                  MDRC DKO:<RTN>.
SHOWALL.BAT ... Displays HDOS information.
TICTOC.BAT .... Start the system clock first.  Then try this.
```

In addition to names of executable files (those with the extension of BAT or ASM) or valid SYScommands such as PATH, PROMPT, TIME, etc., the following commands are allowable in user-created batch files. The following BATCH files may be found on your distribution disks:

List of batch commands:

BATCH Command	Meaning
AS[k]	Wait for key and save key value.
AS[k] text	Show text, wait for key and save key value.
BIT	Show BIT flags.
BIT S digit	Set specified BIT flag (0 thru 7) to 1.
BIT S	Set all BIT flags (0 thru 7) to 1.
BIT C digit	Clear specified BIT flag (0 thru 7).
BIT C	Clear all BIT flags (0 thru 7).
BIT T digit	Toggle specified BIT flag (0 thru 7).
BIT T	Toggle all BIT flags (0 thru 7).
CB[uf)	Clear console buffer.
COU[nt]	Show system counter value.
COU[nt] +	Increment counter.
COU[nt] -	Decrement counter.

SECTION 2: ADVANCED TECHNIQUES (Cont)

BATCH FILES (Cont)

BATCH Command	Meaning
COU[nt] [=] value	Set counter (0 thru 255).
EC[ho]	Show ECHO state.
EC[ho] text	Show text on screen.
EC[ho] ON	Set ECHO state to ON.
EC[ho] OF[F]	Set ECHO state to OFF.
END	END batch file (usually before physical end).
END C	Exit BATCH file & clear console screen & modes.
END <any argument>	Exit BATCH file & clear only console modes.
GO[to] label	Branch to label (label format is ":label"].
IF [NOT] BIT digit command	Test BIT flags.
IF [NOT] EXI[st] filename command	Test for presence of file.
IF [NOT] COU[nt] = value command	Test counter value.
IF [NOT] ERR[or] = value command	Test last error code value.
IF [NOT] CRC = val cmd	Test last CRC value.
IF [NOT] KEY = val cmd	Test ASK or TRAP keystroke value.
IF [NOT] string = string command	Test string value. [NOTE: White space and the equal sign "=" are string delimiters.
JU[mp] label	Same as GOTO, but searches forward ONLY.
KEY	Show current ASK keystroke value.
KEY alpha	Preset ASK keystroke.
KEY ?<CR|space|tab|?>	Preset special value. CR=null.
PAU[se]	Prompt user & wait for key.
PAU[se] text	Show text, prompt user, & wait for key.
REM [text]	Remark: do nothing.
' [text]	Remark: do nothing.
SH[ift]	Shift command line arguments left one position.
TR[ap]	Trap keystroke on the fly & save it.
WAIT	Wait indefinitely for user to touch any key.
WAIT value	Wait for specified seconds, 0=don't wait.

NOTE: Replaceable parameters %0 thru %9 may be used in BATCH files. %0 is always the BATCH file name, even after SHIFT. %1 thru %9 are the corresponding arguments entered in command. White space is used as the delimiter.

%n = default device name (SY)	%# = active user area (0)
%u = default unit number (0)	%p = active LP unit (0)
%: = default device (SY0:)	%k = ASK keystroke
%X = default extension	

Special characters which may be used in PROMPT, ECHO, PAUSE, and ASK text. Other control codes are not allowed. All other characters print normally.

SECTION 2: ADVANCED TECHNIQUES (Cont)

BATCH FILES (Cont)

#b = the bell character	$: = default device (xxn:)
#d = system date (dd-mm-yy)	$@ = the NULL character
#e = the ESCAPE character	$, = the TAB character
#h = the sequence BS, ' ' ,BS	$_ = the NEW LINE character

Special characters:

#k = the ASK keystroke	$^ = the FORM FEED character
#n = default device name (xx)	$= = The CARRIAGE RETURN char
#p = active LP unit (0)	$> = default system prompt
#s = the SPACE character	$' = the CLICK character
$t = system time (00:00:00)	$$ = the DOLLAR character
$u = default device unit (n)	$~ = the TILDE character
$v = version number (3.02)	$# = active user area (0)
$x = default extension (xxx)	$< = the BACKSPACE character

NOTE: The user must put $_ at the end of the ECHO string to go to a new line. Otherwise, the cursor will remain at the end of the string, wherever it may be on the screen.

****

JOB TRANSLATOR (JTRA) AND BATCH PROCESSOR
by Andy Dessler

The Job Translator (JTRA) and Batch Processor, working in tandem, allow the user to control user inputs in an automated fashion. Interactive commands allow complex tasks to be run based on a number of run time criteria.

To Set Up:

Using HDOS 3.0 and SYSCMD/Plus, simply STart BATCH.TAS. After starting the task, a sign-on message will be printed. The system is now ready to run batch files.

To Run:

The format of the command to run a batch file is:

```
>JTRA [$]job[-library] [var0][,var1][,var2][,var3]...[,var9]
```

'job': this refers to the job name given any group of lines that constitutes one job.

'library': this is the file that contains the job. A library can contain any number of uniquely named jobs. If this parameter is not given, the program assumes that the library is SY0:STANDARD.JCL. The default extensions on the library are SY0: and .JCL.

SECTION 2: ADVANCED TECHNIQUES (Cont)

JOB TRANSLATOR (JTRA) AND BATCH PROCESSOR
by Andy Dessler

To Run: (Cont)

'$': this flag signals that the job is not to be executed, but instead to be written to an .ABS file of the name: SY0:JOB.ABS. After executing this instruction, the job may be run at any time by typing 'job' in response to the SYSCMD prompt with BATCH.TAS loaded.

[var0],etc.: these are the string replacement variables.

Example Runs:

```
    >JTRA DOG
```

-Runs the job 'DOG' located in library 'SY0:STANDARD.JCL'.

```
  >JTRA $DOG
```

-Creates the file 'SY0:DOG.ABS'. This short program can be run anytime and produce the same results of the command:

```
  >JTRA DOG
```

The advantage of this command is that 'SY0:STANDARD.JCL' and JTRA.ABS do not have to be present to run DOG.ABS. In order to run the program, however, BATCH.TAS must be resident.

```
  >JTRA DOG-CAT
```

-Runs the job 'DOG' from the library 'SY0:CAT.JCL'.

```
  >JTRA DOG-SY1:HAMSTER RHINO,ELEPHANT,WALRUS
```

-Runs the job 'DOG' from the library 'SY1:HAMSTER.JCL'. String replacement variable &0 is set to 'RHINO'. &1 is set to 'ELEPHANT'. &2 is set to 'WALRUS'.

```
  >JTRA DOG RHINO,,CANARY
```

-Runs the job 'DOG' from the library 'SY0:STANDARD.JCL'. &0 is set to 'RHINO'. &1 is not set. &2 is set to 'CANARY'.

SECTION 2: ADVANCED TECHNIQUES (Cont)

JOB TRANSLATOR (JTRA) AND BATCH PROCESSOR (Cont)
by Andy Dessler

Libraries:

A library is an HDOS file that contains one or more jobs. The format of these files is:

```
$JOB job_name1
some batch statements and lines of text
$ENDJOB

$JOB job_name2
more batch statements and more lines of text
$ENDJOB

etc.
```

As one can see, the job is identified by the 'job_name'. Thus, to name a job 'DOG' (as from the previous examples), use the line:

```
$JOB DOG
```

followed by the lines that make up the job followed by:

```
$ENDJOB
```

This job can be embedded in a file containing many other jobs.

The 'job_name' is an alpha string (no numbers) up to 6 characters long. No reserved word (BATCH command) can be used.

[A] STRING VARIABLES

Use:

String variables are used to make submit files more flexible by allowing the user to specify certain parameters (such as file names) at run time. String variables are referred to in the form &n where '&' is the ampersand character and n is an integer between 0 and 9. Each integer refers to a unique string variable not to exceed 24 characters in length.

Thus, if &0 is set to 'SY0:', then the line:

```
DELETE &0WALRUS.JCL
```

will be expanded to:

```
DELETE SY0:WALRUS.JCL
```

when the job is run.

SECTION 2: ADVANCED TECHNIQUES (Cont)

JOB TRANSLATOR (JTRA) AND BATCH PROCESSOR (Cont)

by Andy Dessler

Definition:

String variables are defined two ways. The first has already been discussed: command line definition. This way, the variables are defined when the job is submitted.

The second way is by use of the $VAL command. This is a line at the start of your job (immediately following the $JOB statement). The syntax of the command is:

```
$VAL [var0][,var1][,var2]...[,var9]
```

For example, the first two lines of a job could be:

```
$JOB DOG
$VAL COLLIE,LABRADOR,GERMAN SHEPARD
```

This program segment sets &0 to 'COLLIE', &1 to 'LABRADOR', and &2 to 'GERMAN SHEPARD'. Imbedded and leading and trailing blanks are significant.

It is important to know that strings defined on the command line are more signifigant than strings defined on $VAL statements. In other words, if a string is defined in both places (the command line and a $VAL statement), the definition given on the command line is the one used.

For example, if the job 'DOG', whose first few lines are defined above is run with the command line:

```
>JTRA DOG POODLE,,GREAT DANE,BEAGLE
```

then the strings are defined as follows:

&0: 'POODLE' - since this string is defined in both the $VAL statement and on the command line, the command line takes precedence.

&1: 'LABRADOR' - two consecutive commas on the command line tell the translator that no definition is given, thus the program uses the definition given in the $VAL line.

&2: 'GREAT DANE' - again, this string is defined in both places. Hence, the command line value is used.

&3: 'BEAGLE' - no value is given for this string on the $VAL line, hence the value from the command line is used. Note however that EVEN if the string was defined on the $VAL line, the command line value would still be used because the command line is more signifigant.

SECTION 2: ADVANCED TECHNIQUES (Cont)

JOB TRANSLATOR (JTRA) AND BATCH PROCESSOR (Cont)

by Andy Dessler

Definition: (Cont)

If the command to run the job had been:

```
>JTRA DOG
```

then the definitions would rely strictly on the $VAL command line. Hence, &0 would be 'COLLIE', &1 would be 'LABRADOR', and &2 would be 'GERMAN SHEPARD'.

If these are the definitions, then the line:

```
      my dogs are &0s, &1s, and &2s!
```

would be:

```
      my dogs are COLLIEs, LABRADORs, and GERMAN SHEPARDs!
```

A sample job could be:

```
          $JOB DTE
      ' delete a file
      ' NOTE: any line starting with a ' is considered a comment.
      $VAL ,SY0:,.ABS
      ' no default value is given for &0
          '    default values for &0 is 'SY0:' and &1 is '.ABS'
          DELETE &1&0&2
      $ENDJOB
```

If the user runs:

```
>JTRA DTE BASIC
```

will produce:

```
DELETE SY0:BASIC.ABS
```

If the user runs:

```
>JTRA DTE BASIC,SY1:
```

will produce:

```
DELETE SY1:BASIC.ABS
```

If the user runs:

```
  >JTRA DTE BASIC,,.COM
```

will produce:

```
  DELETE SY0:BASIC.COM
```

SECTION 2: ADVANCED TECHNIQUES (Cont)

JOB TRANSLATOR (JTRA) AND BATCH PROCESSOR (Cont)
by Andy Dessler

[B] JTRA COMMANDS

All commands start with a '$'. The '$' must be in column one.

There are two types of variables mentioned in JTRA commands. The first is the 'sense switch.' This is a binary switch that can only be 0 or 1. There are 32 switches named SW1 to SW32. The second is the numeric variable. It is an 8 bit variable that can be set from 0 to 99. It is referred to as ^l where l is any letter from A to Z.

$JOB job_name

This command defines the start of a job. Job_name can be up to 6 characters and must only be alphas.

$ENDJOB

This command marks the end of a job. This command must be present, even if there is only one job in a file.

$LET SWn,value

This command assigns sense switch n (n ranges from 1 to 32) to value. Value can be either 0 or 1. On initial start of the batch task, all switches are set to zero. However, once a switch is set to one, the switch remains set even after the job that set it has ended and others have run. This allows inter-job communication.

$LET ^l,value

Assigns the variable ^l (l can be any letter from A to Z) to value. Value is an integer between 0 and 99. Like sense switches, the initial value of all variables are 0. However, all variables are zeroed after a job has ended.

$LET ^l,^k

This command assigns ^l (l is any letter from A to Z) to ^k (k is any letter from A to Z)

$JUMP label

This is an unconditional jump to 'label'.

$JUMP SWn:label1/label2

This is a conditional jump. If the value of sense switch n is 0, then command is transfered to 'label1'. If the value is 1, then command goes to 'label2'. Label1 or label2 can be omitted. If this occurs and that branch is chosen, then execution continues at the next statement.

SECTION 2: ADVANCED TECHNIQUES (Cont)

JOB TRANSLATOR (JTRA) AND BATCH PROCESSOR (Cont)
by Andy Dessler

[B] JTRA COMMANDS (Cont)

EXAMPLE:

(SW5 is 0): $JUMP SW5:/DOG

This command causes execution to continue with the statement immediately following this one. If SW5 had been one, then execution would have continued with label DOG.

$INQUIRE 'question?',SWn

Asks the user the 'question?' and waits for a yes/no answer. If the answer is YES, then SWn is set to 1. If NO, then SWn is set to 0. A carriage return or any inappropriate character is set to YES.

SECTION 2: ADVANCED TECHNIQUES (Cont)

JOB TRANSLATOR (JTRA) AND BATCH PROCESSOR (Cont)
by Andy Dessler

[B] JTRA COMMANDS (Cont)

$OUTPUT fname
: This command opens the log file 'fname'. The defaults are SY1: and .LOG. All characters (excluding $INQUIRE commands and output during $SUSPEND) that appear on the screen are sent into this file. It is not possible to dismount or reset the disk that is being logged to.

$CLOSE
: This command closes the log file opened with the $OUTPUT command. After closing the file, another may be opened with another $OUTPUT command.

$INR ^l/$DCR ^l
: These commands increment and decrement the numeric variable ^l.

$TEST ^l,SWn
: If ^l is zero, then SWn is set to zero. Otherwise, SWn is set to one.

$SUSPEND
: When this command is encountered, execution in the job is halted until the next SCALL .EXIT is executed.

$VAL variable list
: As discussed earlier, this command (which must appear at the front of the file) assigns string replacement variables.

$LABEL label
: This command assigns 'label' to the statement that follows.

[C] CHARACTER MAPPING AND SPECIAL CHARACTERS

The Job Translator allows the control characters in the job. All controls characters are defined as:

\c where c is any letter (case blind).
Thus, \a (the same as \A) maps to ASCII 01 or cntrl-a.
\D is a cntrl-d.

There are also several specially defined characters, as follows:

SECTION 2: ADVANCED TECHNIQUES (Cont)

JOB TRANSLATOR (JTRA) AND BATCH PROCESSOR (Cont)
by Andy Dessler

[C] CHARACTER MAPPING AND SPECIAL CHARACTERS (Cont)

```
  \; is a carriage return
  \$ is an escape
  \\ is a backslash (\)
```

There is one character that does not convert to any literal character. That is the \# character. When this character is encountered, the program delays for approximately 1/2 second. This allows the user to make jobs more readable by pausing. This helps during fast screen I/O.

[D] PROGRAMMER NOTES:

The JTRA program and BATCH task communicate using SCALL .SUBMIT (123Q). Here are the appropriate setups:

```
  To send a job (it must be completely tokenized):
        (A) = 0
        (HL) = start
        (DE) = end (points at last byte)
  'c' set if error occurs

  Sense Switch Write
        (A) = 1
        (B) = sense switch # (1 - 32)
        (C) = 0,1,2
              0: set switch to 0
              1: set switch to 1
              2: invert switch
  'c' set if error occurs

  Sense Switch Read
        (A) = 2
        (B) = sense switch # (1 - 32)
  Value returned in (A).
  'c' set if error occurs
```

****

SECTION 1: ADVANCED TECHNIQUES (Cont)

TDU UTILITY

by Wayne J. Parnin II

1. Credits:

- *Original H8 version by Andy Dessler, called "DBUG.TSK"
- *Enhanced by Wayne Parnin for H8/H89 and M80
- *Enhanced for HDOS 3.02 by Mighty/Soft

2. Description:

TDU is a basic debugging task, which allows the user to examine and alter the current state of his CPU. TDU is compatible with any HDOS environment as long as SCALL 122Q is not implemented by HDOS.

TDU must be used with DG electronics' SYSCMD/+, but does not use the facilities of TMG; that is, it is a stand-alone task.

TDU cannot set breakpoints in a program; they must be assembled in as SCALL 122Q. This is NOT as bad as it seems, read on.

3. Usage:

TDU is activated by STARTing TDU. TDU will identify itself, but will not make itself obvious in any other way at this time. For details on
? how to start TDU, see page 7-

TDU will trap any SCALL 122Q's in the program being run and will enter its command mode with the prompt 'TDU:'. The user may then use any of the commands listed below.

The user should assemble in SCALL 122Q at all points in his program at which he wishes to examine the CPU state. I usually put breakpoints at the entry and exit of each subroutine. If the breakpoints are put in as conditionally assembled, they are easily turned on and off as needed.

4. Commands:

The user may type any of the following commands in response to the TDU: prompt.

S	Display CPU state. Displays all 8080 registers, with the PSW decoded for easy reading.
M	Display memory locations. Uses each register as a memory pointer and displays 20 bytes for each register. Displays 10 words from current target of stack pointer.

SECTION 1: ADVANCED TECHNIQUES (Cont)

TDU UTILITY (Cont)
by Wayne J. Parnin II

4. Commands: (Cont)

X — Execute. Restores CPU state and continues running the user's program until another SCALL 122Q is encountered. Does an automatic "S" command when a breakpoint is encountered.

Daaaa — Display 128 bytes of memory starting with "aaaa" anded with FFF0H to get us an even paragraph. If "aaaa" is omitted, the next 128 bytes from where you last used the "D" command will be displayed.

Waaaa — Display word located at address "aaaa". User may change the contents of "aaaa" by entering a valid hex number, use "RETURN" to examine the next location, or return to the command prompt by use of "SPACE".

Baaaa — Same as "W" except displays and updates BYTE values.

Aaaaa — Same as "B" except displays and updates ASCII values.

Rr — Displays current contents of register "r", where "r" is any of A, B, C, D, E, F, H, or L. User may update the contents of "r" as in "B", but TDU will not sequence to the next register.

Pp — Same as "r" except accesses a register pair, where "p" is any of B, D, H, or P ("P" is PC). User may update the contents of "p" as in "W", but TDU will not sequence to the next register pair.

NOTE: At the present time TDU does NOT support update of the stack pointer. To display the contents of the stack pointer you must use the "S" or "M" commands.

Ooooo — Sets word input and output offset to "oooo". This is intended for use with relocatable programs developed with M80, but is also useful when debugging device drivers, etc.

Once the offset is set to a non-zero value, all word displays resulting from *any command will display two values: the first is the absolute data. The second, which is enclosed in parentheses, is the same data, relative to the current offset.

SECTION 1: ADVANCED TECHNIQUES (Cont)

TDU UTILITY (Cont)
by Wayne J. Parnin II

4. Commands: (Cont)

```
EXAMPLE:     Offset = 0100H
             :TDU:W1000                  < user types "1000"
             1000 (0F00) :1234 (1134)    < TDU types
```

This indicates that TDU found 1234H at location 1000H. The relative values are 0F00H and 1134H for the address and its contents, respectively.

*NOTE: The display of stack contents during the "M" command is absolute only.

The user may also enter word data and addresses in relative form by appending "/" at the end.

```
EXAMPLE:     Offset = 0100H
             :TDU:WF00/                  < user types "F00/"
             1000 (0F00) :1234 (1134)    < TDU types
```

This is the previous example, but the user entered the address in relative form. TDU added the offset to the user's input to determine the actual address.

The offset capability is unique to TDU among HDOS debugging aids, to my (wjp's) knowledge. This feature makes debugging of relocatable programs very easy.

HELP: Any invalid command character will display a brief help table.

5. Hints

*For M80 users:

-Define "BRKPNT" as a macro:

```
     BRKPNT      MACRO
           IF     DEBUG
           DW    52FFH
           ENDIF
           ENDM
```

-Put BRKPNT in at each key point in your module when you write it.

-Set DEBUG EQU 1 before assembling.

SECTION 1: ADVANCED TECHNIQUES (Cont)

TDU UTILITY (Cont)
by Wayne J. Parnin II

5. Hints (Cont)

-Load the program containing the new module and use the L80 command, /M to display the load map. Note the load point of your module.

-At the first entry to TDU, set the offset to the module load point.

-Debug your module EASILY.

-Write 0's to location of breakpoints which are no longer needed.

-Set DEBUG EQU 0 and reassemble when debugging complete.

*For ASM users:
-Use any XTEXT to define the breakpoint (this will slow down assembly).

OR

-Put in and remove the SCALL's by hand.

-I will upload a split-octal version of TDU shortly.

6. Support:

If you have any questions, problems, suggestions, etc.

Wayne J Parnin II
3 Wagner Way
Hudson, NH 03051
(603) 883-4885
MNET: 70310,362

****

SECTION 2: ADVANCED TECHNIQUES

OPE UTILITY
by Bill Parrott III

****** SCALL

SCALL is entered via CTRL/A in OPE

SCALL is a training tool for new programmers to see what their computer can do. It will issue SCALL's to the system given by the user with parameters for the registers also entered by the user. The register definitions are setup as described below. Below is a sample session:

```
  SAMPLE of SCALL
      ---------------
OCTAL SCALL VALUE>41

**** REGISTER DEFINITIONS ****

PSW = 3.0
BC =
DE =
HL = 40.100

OCTAL SCALL VALUE>
------------------
```

In the above example, SCALL 41Q (.CTLC) was issued for CTRL-C to goto address 40.100. All <RTN>'s entered place a 0.0 in the register. This can be because you want zero in them or the SCALL doesn't need the register. Also, any errors encountered will be printed after the program executes. Also note that if you use this program without knowing what is going on, it may crash. Refer to the programmer's guide for what parameters should be passed.

If the first value given for the register definition is not a number or <RTN>, then SCALL will assume that the value is a string to be passed, such as to open a file. At this point, SCALL will set up a buffer for the string and place the address of the string in the registers currently in use.

****** OPEN

OPEn is used to examine or alter memory locations. Included within the program are 8 registers for use by the user for relocation addresses or just to remember it without using the internal stack. To invoke OPEn you simply reply to HDOS with OPE followed by an optional address to alter.

SECTION 2: ADVANCED TECHNIQUES (CONT)

OPE UTILITY (Cont)
by Bill Parrott III

>OPE [Address]

If the optional address is given, OPEn will evaluate the expression and place it in the current address register. OPEn is a delimiter-driven program, that is, depending on the delimiter, used OPE will perform a certain action on the address and its location. The example below describes how to use OPE delimiters and the relocation registers.

COMMAND FORMAT IN OPEN

OPEn uses the following command format when issuing commands to it. It is similar to using DEC's ODT in form. The format is as follows:

```
prompt (P1);(P2)delimiter
```

where 'prompt' is the prompt character given by OPE, described below. '(P1)' is the first parameter given after the prompt and followed by a ';' if '(P2)' is given. Note that neither '(P1)' or '(P2)' must be given if not wanted. '(P2)' is the second parameter given and generally is used as an offset to the current address. '(P2)' may not include any references to a relocation register internal to OPE. These parameters vary on the mode of input. '(P1)' uses the current mode to take data in, but '(P2)' must be an octal or split octal value.

OPE PROMPTS

The general prompt for OPE is an '@' to tell the user that OPE has no valid address in its address buffer register. In this case '(P1)' is taken as an address to be placed in the address buffer, and '(P2)' is added as an offset to the first value. Also note that only a mode delimiter may be used with this prompt. The other type of prompt that is given is a mode prompt after an address and its contents have been displayed. These are as follows,

'/'	byte mode OCTAL
'\'	word mode OCTAL
'%'	byte mode ASCII

These prompts tell the user that an alter or other operation may be performed on the data in question.

SECTION 2: ADVANCED TECHNIQUES (CONT)

OPE UTILITY (Cont)
by Bill Parrott III

OPE REGISTERS AND USE

OPE's eight registers may be accessed by the user in three different modes. The first is addressing the location where the register is stored. In this mode the user may set up his registers to key entry points in a PIC program or some other point of interest. To address this location the user will use the character '$' followed by the register number to address that register. For example:

```
@$3\ 000.000 \
```

would let the user enter a new address to be defined in register 3.

The second mode for accessing the registers is to get the contents of the register that the user wishes to use. For example, when altering a large number of memory locations and wishing to place the same value in them, the user may want to place the actual value in a register and only use two letters to get the desired contents placed in them. In the example below, the user wants to place 147.376 split octal in memory location 70.000:

```
@$1\ 000.000 \147.376<esc>     ;used to go back to prompt
@70.000\ 254.657 \R1
```

would place the contents of register 1 'R' in the current memory location. The user could continue placing the contents of register 1 (R1) in as many places as desired using offset features, etc. The

The third mode of accessing the registers is during normal procedure when the user would like to place either the current address or the contents of the current address in a register. The command described below does not interfere with normal (P1) or (P2), but is in addition to these features. The format for the command is as follows:

```
^Xdelimiter
```

where '^' tells OPE to perform this command, 'X' is the register to receive the data, and delimiter is either ';' to use the current address or ':' to use the contents of the current address. After using this command the user may continue and place (P1) and (P2) following the command with the delimiter for the command.

OPE DELIMITERS

Since OPEn is a delimiter-driven program, the delimiters are the most important part. Described below are OPE's delimiters and their action in regard to (P1) and (P2). In general, (P1) is placed in either the current memory location, if given, or into the address buffer if no

SECTION 2: ADVANCED TECHNIQUES (Cont)

OPE UTILITY (Cont)
by Bill Parrott III

OPE DELIMITERS (Cont)

address is in the buffer. (P2), on the other hand, is generally used as an offset to the current memory address and will advance the given number after executing the command.

<RTN>

A carriage return is used to close the current location and advance to the next location, opening it and displaying the contents. As described above, (P1) is placed in the memory location, if specified. Unless otherwise stated, (P1) and (P2) will be used as described above.

<ESC>

The escape key is used to return to the OPE prompt. This is when the user wants to address another location but doesn't wish to use the offset feature. Also, in this command, (P2) is ignored.

<CTRL-D>

CTRL-D is used to terminate OPE. (P1) will be placed in the current memory location before OPE exits. (P2) is ignored.

<@>

The character '@' is used to simulate a call instruction when following code in memory. It will alter the current location with (P1), then place the current address in the buffer onto OPE's internal stack, and finally place the contents of the current address into the current address buffer, thus simulating a call instruction. Note that issuing this command when the CALL instruction is at the current address will not work. The user must advance one location and the use '@'.

'>'

The '>' character is used to push the current address onto OPE's internal stack for later use. The address buffer is incremented by one to display the next location.

'<'

The '<' character is used to 'pop' a value off OPE's internal stack and

SECTION 2: ADVANCED TECHNIQUES (Cont)

OPE UTILITY (Cont)
by Bill Parrott III

'<' (Cont)

place it in the current address buffer. This may simulate a RETurn instruction or just pop back to an address pushed on before.

'/'

The '/' character is used as a mode expression. It will alter the current mode and set the current mode to the byte mode octal. It will not affect the current address and will not use (P1) to alter it, only change the current mode. The user may also use '\', or '%' to alter modes as described above.

'_'

The '_' character is used to list bytes. It will list (P2) bytes from the current address and add (P2) to the current address when it is finished.

****

KEYS TASK UTILITY
by Andy Dessler

The Keys program is an interrupt-driven utility written in Task format that allows the user to define the 8 function keys on the H19 terminal to any key sequence up to 256 characters. In addition to defining the keys, the KEYS task also puts labels on the 25th line of the terminal. These labels are displayed above the keys they represent. In order to run this program, the user needs:

SYSCMD/Plus
TMG.TAS (the task manager)
KEYS.TAS

To run the KEYS program, the user should type:

ST TMG (only if not currently loaded)
ST KEYS

at the SYSCMD/plus prompt. The program will then load into memory and request a file name. This file should contain the key definitions.

SECTION 2: ADVANCED TECHNIQUES (Cont)

KEYS TASK UTILITY (Cont)
by Andy Dessler

The default drive is SY0:, and the default extension is .KDF. Just hitting <RTN> defaults to athe file, SY0:STANDARD.KDF.

1. KEY Definition File

The key definition file, the file given KEYS when the program is run, contains the labels and text that are assigned to each key. For example, an average line in the key defintion file might look like:

```
RESET1:R1\;
```

Every line in the KDF (key definition file) must have two parts. The first is a label. This is what will appear on the 25th line of the H19 terminal. In the above example, the label is "RESET1". The label is separated from the commands assigned to the key by a colon [:]. Thus, everything up to the first colon in the line is the label. If the user wants to have a colon in the label, then the colon must be preceded by a backslash (\). For example, if the user wants one of his keys to run the program SY1:D, and wants to have a label of "SY1:D", the user would have the line in the KDF look like:

```
SY1\:D:SY1\:D\;
```

Due to lack of space on the 25th status line, no label should exceed 6 printing characters. Remember that character sequences such as \: constitute only one printing character. Also, all other SYSCMD/Plus character mapping rules apply. For example, a backslash [\] must be represented by \\, to distinguish it from other character combinations.

After the label comes the text of the key. This text should appear exactly as you would type it on the console, except for certain characters that must be represented by two-character sequences. For example, if the user wants a key that runs the assembler on SY1:, and then assembles a program called ZOT, the user would normally type:

```
SY1:ASM (carriage return)
ZOT=ZOT (carriage return)
```

In the keys program, if the user wanted the above text to be a key with the label of "ASM Z", that sequence would be represented by:

```
ASM Z:SY1\:ASM\;ZOT=ZOT\;
```

Note the use of \; as a carriage return. This must be done because a carriage return signals the end of that particular key. See section 3 for a complete listing of all character mapping sequences. Remember

SECTION 2: ADVANCED TECHNIQUES (Cont)

KEYS TASK UTILITY (Cont)
by Andy Dessler

1. KEY Definition File (Cont)

that each KDF contains eight lines, each one defining a key. The first line defines key f1, the second one defines f2, and so on, continuing from left to right. If the user decides that he/she does not wish to use a key (just leave it as a null function key), then that line should have just a colon [:].

For example, a typical KDF would look like:

```
CPS:SY1\:CPS
PIP:PIP\;
RESET1:R1\;
ASM:SY1\:ASM\;
:
:
:
```

The first line, which that defines function key f1, commands the computer to run the program SY1:CPS. The second line, which defines f2, commands the computer to run PIP. The third key, which defines f3, RESETs SY1:. The fourth key, which defines f4, runs SY1:ASM. Keys f5, BLUE, RED and GRAY are undefined.

2. Program Operation

After entering the KDF file name, the program will begin operation. All the user has to do now is hit the function key that corresponds with the command sequence that is desired. The labels on the 25th status line correspond with the keys (in other words, the label desired appears over the function key). For instance, if the above example KDF was resident in memory and the user wanted to run PIP, then the user merely hits the f2 key.

This program also intercepts CTRL-X. This key toggles the program on and off. For example, when the program is running, if a CTRL-X is typed, then the 25th status line is cleared and the program is temporarily disabled. When a CTRL-X is again typed, the 25th line labels are displayed again, and the program is reenabled.

Also, the function keys are dynamically redefineable. In other words, if the KEYS program has already been loaded, but the user wishes to have another set of key definitions in memory, then the user simply reSTarts the KEYS program. All the user has to do is type:

```
ST KEYS
```

SECTION 2: ADVANCED TECHNIQUES (Cont)

KEYS TASK UTILITY (Cont)
by Andy Dessler

2. Program Operation (Cont)

Then the user gives the computer the new .KDF filename, and the keys are redefined.

3. Mapping Characters

In the KEYS program, certain characters, commands, or key sequences must be represented by a "word" of two characters, as per the following list:

```
\;    Carriage Return
\$    Escape
\:    Colon
\\    Backslash
\A      Ctrl/A
\B      Ctrl/B
 .
 .
 .
\Y      Ctrl/Y
\Z      Ctrl/Z
```

4. Program Cautions

In using this program, there are certain limitations the user should observe. First, the KDF should not exceed 4 sectors (if you purchased the source code, then you can change this by modifying the PAGE EQU).

Also, KEYS uses SCALL 111Q, .PPRES. Any other use of this SCALL will cause terrible results.

If the user attempts to use a function key while another is in progress, the program will ignore the second function key. Also, if the user decides to not put a label on a key, then the first character should be a colon [:]. For example, if the user wants the key to MOUNT SY1:, but he does not want the .KDF command line, the key would appear as:

```
:M1\;
```

This would leave blank spaces where the label would have appeared.

SECTION 2: ADVANCED TECHNIQUES (Cont)

KEYS TASK UTILITY (Cont)
by Andy Dessler

Some programs (such as Txtpro and PIE) use the function keys. Before operation of these or any other program that uses the function keys, the KEYS program should be disabled through CTRL-X. Failure to do so will cause serious problems for the program.

Each key sequence must be completely contained on one line. The line can be as long as the user desires, but must be contiguous. Any embedded carriage returns must be defined as a \;. Also, the line should not exceed 256 characters. Note that the special character sequences, or words, constitute only one character.

SECTION 3: SYSTEM ANALYSIS

SYSTEM ANALYSIS PROGRAMS

HDOS 3.02 provides a comprehensive disk analysis, using a series of associated .ABS files. A summary of these files and what they do follows.

DFD.ABS .... Displays a directory of deleted files on any mounted disk specified.

DVL.ABS .... Displays volume label sectors on any mounted disk specified.

DVT.ABS .... Displays the contents of the device table for the system disk.

IOT.ABS .... Provides comprehensive dump and interpretation of the I/O Channel Table.

MAP.ABS .... Displays all the magic addresses for HDOS 3.02.

USR.ABS .... Calculates and displays CPU speed.

Examples of the contents of selected files are as follows:

= = = = = = = = = = = = = = = = = =

DFD -- Deleted Files Directory

Usage: DFD [DVn:]

DVn: is any valid HDOS device (SY1:, DK0:, etc.)

The directory on the specified disk will be processed.
Only deleted files will be displayed.

The form of the output is:

```
DFD -- Deleted Files Directory -- by Mighty/Soft -- Ver 3.0 (dd-mmm-yy)

Name    .Ext FGN LGN LSI  Created  Time  Flags---  Accessed A/C Status

?$$$$$$$.$$$ nnn nnn nnn dd-mmm-yy 00:00 SLWCABDU dd-mmm-yy nnn status...
?$$$$$$$.$$$ nnn nnn nnn dd-mmm-yy 00:00 SLWCABDU dd-mmm-yy nnn status...
?$$$$$$$.$$$ nnn nnn nnn dd-mmm-yy 00:00 SLWCABDU dd-mmm-yy nnn status...

DFD -- nnn deleted files.
```

SECTION 3: SYSTEM ANALYSIS (Cont)

SYSTEM ANALYSIS PROGRAMS (Cont)

DFD - Deleted Files Directory (Cont)

The status can be one of the following:

```
Complete  = File is possibly complete and could be recovered.
Partial B = File is incomplete, only first part was found.
Partial M = File is incomplete, only middle part was found.
Partial E = File is incomplete, only last part was found.
Lost File = File is gone, all sectors are reallocated.
```

= = = = = = = = = = = = = = = = = = =

DVL -- Display Volume Label Sector

Usage: DVL DVn:

DVn: is any valid HDOS device (SY1:, DK0:, etc.)

The label sector on the specified disk will be displayed.

HDOS 3.0 and after

```
Volume Number:          nnn     oooQ
Long Volume Number:     nnnnn   xxx.xxxA
Volume Label: "... text ..."
Trailing Bytes:         oooQ    oooQ    "aa"
Initialized by HDOS Version 3.? on dd-mmm-yy.
Volume Type:            system/data/??????
RGT Sector Index:       nnnnn
GRT Sector Index:       nnnnn
First Directory Sector: nnnnn
Volume Size:            nnnnn Sectors
Physical Sector Size:   nnnnn Bytes
Sectors per Group:      nnn
Sectors per Track:      nnn
Volume Flags:           bbbbbbbb    (sides/tpi/type/???)
```

HDOS 2.0

```
Volume Number:          nnn     oooQ
Volume Label: "... text ..."
Trailing Bytes:         oooQ    oooQ    "aa"
Initialized by HDOS Version 2.0 on dd-mmm-yy.
Volume Type:            system/data/??????
RGT Sector Index:       nnnnn
```

SECTION 3: SYSTEM ANALYSIS (Cont)

SYSTEM ANALYSIS PROGRAMS (Cont)

DVL -- Display Volume Label Sector (Cont)

```
GRT Sector Index:       nnnnn
First Directory Sector: nnnnn
Volume Size:            nnnnn Sectors
Physical Sector Size:   nnnnn Bytes
Sectors per Group:      nnn
Sectors per Track:      nnn
Volume Flags:           bbbbbbbb    (sides/tpi/type/???)
```

An example follows:

```
Volume Number           2         002Q
Long Volume Number      2         000.002A
Volume Label: HDOS 3.02 - 48 TPI System Disk
Trailing Bytes          040Q      000Q     "."
Initialized by HDOS Version 3.0 on 13-Jul-89
Volume Type             System
RGT Sector Index        12
GRT Sector Index        444
First Directory Sector  420
Volume Size             1278 Sectors
Physical Sector Size    256 Bytes
Sectors Per Group       6
Sectors Per Track       16
Volume Flags            00000001 (Double-Sided Floppy Disk)
```

HDOS 1.6

```
Volume Number:          nnn     oooQ
Volume Label: "... text ..."
Trailing Bytes:         oooQ    oooQ    "aa"
Initialized by HDOS Version 1.6 on dd-mmm-yy.  What an old diskette !
Volume Type:            system/data/??????
GRT Sector Index:       nnnnn
First Directory Sector: nnnnn
Sectors per Group:      nnn
```

HDOS 1.5 and before

```
Volume Number:          nnn     oooQ
Volume Label: "... text ..."
Trailing Bytes:         oooQ    oooQ    "aa"
```

SECTION 3: SYSTEM ANALYSIS (Cont)

SYSTEM ANALYSIS PROGRAMS (Cont)

DVL -- Display Volume Label Sector (Cont)

HDOS 1.5 and before

```
Initialized by HDOS Version 1.? on dd-mmm-yy.  GOLLY !! What an old
                                               diskette !
Volume Type:            system/data/??????
GRT Sector Index:       nnnnn
First Directory Sector: nnnnn
Sectors per Group:      nnn
```

= = = = = = = = = = = = = = = = = =

DVT -- Contents of Device Driver Table

```
Device Table FWA:        357.152

Device Name:     TT:
********************

Device Driver Address:   366.114
Driver Byte Length:      004.264 (1204)
Driver Group Address:    10
Set Preamble Length:     6 Sectors

Driver Residence Flag:   01000111  (In memory/Locked/Fixed)

Device Flag:             00010110  (Read/Write/Characters)
Mounted Units Mask:      00000001  (Unit 0 Available)
Maximum Number of Units: 1

Unit Specific Data At:   357.142

Unit 0:

Unit Specific Flag:      00010110  (Read/Write/Characters)


Device Name:     SY:
********************

Device Driver Address:   025.252
Driver Byte Length:      006.233 (1691)
Driver Group Address:    29
Set Preamble Length:     2 Sectors

Driver Residence Flags:  10001111  (In memory/Locked/Fixed)
```

SECTION 3: SYSTEM ANALYSIS (Cont)

SYSTEM ANALYSIS PROGRAMS (Cont)

DVT - Contents of Device Driver Table (Cont)

DEVICE NAME: SY: (Cont)

```
Device Flag:              10001111  (Dir/Read/Write/Random/Notify)
Mounted Units Flag:       00000011  (Units 0,1, Mounted)
Maximum Number of Units: 3

Unit Specific Data AT:    357.112

Unit 0:

Unit Specific Flag:       10001111 (Dir/Read/Write/Random/Notify)
Sectors Per Group:        6
Address of GRT:           375,000
GRT Sector Number:        444
DIRECT Sector Number:     420

Unit 1:

Unit Specific Flag:       10001111 (Dir/Read/Write/Random/Notify)
Sectors Per Group:        10
Address of GRT:           376,000
GRT Sector Number:        870
DIRECT Sector Number:     840

Unit 2:

Unit Specific Flag:       10001111 (Dir/Read/Write/Random/Notify)
Sector Per Group:         0
Address of GRT:           377,000
GRT Sector Number:        0
DIRECT Sector Number:     0
```

Device Name: UD:

```
Device Driver Address:    342.276
Driver Byte Length:       072.242 (2722)
Driver Group Address:     65
Set Preamble Address:     2 Sectors

Driver Residence Flag:    00000100 (Write)
Mounted Units Mask:       11111111 (Units 0,1,2,3,4,5,6,7 Available)
Maximum Number of Units: 8
```

SECTION 3: SYSTEM ANALYSIS (Cont)

SYSTEM ANALYSIS PROGRAMS (Cont)

Device Name: UD:(Cont)

Unit Specific Data AT: 357.012

Unit 0:

Unit Specific Flag: 00000100 (Write)

Unit 1:

Unit Specific Flag: 00000100 (Write)

Unit 2:

Unit Specific Flag: 00000100 (Write)

Unit 3:

Unit Specific Flag: 00000100 (Write)

Unit 4:

Unit Specific Flag: 00000100 (Write)

Unit 5:

Unit Specific Flag: 00000100 (Write)

Unit 6:

Unit Specific Flag: 00000100 (Write)

Unit 7:

Unit Specific Flag: 00000100 (Write)

Device Name: DK:

Device Driver Address: 014.167
Driver Byte Length: 007.150 (1896)

Driver Group Address: 79
Set Preamble Length: 2 Sectors

Driver Residence Flag: 00000011

SECTION 3: SYSTEM ANALYSIS (Cont)

SYSTEM ANALYSIS PROGRAMS (Cont)

DVT - Contents of Device Driver Table (Cont)

```
Device Name:    DK: (Cont)
**************************

Device Flag:             10001111 (Dir/Read/Write/Random/Notify)
Mounted Units Mask:      00000000 (No Units Mounted)
Maximum Number of Units: 2

Unit Specific Data At:   356.372

Unit 0:

Unit Specific Flag:      10001111 (Dir/Read/Write/Random/Notify)
Sectors Per Group:       0
Address of GRT:          000.000
GRT Sector Number:       0
DIRECT Sector Number:    0

Unit 1:

Unit Specific Flag:      10001111 (Dir/Read/Write/Random/Notify)
Sectors Per Group:       0
Address of GRT:          000.000
GRT Sector Number:       0
DIRECT Sector Number:    0


Device Name:     LI:
********************

Device Driver Address:   014.167
Driver Byte Length:      005.014 (1292)
Driver Group Address:    130
Set Preamble Address:    2 Sectors

Driver Residence Flag:   00000100 (Write)
Mounted Units Mask:      11111111 (Units 0,1,2,3,4, Available)
Maximum Number of Units: 5

Unit Specific Data At:   356.322


Unit 0:

Unit Specific Flag:      00000100 (Write)
```

SECTION 3: SYSTEM ANALYSIS (Cont)

SYSTEM ANALYSIS PROGRAMS (Cont)

DVT - Contents of Device Driver Table (Cont)

Unit 1:

Unit Specific Flag: 00000100 (Write)

Unit 2:

Unit Specific Flag: 00000100 (Write)

Unit 3:

Unit Specific Flag: 00000100 (Write)

Unit 4:

Unit Specific Flag: 00000100 (Write)

= = = = = = = = = = = = = = = = = =

IOT - I/O Channel Table Display

This program will provide the user with a comprehensive dump and interpretation of the HDOS I/O channel table.

```
Usage:  IOT                    ; show all channels
        IOT n                  ; show specific channel
```

The following information is displayed for each requested I/O channel:

```
Link to Next Channel:   xxx.xxx
Device Driver Thread:   xxx.xxx
File Type Flags:        bbbbbbbb        (Dir/Read/Write/Random/Char/???)
Address of GRT:         xxx.xxx
Sectors per Group:      nnn
Current Group Number:   nnn
Current Sector Index:   nnn
Last Group Number:      nnn
Last Sector Index:      nnn
Device Table Address:   xxx.xxx
Dir Entry Sector No.:   nnnnn
File Name:              xxn:$$$$$$$$.$$$
Creation Time:          hh:mm
Number of Accesses:     nnn
File Flags:             bbbbbbbb     (Sys/Lock/Prot/Contig/Arc/Bad/Del/Usr)
User Area Mask:         bbbbbbbb
```

SECTION 3: SYSTEM ANALYSIS (Cont)

SYSTEM ANALYSIS PROGRAMS (Cont)

IOT - I/O Channel Table Display (Cont)

```
First Group Number:     nnn
Last Group Number:      nnn
Last Sector Index:      nnn
Creation Date:          dd-mmm-yy
Last Access Date:       dd-mmm-yy
```

= = = = = = = = = = = = = = = = = =

MAP - Magic Addresses for HDOS 3.02

System High Memory:	377.377 (FFFF)	HDOS 3.0a Magic Addresses		
HDOS Scratch Area:	373.000 (FB00)			
SY: Driver FWA:	357.261 (EFB1)	UIVEC 7:	000.200	(0080)
Device Table FWA:	357.152 (EF6A)	UIVEC 6:	000.316	(00CE)
Channel Table FWA:	355.254 (EDAC)	UIVEC 5:	000.316	(00CE)
Task Table FWA:	Not Available	UIVEC 4:	362.210	(F288)
HDOS Resident FWA:	330.226 (D896)	UIVEC 3:	370.117	(F84F)
System Resident FWA:	330.226 (D896)	UIVEC 2:	000.316	(00CE)
User Memory FWA:	042.200 (2280)	UIVEC 1:	334.162	(DC72)
Type-Ahead Buffer FWA:	037.224 (1F94)			
Editor Buffer FWA:	037.057 (1F2F)	NMIVEC:	025.342	(15E2)
Prompt Buffer FWA:	036.312 (1ECA)			
Path Buffer FWA:	036.145 (1E65)	RST 7:	040.061	(2031)
Subst. Buffer FWA:	036.000 (1E00)	RST 6:	040.056	(20E2)
Batch Buffer FWA:	035.000 (1D00)	RST 5:	040.053	(202B)
System Label FWA:	034.000 (1C00)	RST 4:	040.050	(2028)
HDOS Installed Size:	026.157 (166F)	RST 3:	040.045	(2025)
HDOS Data Link Addr:	026.053 (162B)	RST 2:	040.042	(2022)
SCALL Dispatcher:	000.200 (0080)	RST 1:	025.354	(15EC)
Feature Mark Addr:	000.005 (0005)	RST 0:	041.013	(210B)

****

APPENDIX 7-A: TASK MANAGER

by Bill Parrott and David Carroll

The TASK MANAGER was designed and written to provide a vehicle for managing running tasks within the HDOS environment. To this end, the TASK MANAGER (TMG) provides to the individual tasks complete interrupt control and dispatching, as well as task communication and control services. TMG is itself a task and must be started (via the START command) before any of its facilities can be made available to the user tasks.

The most obvious function of the TASK MANAGER is that of TASK identification. Each TASK started must call upon the TASK MANAGER for identification before any other services may be requested. The fact that a TASK has called for identification is evidenced by a message issued by TMG stating the TASK's name, version, and a special 'TASK sequence number' or TSN. It is with the TSN that tasks may be most easily manipulated by other TASKs or programs.

Any TASK wishing to identify itself to TMG must contain in its resident section a structure called a 'TASK BLOCK.' The task block contains certain information used by the TASK MANAGER and certain other programs to identify the TASK by name, find the TASK in memory, and provide a simple means of determining the TASK's current status. The task block is defined in the file TASKDEF.ACM, which is included on the SYSCMD/Plus distribution disk. Also included in TASKDEF.ACM are complete descriptions of each of the support services provided by TMG. It follows that with the introduction of tasks into the system, there may occur errors from time to time, which are neither covered in the HDOS reference manual nor included in ERRORMSG.SYS. These errors are defined individually in the appendices of this manual and should be added to ERRORMSG.SYS. Refer to Appendix 3-A, page 3-33 for details concerning ERRORMSG.SYS.

Rather than explain in detail each facet of the TASK MANAGER, it is felt that the experienced programmer can, using the examples provided, become quickly proficient at writing his or her own custom tasks. Each of the sample programs is heavily documented, providing an excellent learning tool for both the novice and the experienced assembly language programmer.

TASK PROGRAMMING

A 'TASK' is a position independent program, much like a device driver, which loads and executes below HDOS. Typically, TASKS are used to process interrupts, and gain control under the supervision of the TASK MANAGER. The actual TASK program consists of two parts. The first part is the task initialization code. This is called by SYSCMD/Plus when the TASK is first loaded into memory. TASK initialization may include TASK identification, requesting interrupt service from TMG, prompting for user input, allocating buffers, etc. Nearly all TASKS will perform the first two functions of identification and interrupt

APPENDIX 7-A: TASK MANAGER (Cont)
by Bill Parrott and David Carroll

service. It should be noted that a TASK may not request interrupt service, without first identifying itself to TMG. If a TASK wishes, it may elect not to utilize the services of the TASK MANAGER and perform all vector initialization itself. This is not the recommended practice, however, since control of the TASK will be lost once initialization is complete. Generally, upon completion of TASK start-up, the task initialization code is removed from memory by SYSCMD/Plus to provide for maximum memory utilization. This removal of code is controlled by the TASK when control is returned to SYSCMD/Plus. The (PSW) is used to indicate the type of exit which is to be performed.

If the 'C' flag is set, SYSCMD/Plus will reset the system low memory limit to the value contained in the (HL) register. If the 'C' flag is clear, the low memory bound will remain unchanged. To remove the TASK initialization code from memory, the TASK would set the contents of the (HL) register to the end of that code, set the 'C' flag, and return. Note that if this technique is to be used, ALL resident code MUST be located after or at a higher address than the initialization code. The contents of the (A) register have exactly the same effect as when a SCALL .EXIT is performed. If the (A) register contains a zero, the exit is said to be normal. If the (A) register is not zero, the exit is 'aborted'. When an aborted exit is performed, either by an .ABS program or by a TASK, the type-ahead buffer is emptied, the console ports are re-initialized, all open channels are cleared, memory is cleared of all non-locked devices and overlays, and the system memory limits are reset. The second portion of the task program is the TASK resident code. This is the actual run-time TASK program which processes interrupts or whatever. This code may perform any function the user feels appropriate within the following constraints.

If the interrupt is an SCALL, *ALL* registers with the exception of (PSW), which is saved by TMG, must be preserved. During an SCALL, the TASK will receive control in the following manner:

```
(HL)   = User's return address  (pointing to SCALL)
(SP+0) = Return to TASK MANAGER
(SP+2) = TASK MANAGER's (HL)
(SP+4) = User's (PSW)
(SP+6) = User's (HL)
```

The TASK may determine whether to process the SCALL by executing the following set of instructions:

```
MOV     A,M             ; Get function code
CPI     .code           ; Compare for right code
RNE                     ; Not equal.  Don't process it.
 .
 .
```

APPENDIX 7-A: TASK MANAGER (Cont)
by Bill Parrott and David Carroll

If the TASK intends to handle the SCALL, the TASK MANAGER's registers should be popped from the stack and the SCALL processed normally.

If the interrupt is other than an SCALL (vectors 1-6), control will be passed in the following manner:

```
(SP+0) = Return to TMG
(SP+2) = Return to user program via TMG
(SP+4) = TMG's (HL)
(SP+6) = TMG's (BC)
(SP+8) = Return to user via $RSTALL
(SP+10) = (SP+18) = User's registers
(SP+20) = User's interrupted (PC)
```

Since all user registers are saved by TMG, it is not necessary for the TASK to preserve registers, provided that the task returns to the user via TMG or $RSTALL. If the interrupt is a clock interrupt, the following must be taken into consideration. The interrupt is a real-time event and must be processed in as little time as possible since clock interrupts are only 2ms apart and are used for critical timing within HDOS. There may be ABSOLUTELY NO SCALLs, and interrupts must NOT be enabled by the TASK. (SCALLs always re-enable interrupts!). Within these simple constraints, tasks may do as much or as little processing as is needed to perform the desired function. A task must be 4k or less in size, including the initialization code. In most TASKS, this equates to approximately 17-18 sectors (16 sectors for TASK plus 1-2 sectors for PIC table).

The general form of a task program is given here for easy reference, however, more can be learned by studying the tasks provided.

```
TITLE 'Task Sample Format' STL 'System Definitions'

*** MYTASK - Sample TASK * * This is a sample task program to show the
*            recommended  structure of  a  TASK  and to  point out all
*            required components
*
*       Required components are flagged with '+++'.
*
*       Note that this example assumes use of TMG.

VER     EQU     1               ; Version #             +++
SUBV    EQU     0               ; Sub-version #         +++

<<      Necessary XTEXT's       >>

        XTEXT   TASKDEF                                 +++

        STL     'Initialization Code'
```

APPENDIX 7-A: TASK MANAGER (Cont)
by Bill Parrott and David Carroll

```
        EJECT
        CODE    PIC             ; Position Independent  +++

        DB      TASKID          ; Identify as a TASK    +++



***     Initialization
*
TSKINIT LXI     H,TASBLK        ; Addr. of task block   +++
        MVI     B,TAS.ID        ; TMG function code     +++
        SCALL   .TASK           ; Identify TASK         +++

        PUSH    PSW             ; Save TSN

<<      Other initialization processing    >>

        POP     PSW             ; (A) := TSN

        MVI     C,1             ; Int Vector #1
        MVI     B,TAS.RIS       ; TMG function code     +++
        LXI     H,TSKMAIN       ; Processor Address     +++
        SCALL   .TASK           ; Request service       +++
        JC      ABORT           ; Had an error

        LXI     H,TSKMAIN       ; Where to set new memory bound
        XRA     A               ; Normal Exit
        STC                     ; Tell SYSCMD to squash us.

        RET                     ; Return to SYSCMD      +++

ABORT   PUSH    PSW             ; Save error
        CALL    $TYPTX
        DB      NL,'TASK Initialization Error ...',240Q
        POP     PSW             ; (A) := Error code
        MVI     H,BELL          ; Ding bell
        SCALL   .ERROR          ; HDOS to print message
        LXI     H,TSKEND        ; Where this TASK ends
        MVI     A,1             ; Aborted Exit.
        STC                     ; SYSCMD will squash us.
        RET                     ; Return to SYSCMD

        STL     'Task Resident Code'
        EJECT
```

APPENDIX 7-A: TASK MANAGER (Cont)
by Bill Parrott and David Carroll

```
***     TSKMAIN - TASK time code.
*

TSKMAIN EQU     *               ; Resident code begins here

<<     Task time processor     >>

        RET                     ; Exit to TASK MANAGER

        STL     'Data Areas'
        EJECT



***     TASK Block
*

TASBLK  DB      'MYTASK',0,0            ; TASK Name             +++
        DB      VER*16+SUBV             ; Version               +++
        DB      'foo!'                  ; Identification        +++
        DB      TSS.ACT+TSS.UFP         ; Status                +++
        DW      TSKMAIN                 ; Start of TASK         +++
        DW      TSKEND-1                ; End of TASK           +++
        DW      0                       ; Processor Address     +++
        ERRNZ   *-TASBLK-TSB.LEN                                +++

TSKEND  EQU     *                       ; The end ...           +++

        END                                                     +++
```

Note the processor address contained as part of the TASK block. If a TASK wishes to be notified of suspension and/or re-activation by the TASK MANAGER, this must contain the address of the TASK's processor to handle such notification. When either suspension or re-activation of a TASK occurs, TMG will call the TASK's processor with the (A) register set to the value corresponding to the function being performed. If (A) contains TAS.DEA, the TASK is being de-activated or suspended. If (A) contains TAS.REA, the TASK is being re-activated. TASKS may perform such housekeeping as disabling interrupts and restoring tables and vectors when being de-activated. When re-activated, a TASK might re-enable interrupts, etc. Once a TASK has been de-activated, TMG will no longer pass control to that TASK for requested interrupts. When the task is re-activated, TMG will automatically resume processing of the interrupt for the TASK. It is important that a TASK insure that no interrupts which may go unserviced can happen after de-activation. Also, the ONLY way to re-activate a suspended TASK is via the TASK MANAGER.

APPENDIX 7-A: TASK MANAGER (Cont)
by Bill Parrott and David Carroll

SAMPLE TASKS

Four (4) TASKS have been been included with SYSCMD/Plus so that the user may be readily introduced to the concept of TASKS and to provide a means by which the user may easily learn about the structure of the TASKS themselves. As the user becomes more comfortable with TASKS and begins to develop his own, he will realize the enormous potential of this powerful tool. Below are described the TASKS included with SYSCMD/Plus, including instructions for operation. All of these TASKS have been provided for the user in source as well as object code.

NOTE: All tasks included here utilize the services of the SYSCMD/Plus Task Manager. TMG (included on the SYSCMD/Plus distribution disk) must therefore be STARTed before any of these tasks may be run.

CLOCK80 (NOTE: HDOS 3.0 USES 'CLOCK')

This TASK provides the system with a real-time clock identical in function to that found in the D-G FPM/80 monitor. This TASK may be used in systems not utilizing this monitor to enable use of the TIME command described in Section I. This TASK contains processors for both the 2ms clock interrupt generated by the system and for a special SCALL called .CLOCK. The value of .CLOCK is 376Q. Clock is used to determine the location in the system of the CLKPTR register. CLKPTR contains the address of the actual clock data. This clock keeps hours, minutes, seconds, and milliseconds/2 (2ms TICs). In addition, the speed at which the clock ticks may be altered by changing the proper value in the clock table. The format of this table is described in detail in the source code. When the clock reaches midnight, CLOCK80 will automatically increment the HDOS system date. No check is made, however, for the end of a month or year. CLOCK80 contains no Z80 instructions.

CLOCK80 may be started by entering 'START CLOCK80' at the system prompt. To set or display the time, the TIME command may be used.

ECHO

ECHO is designed to simulate a system console logging function within HDOS. That is, all characters which appear on the system console will be logged (or ECHOed) on a selected output listing device. ECHO has been included as an excellent example of cooperation between a TASK and the rest of the HDOS operating environment. It demonstrates several powerful techniques which may be employed when processing SCALLs. In addition, ECHO is a very useful TASK for monitoring system operation and preparing hard-copy output of program execution. ECHO processes only the SCALL vector and contains no Z80 instructions.

APPENDIX 7-A: TASK MANAGER (Cont)
by Bill Parrott and David Carroll

ECHO (Cont)

ECHO is started by typing 'START ECHO' at the system prompt. ECHO will identify itself and request the user input the name of a serial listing device to act as the log device. This must not be a disk file. In addition, ECHO will not permit the entry of any unknown device or of TT:. To abort the task at this point, the user may enter CTRL-D. When the user has entered the device name and ECHO has verified its validity, the TASK MANAGER will identify ECHO, and control will be returned to SYSCMD/Plus. To enable and/or disable ECHO, please refer to the LOG command described in Section I.

CHAN

This TASK monitors the system I/O channel table by processing all file open/close related SCALLs. A status display is maintained on the H19 25th line indicating the status of the system channels. Channels are numbered from -1 thru 5. Each entry consists of five (5) flags. When all five flags are displayed as '.', the channel is closed. The following values may appear in the flags: 'C' - File is open in character mode; 'U' - File is open for UPDATE; 'W' - File is open for WRITE; 'R' File is open for READ; 'D' - File is open on directory device. CHAN processes the SCALL vector and may be run ONLY in a Z80 system. Attempting to run CHAN in an 8080 system will result in a fatal system error.

CHAN may be started by typing 'START CHAN' at the system prompt.

HDOS SOFTWARE REFERENCE MANUAL

HEATH DISK OPERATING SYSTEM

VERSION 3.02

CHAPTER 8

THEORY OF OPERATION

HEATH DISK OPERATING SYSTEM

SOFTWARE REFERENCE MANUAL

VERSION 3.02

 Through the years it continued to be improved by successive revisions which included 1.5, 1.6, and finally 2.0. It was entered into public domain on 19 July 1989 per letter by Jim Buszkiewicz, Managing Editor, Heath Users' Group, P.O. Box 217, Benton Harbor, MI 49022-0217 (616)982-3463. A copy of this letter is available for public inspection.

This manual is indicative of further improvements and provides for the latest revision, HDOS 3.0 and HDOS 3.02. Revision 3.0 is detailed in chapters 1, 2, and 3, while chapters 4 ,5, 6, 7, 8, 13 and 14, are related to revision 3.02. Chapters 9 through 12, with minor

improvements, are essentially picked up from the original HDOS 2.0 manual. Indeed, HDOS is still alive and well!

Chapter 8, Theory of Operation, provides a description of the inner workings of HDOS.

SPECIAL DISCLAIMER: The Heath Company cannot provide consultation on either the HDOS Operating System or user-developed or modified versions of Heath software products designed to operate under the HDOS Operating System. Do not refer to Heath for questions.

Instead, you are invited to direct any questions concerning the Heath Disk Operating System (HDOS) to Mr. Kirk L. Thompson, Editor "Staunch 89/8" Newsletter, P.O. Box 548, #6 West Branch Mobile Home Village, West Branch, IA 52358.

TABLE OF CONTENTS

INTRODUCTION

Within the following paragraphs, detailed information concerning the inner workings of HDOS is presented. It is designed to tell you all you ever wanted to know about the HDOS that functions "behind the scenes."

You will learn about the detailed construction of a computer disk; magnetic domains, bits and bytes, tracks and sectors, groups, clusters, the directory, the buffer, disk file storage and much, much more. In addition, you will learn about how the disk drive interfaces with the floppy disks, including details about disk access time, seeking, read/write rate, the HDOS Operating System, the heart of HDOS, the disk nucleus, and the command processor.

It is hoped that this chapter will be as interesting as it is unique.
****

Theory of Operation

THE DISK

The disk is a circular sheet of Mylar (DuPont Registered Trademark) coated with a magnetic oxide. When you insert the disk into the drive and close the drive door, the disk settles onto a spindle attached to the drive door. As the disk rotates at approximately 300 revolutions per minute, a fixed electromagnetic read/write head passes over the oxide material, or "medium," and interacts with it by means of a magnetic field. This combination of head and medium is very similar to that of a magnetic tape recorder.

DOMAINS

The medium consists of millions of tiny magnetic particles called "domains." Each of these domains is magnetized, and the polarity of a given domain describes the binary data stored on the disk.

BITS AND BYTES

If the read/write head is "reading" information, it senses the polarity of the domains in a given area and converts this information into electric impulses. These, in turn, are converted into binary digits, or "bits." If the head is "writing," the procedure is reversed: the bits are converted into electric impulses which change the magnetic field around the head. The changed magnetic field polarizes the domains according to the value of the bits. Eight "bits" are combined together to make a single "byte."

TRACKS

Each disk is subdivided into a series of concentric rings, called "tracks." Typically, the H17-style disk uses 40 tracks, although it is possible, using non-Heath software, to use special floppy disk drives that provide 80 tracks. The H37-style drives can hold either 40 or 80 tracks, using the standard Heath software. The H47-style drives (i.e., 8-inch size) always use 77 tracks.

SECTORS

Each track is subdivided into areas called "sectors." The H17 disks have 11 holes punched into the media to indicate the location of each of the 10 sectors allowed per track. The eleventh hole is a marker hole. The H37 and H47 can control the number of sectors in their formatting software; hence they are referred to as "soft-sectored" disks. Typically, an H37 disk holds 10 sectors in "single density" mode, or 16 sectors in "double density" mode. With non-Heath software, it is possible to INIT H37 disks with 18 sectors per track. The H47 disks can hold 13 or 26 sectors per track, depending upon whether they are formatted in "single density" or "double density." Each sector holds 256 bytes of data.

SINGLE OR DOUBLE-SIDED DISK DRIVES

Standard H17 drives record only on a single side of the disk medium. With non-Heath software, it is possible for the H17 controller to handle a double-sided disk drive, and record on both sides of the disk.

THE DISK (Cont)

The standard H37 software can control either single or double-sided floppy disk drives. The H47 uses special 8-inch disk drives that are always double-sided. You have the option of using less expensive single-sided disks for recording on only one side if you wish.

DISK CAPACITY

Because of the enormous number of possible combinations of disk controllers, number of sides, number of tracks, and electronic recording densities, it is impractical to list all the different disk capacities. They range from 400 sectors (H17, single-side, 40-track) to 4004 sectors (H47, double-side, double density).

GROUPS

HDOS internally combines nearby sectors of data together into what are called "groups." Because of the intrinsic structure of HDOS, it is possible to keep track of no more than 255 "groups" of data sectors per disk. A 400-sector disk will be organized into 200 groups, with 2 sectors per group. A 2560-sector disk (H37, double-sided, double-density 80 track, 16 sectors per track) will be organized into 213 groups of 12 sectors each. A 4004-sector H47 disk (double-sided, double-density) will be organized into 250 groups of 16 sectors each.

DISK FILE STORAGE

Because of this internal grouping of sectors, the minimum amount of space HDOS will use when recording a file is a single group. Using the examples shown of standard H17, H37, and H47 disk sizes, 10 files of one sector each will use the following amount of actual disk space: H17 - 20 sectors; H37 - 120 sectors; H47 - 160 sectors. However, to store 10 files of 16 sectors each would require: H17 - 160 sectors; H37 - 240 sectors; H47 - 160 sectors.

****

ACCESS TIME -- FINDING THE RIGHT SECTOR

SEEKING

The process by which HDOS finds a given track is called "seeking." It takes a small amount of time for the head to move from track-to-track as it seeks the correct one. This is called "track step-time." The 5-1/4 inch drive used in the H89, H17, and H77 has a guaranteed step time of 30 milliseconds (0.03 seconds), although many 5 1/4 inch floppy disk drives are actually capable of much faster operation. You may determine the optimum step-time of your 5-1/4 inch drives during the TEST17 procedure. (NOTE: TEST17 is only available under HDOS Version 2.0.) The guaranteed step-time for the 8-inch drive(s) in the H47 is 3 milliseconds (0.003 seconds).

ACCESS TIME - FINDING THE RIGHT SECTOR (Cont)

ROTATIONAL LATENCY

Another factor that determines system speed is called "Rotational Latency." When you type in a command and a filename, the system first locates the track upon which the file is located. Since the read-write head is fixed, the system may have to wait for the correct sector to rotate underneath the head after having selected the correct track. Rotational latency is thus defined as the amount of time that it takes for the spindle in the disk drive unit to rotate the desired sector into a position where the read/write head can read data from the file you have specified. The amount of rotational latency depends upon the relative positions of sector and read-write head. If there is a great amount of distance between the head and sector, the rotational latency of the drive may be as great as 200 milliseconds for 5-1/4 inch drives, and as great as 166.6 milliseconds for the 8-inch drives. The rotational latency period of a 5-1/4 inch disk averages 100 milliseconds: the average rotational latency for 8-inch drives is 83.3 milliseconds.

ACCESS TIME

The access time, or period of time it takes to locate a sector, depends upon the track step-time of the disk drive as well as the relative positions of sector and head. This access time may be as great as 216 milliseconds in the case of the 8-inch drives. The average access time for 5-1/4 inch drives is 225 milliseconds; the average access time for 8-inch drives is 91 milliseconds. These averages are based upon calculations which presuppose that during the average disk access, one-third of the tracks will have to be skipped over. If many sectors close to one another are accessed, the average access time will be much lower, i.e., faster.

READ/WRITE RATE

As soon as HDOS finds the correct sector, it reads or writes the data serially one bit at a time. This serial transfer of data takes place at a rate of 16,000 bytes per second with H17 5-1/4 inch drives, at a rate of 30,000 bytes per second with H37 drives, and a rate of 62,745 bytes per second with 8-inch drives. The binary data is transferred between the disk drive and the disk drive controller. The controller converts the serial data into parallel format. That is, the controller reformats the single-bit data so that, instead of being transferred one bit at a time, the data is transferred 8 bits (one byte) at a time. The controller then sends the data to the CPU (central processing unit) by means of an input/output port.

THE SOFTWARE SYSTEM

THE HDOS OPERATING SYSTEM

An operating system such as HDOS is a logically organized set of programs which perform various tasks, such as communicating with peripherals and finding data on the disk. All the programs of HDOS are accessible to you by means of operating system "commands." The exact command syntax is discussed in the "General Operations" Chapter of this manual.

THE HEART OF HDOS

Two vital programs within the operating system control the other portions of HDOS. One of these programs, HDOS30.SYS, is called the "nucleus;" the other, SYSCMD.SYS, is called the "command processor."

THE NUCLEUS

The "nucleus" is the heart of HDOS. It is a collection of control routines which the command processor and utility programs, such as PIP and BASIC, call upon to execute your commands. The machine-language routines in the nucleus control such operations as allocating disk space and memory and transferring data from memory to disk and from disk to memory. The nucleus resides in memory whenever the operating system is running.

THE COMMAND PROCESSOR

Working in conjunction with the nucleus is the "command processor," which processes all input from the keyboard when the HDOS system prompt is displayed. When you enter data at the terminal keyboard, the data go directly to the nucleus, where it is temporarily stored. The command processor continually checks the data in the nucleus to see if a carriage return has been entered. When the command processor "sees" that the data in the nucleus have been terminated by a carriage return, it takes whatever data precedes the carriage return as a command. If you enter a valid command, the command processor then either executes the command itself (an internal command) or passes control to a sub-program which is capable of executing the command (an external command). If the command involves some form of disk I/O, the processor or subprogram utilizes the control routines in the nucleus. After an internal command has been executed, control returns to the command processor, which then awaits the next command. When an external command is executed, however, that program takes over control of the keyboard and communicates with the nucleus. It is not until the program "exits to HDOS" that the system processor regains control.

****

FILES

All information on a disk is stored as bytes within a 256-byte sector. If the storage of a program or some other type of data requires more than 256 bytes, which is often the case, some means of quickly locating all the sectors which contain that data is needed. Thus, to facilitate location and retrieval of all the component sectors of a program or mass of data, HDOS organizes sectors into data structures known as "files." Files may be composed of many sectors.

THE HDOS "LIBRARIAN"

With regard to file management, the operating system is like a librarian, who must know where to find all the books in the library. The operating system must: find space on the disk for new files; give each file a new name so that it can be easily located; be able to copy from one file to another; and be able to rename, update, and delete files. Above all, the operating system must be able to communicate with you and execute your commands quickly and effectively.

GROUPS OR CLUSTERS

HDOS allocates sectors to files in groups called "clusters" or "groups." The name "clusters" was coined by programmers who discovered the inner workings of HDOS by disassembling the machine code. The 'official' name as used in the original HDOS source code is "group."

Each group or cluster is composed of two contiguous sectors in the case of the H17 5-1/4 inch disks, and up to sixteen contiguous sectors in the case of H47 8-inch disks, when using double-sided, double-density recording. inch disks. Clusters on a single-sided, single-density 8-inch disk are composed of 4 sectors. Clusters on a double-sided, double-density 8- inch disk are composed of 16 sectors.

CLUSTER FACTOR

The number of sectors HDOS assigns to a cluster on a particular type of disk is called the "cluster factor." In the case of a 5-1/4 inch disk, the cluster factor is 2. If the cluster factor were 2, HDOS would allocate two sectors to a file which required only one sector to store. The system "remembers" whether or not all the sectors in a cluster are used. If HDOS needs to extend a file, it uses one of the unused sectors in the cluster it has assigned to that file. By keeping the component sectors of the file close together in this way, HDOS minimizes seek-time when reading the disk.

THE BUFFER

Since the 256-byte sector is the basic allocation unit of the disk, all data transfers to and from the disk must involve some multiple of 256 bytes. Many of the HDOS system programs such as BASIC, ASM, and EDIT use an area of RAM called the "buffer" to store data until some multiple of 256 bytes is accumulated. When the buffer has become full, whether by "padding" the data is to be transferred or by simply waiting for more information to be added to the buffer, the data are written to the disk. A buffer of sufficient size must be set aside by each program which reads files. BASIC automatically sets aside any necessary buffer space.

FILES (Cont)

THE DIRECTORY

In order to keep track of all the files in the system, HDOS uses a listing known as the "directory." The directory contains the name, location, size, and creation date of every file in the system. In order to access a given file, HDOS first looks up its name in the directory. It then uses information listed along with the filename to locate the sectors which comprise the file. The directory is also an HDOS file called "Direct.Sys." HDOS cannot use this file to look up the location of a file, because it cannot read "Direct.Sys" until it can look at the data in "Direct.Sys" that tells where "Direct.Sys" is located! HDOS solves this paradox by recording the location of "Direct.Sys" on physical track zero of every disk.

MANIPULATING FILES

Files can be manipulated at the HDOS level, at the level of subprograms, such as ONECOPY, as well as within BASIC and assembly language programs. Refer to the appropriate section of this manual for the proper command syntax.

****

MEMORY MAP

MEMORY MANAGEMENT

An operating system requires a memory management capability in order to function with other programs. When you issue a command to run a program, HDOS locates the file that bears the name of the program and then determines whether or not it will fit into the available memory. If it will fit, HDOS reads the file into a memory area that begins at the program's starting address. After the loading process is completed, HDOS jumps to the starting address of the program and execution begins.

Normally, HDOS will give a program only the minimum amount of memory that it actually needs. However, the program may request HDOS to release additonal memory as needed. Example: Microsoft BASIC. This program requires all available memory. For this reason prior to calling Microsoft BASIC, one first loads the printer device driver. Refer to Appendix 8-A, "Memory Layouts - Memory Map" for details of the HDOS 3.02 memory map.

****

CONTROLLING PERIPHERALS

HDOS will also manage all the peripheral devices on your computer system. This gives you the capability to type in commands from the console terminal and have the operating system perform some input or output function without your further intervention. You can also include I/O commands in your programs without having to write detailed instructions for controlling the peripherals.

CONTROLLING PERIPHERALS (Cont)

HDOS uses programs called "device drivers" to control peripherals. Each device driver is written with the characteristics of a particular device in mind. Its speed, line length, and "handshaking" capability or the capability of a peripheral to instruct the CPU not to send any more data until the peripheral is ready, are all examples of these characteristics. Device drivers are stored on the disk as files, with each device driver corresponding to one peripheral device. For example, to control the disk drives of your system, HDOS uses two device drivers called SY.DVD and DK.DVD. The SY.DVD disk driver controls all disk drives which have been configured as primary boot drives, while the DK.DVD driver controls all disk drives which have been hardware configured as secondary boot drives.

After the device drivers are written as files, one for each peripheral, the drivers become a part of the operating system. Since device drivers are a part of HDOS, it is not necessary to include the drivers in each program which makes use of the peripherals. You can also make a program communicate with many different peripherals, without rewriting it, by defining the appropriate device drivers. This ability to refer to a peripheral without keying instructions into a program is called "device independence."

The effect of HDOS's device independence is that each peripheral is given a symbolic name, as if it were a file. For instance, the system console terminal is referred to as "TT:," and "SY0:" identifies the system disk drive. The colon (e.g. :) at the end of the symbolic name identifies it as a device name rather than a filename.

In addition, it is possible to have more than one printer driver on one disk. For example, one could add the driver called "UD.DVD" to complement the existing "LP.DVD." Other device drivers could be added as well. If new peripheral devices are added to your computer system, you might, for example, add a device driver called "MD.DVD" to control a modem. A modem is a hardware item that connects a serial port to a telephone line. Data sent to "MD.DVD" would then go out the telephone line in the same way that data sent to device LP: go to the printer. Similarly, one could add a device driver called "CK.DVD." This driver would contain instructions that would enable HDOS to communicate with a real-time clock installed in the computer. In addition, if you have a Spooldisk mounted in your computer, the device driver "SS.DVD" would enable you to communicate with that versatile RAM card. There are numerous device drivers available.

APPENDIX 8-A: MEMORY LAYOUTS - MEMORY MAP

The following is a complete memory map of the HDOS 3.02 environment. It is the same for an H/Z 90, H/Z89, or an H8 after the BOOT process is complete and you are running.

Everything between the System Resident FWA (317.116 in this example) and the Channel Table FWA (364.170 in this example) is dependent upon the device drivers and tasks you have in your computer setup. That is, which ones you have LOADed and/or STARTed.

In this example, seven device drivers and two tasks are resident on the system disk. The Device Driver Table is dynamically allocated at BOOT time, and is thus 99 bytes long. It consists of 7 entries of 14 bytes each, and a trailing zero byte. Following this table are the Unit Specific Tables. Each one is a multiple of 8 bytes; one entry for each available unit on each device. Next is the Channel table. It normally consists of 7 entries of 42 bytes each, for a total of 294 bytes. The 7 normal channels are designated as -1 through 5.

For this sample Memory Map, all 7 devices and started and two tasks started. Under normal operation, this probably would not be the case. Also, with the use of the UNLOAD command, much of this allocated memory could be recovered and reused for running application programs or other purposes.

The GRT buffer allocation is an interesting function, and worth a few comments. The SY: device's GRT buffers are allocated first at the top of memory. One memory page, 256 bytes, is reserved for each available unit on the SY: device. In our example, my SY: device had only one unit. As any additional directory devices are loaded into memory, their GRT buffers are allocated in a similar manner, with one exception. A GRT buffer must reside on an exact page boundary. That is, the low order byte of its starting address must be zero. This is necessary because of the way HDOS (older versions and newer versions) navigates through Group Reservation Tables (GRT). The GRT table is also known as the File Allocation Tables (FAT) in the MS-DOS world. To achieve this requirement, HDOS abandons any memory between the current System Resident FWA and the next available page address below it. This step is illustrated in the example at address 363.000, where we have lost 56 bytes and at address 355.000 where we have lost 45 bytes. Since HDOS has no way of controlling how many bytes it will need to abandon, you could lose as many as 255 bytes each time, or none at all.

Anything residing between the System Resident FWA and the HDOS Resident FWA is ONLY temporary, and will be abandoned the next time SYSCMD is re-entered, unless it is a directory device, and there are mounted units on the device. The exception to this rule is a device driver that locks itself in memory, such as the H17 driver. Since it re-routes an interrupt vector, it MUST stay in memory until it is physically unloaded, so that it can restore the original vector and, thus, be "polite" to HDOS.

APPENDIX 8-A: MEMORY LAYOUTS - MEMORY MAP (Cont)

HDOS 3.02 MEMORY MAP

Description	Address	(Addr)*	Bytes	Comment
System High Memory	377.377	(FFFF)	---	Last real address in memory
SY0: GRT Buffer	377.000	(FF00)	256	
HDOS Scratch Area	375.000	(FD00)	512	Directory blocks, etc.
TT: Driver FWA	370.114	(F84C)	1204	H19
SY: Driver FWA	366.121	(F651)	507	Super89 Ram Drive
Device Table FWA	365.356	(F5EE)	99	14 bytes per device plus trailing zero
TT: Unit Specific Data	365.346	(F5E6)	8	8 bytes/unit
SY: Unit Specific Data	365.336	(F5DE)	8	8 bytes/unit
DK: Unit Specific Data	365.316	(F5CE)	16	8 bytes/unit
DY: Unit Specific Data	365.266	(F5B6)	24	8 bytes/unit
ND: Unit Specific Data	365.256	(F5AE)	8	8 bytes/unit
LP: Unit Specific Data	365.246	(F5A6)	8	8 bytes/unit
SP: Unit Specific Data	365.236	(F59E)	8	8 bytes/unit
Channel Table FWA	364.170	(F478)	294	42 bytes/channel
S89CLK Task FWA	363.105	(F345)	307	Super89 Real Time CLK
Task Table FWA	363.070	(F338)	13	Task Table Entry
Memory Hole	363.000	(F300)	56	<<< Wasted Space >>>
DK0: GRT Buffer	362.000	(F200)	256	
DK1: GRT Buffer	361.000	(F100)	256	
DK: Driver FWA	355.057	(ED2F)	977	H47
Pointer to UNLOAD DK:	355.055	(ED2D)	2	
Memory Hole	355.000	(ED00)	45	<<< Wasted Space >>>
DY0: GRT Buffer	354.000	(EC00)	256	
DY1: GRT Buffer	353.000	(EB00)	256	
DY2: GRT Buffer	352.000	(EA00)	256	
DY: Driver FWA	342.230	(E298)	1896	H17
Pointer to UNLOAD DY:	342.226	(E296)	2	
ND: Driver Flag	342.140	(E260)	54	Null Device
Pointer to UNLOAD ND:	342.136	(E25E)	2	
LP: Driver FWA	340.076	(E03E)	544	H25
Pointer to UNLOAD LP:	340.074	(E03C)	2	
SP: Driver FWA	324.205	(D485)	2999	Screen Dump Utility
Pointer to UNLOAD SP:	324.203	(D483)	2	
TDU Task FWA	317.133	(CF5B)	1320	Terminal DeBug Utility Task
Task Table Entry	317.116	(CF4E)	13	Task Table Entry
HDOS Resident FWA	317.116	(CF4E)	---	
System Resident FWA	317.116	(CF4E)	---	

APPENDIX 8-A: MEMORY LAYOUTS - MEMORY MAP (Cont)

HDOS 3.02 MEMORY MAP

Description	Address	(Addr)*	Bytes	Comment
User Memory FWA	042.200	(2280)	44238	Available for User Programming
Top of STACK	042.200	(2280)	----	
Bottom of STACK	041.150	(2168)	280	
STACK Overflow	041.146	(2166)	2	
HDOS Work Cells	040.100	(2040)	294	
Monitor Work Cells	040.000	(2000)	64	Some are still in use
Free Space	037.371	(1FF9)	7	
Type Ahead Buffer FWA	037.224	(1F94)	101	
Editor Buffer FWA	037.057	(1F2F)	101	
Prompt Buffer FWA	036.312	(1ECA)	101	
Path Buffer FWA	036.145	(1E65)	101	
Substitute Buffer FWA	036.000	(1E00)	101	
Batch Buffer FWA	035.000	(1D00)	256	
System Label FWA	034.000	(1C00)	256	
Free Space	033.257	(1BAF)	81	Reserved for HDOS 3.1
H17 ROM Code	033.145	(1B65)	74	DO NOT MESS WITH THIS CODE !!!
Free Space	032.223	(1A93)	210	Reserved for HDOS 3.1
H17 ROM Code	031.275	(19BD)	214	DO NOT MESS WITH THIS CODE !!!
Free Space	031.222	(1992)	43	Reserved for HDOS 3.1
H17 ROM Subroutines	030.060	(1830)	354	$COMP thru $ZERO
Free Space	030.003	(1803)	45	Reserved for HDOS 3.1
H17 ROM FWA	030.000	(1800)	3	Jump to Fatal System Error!
Free Space	026.157	(166F)	401	Used for PRELOAD.ABS
HDOS System Code	000.200	(0080)	5615	Operating System Code
Base Page	000.000	(0000)	128	Vectors, Data, and Pointers

NOTE:
The term (Addr) indicates the address of an item listed in the left column.

HDOS SOFTWARE REFERENCE MANUAL

HDOS DISK OPERATING SYSTEM

VERSION 3.0

CHAPTER 9

CONSOLE DEBUGGER

DEBUG

HEATH DISK OPERATING SYSTEM

SOFTWARE REFERENCE MANUAL

VERSION 3.0

This manual is indicative of further improvements and provides for the latest revision, HDOS 3.0 and HDOS 3.02. Revision 3.0 is detailed in chapters 1, 2, and 3, while chapters 4 ,5, 6, 7, 8, 13 and 14, are related to revision 3.02. Chapters 9 through 12, with minor improvements, are essentially picked up from the original HDOS 2.0 manual. Indeed, HDOS is still alive and well!

SPECIAL DISCLAIMER: The Heath Company cannot provide consultation on either the HDOS Operating System or user-developed or modified versions of Heath software products designed to operate under the HDOS Operating System. Do not refer to Heath for questions.

Instead, you are invited to direct any questions concerning the Heath Disk Operating System (HDOS) to Mr. Kirk L. Thompson, Editor "Staunch 89/8" Newsletter, P.O. Box 548, #6 West Branch Mobile Home Village, West Branch, IA 52358.

TABLE OF CONTENTS

INTRODUCTION

The Heath Console Debugger, DBUG, allows you to debug machine language programs from a console terminal. DBUG occupies the lowest 4K of user program RAM (random access memory) location, and can be manipulated via DBUG. Note that your program MUST be ORGed above the end of DBUG.

Refer to Appendix 8-A, "Memory Layout - Memory Map" for details.

DBUG 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 compatible register.
* Alter the contents of any 8080 compatible 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 a device.
* Dump user programs to a device.

A number of features were designed into DBUG 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. DBUG also contains a single-instruction facility that permits you to execute your program a single line of 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 control to DBUG for analysis and/or modification.

DBUG makes use of the console facilities of HDOS: therefore, the HDOS console control conventions, CTRL-S, CTRL-Q, CTRL-O, CTRL-P, etc, also apply to DBUG. DBUG does not respond to CTRL-Cs. CTRL-A is used to return to DBUG command mode. This is done so the program being debugged can make use of CTRL-B and CTRL-C.

When it is accepting commands from the console keyboard, DBUG uses the "command completion" technique. As each character is entered, it is checked against a list of all possible commands. If the character could not be a part of any valid command, DBUG will refuse it by echoing an ASCII BELL character. In addition, if DBUG determines that there is only one choice for the next character in the command, DBUG will type that character for you. Thus, if you strike the L key, DBUG knows that all commands that start with L start with the word LOAD, and will print the entire word, LOAD on the terminal.

NOTE: The symbol [^] indicates a space. Spaces are critical in HDOS.

COMPUTER/USER DIALOGUE CONVENTIONS

In order to more clearly show the dialogue between the computer and the user, statements made by the computer are set off by quotation marks, and the responses made by the user are set off by apostrophe marks. However, if the statement is an example of a command or similar phrase, showing one or the other of the dialogs, this convention will be ignored, since the meaning will be clear. Anytime you see the symbol [^] it means to make a space unless otherwise indicated.

****

DISK LOADING/DUMPING

DBUG offers two commands for program loading and dumping (or saving). With these commands, described below, you can load or dump an absolute binary program.

LOAD Filename.Ext

The LOAD command causes the contents of the file specified by <fspec> to be loaded into memory. The file must be in absolute binary format (the format generated by the HDOS assembler). Note that the absolute binary file contains information to tell HDOS where the file must be loaded. DBUG will not allow you to load a program over the DBUG code or over the HDOS operating system. The entry point address for the loaded program will be entered into the program counter (Pc register) automatically. For example:

```
":B:"'LOAD^SY1:TEST.ABS<RTN>'
```

DUMP^<fspec>^saddr-eadder

The DUMP command causes the contents of memory from saddr to and including eadder to be written to the file<fspec> in absolute binary

format. The contents of the Pc register at the time of the DUMP are stored in the file as the program's entry point. You can use this feature to save a patched binary program without reassembling it. For example, to save the demo program used on Page 3-13, type:

```
":B:"'DUMP^TEST^70000-70026<RTN>'
```

****

EXECUTING DBUG

DBUG is called via the HDOS Operating System as follows:

```
">"'DBUG<RTN>'

  "HDOS DBUG # 102.00.00"

  ":B:"
```

Typing a CTRL-D will exit the program.

Note that the version number may not be the same as yours, but a number will be shown.

****

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>
```

[For example: FA^101102-101105^TAP SPACE BAR ONE TIME]

Format control specifies that the 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 keyboard.

FORMAT CONTROL

The format control consists of EITHER zero, one, or 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 as follows:

MEMORY COMMANDS (Cont)

FORMAT CONTROL (Cont)

FORMAT CONTROL	DESCRIPTION
Space Space	Display/alter as octal integers, byte format.
F	Display/alter as octal integers, word format.
A	Display/alter as ASCII characters, byte format.
FA	Display/alter as ASCII characters, word format.
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

MEMORY COMMANDS (CONT)

FORMAT CONTROL (Cont)

DECIMAL FORMAT [D] (Cont)

integers. A blank space 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.

RANGE FORM	DESCRIPTION
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.

MEMORY COMMANDS (Cont)

RANGE (Cont)

For example, to display memory locations 000 043 through 000 047, DBUG simply requires the user to type 43-47 followed by a blank space, which is generated by using the keyboard space bar. For example:

```
":B:"'43-47^'"100 112 107 114 100"
":B:"'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 octal byte format. 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. The form of the error message is:

```
"LWA<FWA"
```

For example:

```
":B:"'47-43^"LWA<FWA"
":B:"
```

NOTE: If you attempt to enter a numerical address which does not fit the offset (split) octal format, DBUG rejects the improper entry and sounds the console terminal bell. For example, the number 067777 does not fit the offset octal format. Therefore, DBUG 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 space following the format and range fields. DBUG immediately executes the command. In the following examples, the contents of a number of locations, 002 143 to 002 163 in the Monitor ROM, are displayed in octal byte format, in octal word format, in decimal byte format, and in decimal word format. NOTE: When all the bytes or words in the specified range cannot be displayed on the line, a new line is started. DBUG supplied the starting address of the new line.

```
":B:"'2143-2163^"343 353 041 011 040 256 136 167 056 033 172 206 276
      302 002162  160 002"
":B:"'F2143-2163^'"325343 041353 040011 136256 172033 276206 160302
      303002"
":B:"'D2143-2163^'"227 235 033 009 032 174 094 119 046 027 122 134 190
      194 002162  112 002"
":B:"'FD2143-2163^'"54755 08683 08201 24238 11895 31259 48774 28866
      49922"
```

Note that you may type CTRL-A to short a memory display.

For example:

```
":B:"'30000-60000^'"303 014 037 041 300 377 071 353 041 100 040 166 042^A
      042"'CTRL-A'
":B:"
**********************************************************************
```

ALTERING MEMORY - DECIMAL OR OCTAL

To alter memory in decimal or octal formats, type an equal sign [=] after the format control and range fields. DBUG will then type the value of the first byte, or double byte if an F was used in the format control. Then follow this with a slash [/]. 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 <RTN>.

If you type a space, DBUG offers the next byte [if there is one in the range] for alteration. If you type a <RTN>, DBUG returns you 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 current octal value may be different from that shown in the example. 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.

ALTERING MEMORY - DECIMAL OR OCTAL (Cont)

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=324/101<RTN>'
":B:"'60001=030/102<RTN>'
":B:"'60002=353/103<RTN>'
":B:"'60003/23=341/104^330/105^203/106^137/107^076/110^000/111^212/
      112^127/113^'
"060013  322/114^365/115^057/116^311/117^315/120^072/121^030/122^345/
      123^"
"060023  365/124^345/125^021/126^012/127^000/130^315/131^106/132^"
":B:"

":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
      117 120"
"060020  121 122 123 124 125 126 127 130 131 132"
":B:"
```

The DELETE key is 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 zeroes to cause the bad digit to be eliminated. For example, if byte 70,000 is to be set to 123 and a mistype of 125 occurs, it may be correctly entered as:

```
":B:"'70000=111/1250123<RTN>'
":B:"'70000=123/<RTN>'
```

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 carriage returns are considered to be ASCII character values. To exit the command prematurely, use the ESCape or CTRL-A key to avoid altering a location.

****

ALTERING MEMORY - DECIMAL OR OCTAL

```
":B:"'A70000=/A'
":B:"'A70001=/B'
":B:"'A70002=/C'
":B:"'A70003-70031= /D /E /F /G /H /IT /JT/ /K /L /M /N7 /O /P /Q /R /S
      /T /U /V'
"070026   /W /X /Y /Z"
":B:"'A70000-70031^'"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:"'A70000/26^'"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:"
```

****

REGISTER COMMANDS

DBUG 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><CTRL-R>

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

```
":B:"'<CTRL-R>'
"A=000 B=000 C=001 D=000 E=004 H=070 L=100 F=203 P=070005 M=070100
      S=042200"
":B:"

":B:D"'<CTRL-R>'
"A=000 B=000 C=001 D=000 E=004 H=056 L=064 F=131 P=14341 M=14400
      S=00832"
":B:"

":B:"'A<CTRL-R>'
"A= B= C= D= E= H=8 L=@ F= P=8 M=8@ S="
":B:"
```

REGISTER COMMANDS (Cont)

DISPLAYING ALL REGISTERS (Cont)

A Control-R [CTRL-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 on 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> =
```

DBUG will then display the previous contents of the register in the specified format [i.e. 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

REGISTER COMMANDS (Cont)

ALTERING REGISTER CONTENTS (Cont)

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 CTRL-A to abort the change. The following examples demonstrate the altering of register contents.

```
":B:"'REGA=102/103<RTN>"   Change contents of A from 102 (subscript 8)
                           (ASCII B) to 103 (subscript 8) (ASCII C).
":B:"'DREGA=067/066<RTN>'  Change contents of A from 67 (subscript 10)
                           (ASCII C) to 66 (subscript 10) (ASCII B).
":B:"'AREGA=B/C'           Change contents of A from ASCII B to ASCII
":B:"                      C.

":B:"'REGA=103/<RTN>'      A carriage return aborts the change.
":B:"'AREGA=C/<CTRL-A>'    A CTRL-A aborts the change.
":B:"'AREGA^=C'            The location is unaltered.
":B:"
```

NOTE: The last three examples illustrate aborting the change or leaving the location unaltered.

****

EXECUTION CONTROL

One of the primary functions of DBUG 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 DBUG control to examine register and memory contents. You may also stop your program execution at any time by typing CTRL-A, which will cause control to return to DBUG. 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, DBUG starts stepping at the current PC-register address. When the instructions are completed, DBUG types the PC-register value and returns to the command mode. If an address is specified, DBUG starts stepping at the

EXECUTION CONTROL (Cont)

SINGLE-STEPPING (Cont)

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 070 100 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 DBUG.

```
ADDRESS                LABEL INSTRUCTION       COMMENT
-------                ----------------        -------

070.000                START  ORG  070000A
070.000  041 100 070   L1     LXI  H,070100A   POINT HL TO 070100
070.003  066 000              MVI  M,000       LOAD MEMORY WITH ZERO
070.005  003           L2     INX  B           INCREMENT BC PAIR
070.006  170                 MOV  A,B         LOAD A WITH B
070.007  376 027              CPI  027Q        IS B 027 OCTAL?
070.011  302 005 070          JNZ  L2          JUMP BACK IF NOT
070.014  064                 INR  M           INCREMENT MEMORY
070.015  176                 MOV  A,M         LOAD A WITH MEMORY
070.016  376 377              CPI  377Q        IS MEMORY 377 OCTAL?
070.020  006 000              MVI  B,000       LOAD B WITH ZERO
070.022  302 005 070          JNZ  L2          JUMP IF NOT ZERO
070.025  327                 RST  2
070.026  000                 END  START
```

NOTE: The RST2 instruction is used to return this program to DBUG. When the CPU encounters an RST2 instruction, it returns to DBUG.

For example, to load the above program using DBUG:

```
":B:"'70000-70025=101/"041^102/100^103/070^104/066^105/000^106/003^
      107/170^110/376^'
"070021  122/000^123/302^124/005^125/070^126/327^"
":B:"

":B:"'REGB=302/000<RTN>'
":B:"'REGC=110/000<RTN>'
":B:"'STEP 70000/6<RTN>'
"-P=070005-"
":B:"
```

NOTE: DBUG returns the value of the PC once the first six steps are executed.

EXECUTION CONTROL (Cont)

BREAKPOINTING

DBUG 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 debugging 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
```

DBUG allows up to 8 breakpoints. They are entered in the breakpoint table within DBUG, replacing any previously defined breakpoints at those addresses. No more than eight 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 DBUG. NOTE: If CNT is not specified, the value 1 is assumed.

EXECUTION CONTROL (Cont)

BREAKPOINTING (Cont)

For example, the program of the previous example is run with breakpoints.

```
":B:"'70100=000/<RTN>'
":B:"'BKPT 70015/6<RTN>'

":B:"'GO 70000<RTN>'

"-P=070015-"

":B:"'70100=006/<RTN>'
":B:"
```

NOTE: 070 100 is incremented by 6.

```
":B:"'70100=006/000<RTN>'
":B:"'BKPT 70015/6,70014/10,70022/30<RTN>'

":B:"'GO 70000<RTN>'

"-P=070015-"

":B:"'GO<RTN>'


"-P=070014-"

":B:"'GO<RTN>'

"-P=070022-"

":B:"
```

DISPLAYING BREAKPOINTS

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

```
BKPT DSPLY
```

DBUG 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

EXECUTION CONTROL (Cont)

DISPLAYING BREAKPOINTS (Cont)

breakpoint count is exhausted, it causes control to return to DBUG. The exhausted breakpoint is removed from the breakpoint table; non-exhausted breakpoints remain.

For example:

```
    ":B:"'70100=036/000<RTN>'
    ":B:"'BKPT 70015/6,70014/10,70022/30<RTN>'

    ":B:"'BKPT DSPLY 070015/006 070014/010 070022/030'
    ":B:"'GO 70000<RTN>'

    "-P=070015-"

    ":B:"'BKPT DSPLY 070014/004 070022/025'
    ":B:"'GO<RTN>'

    "-P=070014-"

    ":B:"'BKPT DSPLY 070022/021'
    ":B:"'GO<RTN>'

    "-P=070022-"

    ":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 clear all breakpoints from the breakpoint table, use the breakpoint clear command:

```
    CLEAR ALL
```

EXECUTION CONTROL (Cont)

CLEARING ALL BREAKPOINTS (Cont)

For example:

```
    ":B:"'BKPT 55012/10,55014/15,55020/20,55022/200<RTN>'
    ":B:BKPT DSPLY 055012/010 055014/015 055020/020 055022/200"
    ":B:"'CLEAR 55014,55022<RTN>'
    ":B:"'CLEAR ALL<RTN>'
    ":B:BKPT DSPLY"
    ":B:"
```

****

GO COMMAND

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 [SADDER]
```

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

```
    ":B:"'GO<RTN>"
    "-P=070025-"
    ":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 070 and to execute to byte 070 015, type the command:

```
    ":B:"'EXEC 70000-70015<RTN>'
    "-P=070015-"
    ":B:"
```

EXEC (Cont)

```
    ":B:"'GO 70210<RTN>'
    'CTRL-A'
    "-P=072121-"
**************************************************************************
```

CTRL-A

When you are executing your program via EXEC or GO, you may return to DBUG by typing CTRL-A. This is useful when you fail to set a breakpoint, or fail to reach the one that you have set. For example:

```
    ":B:"'GO 70210<RTN>'
    'CTRL-A'
    "-P=072121-"
    ":B:"
**************************************************************************
```

CTRL-D

When you are finished with the DBUG program, you can return to HDOS by typing CTRL-D. The program will respond with," ARE YOU SURE?" To exit, you must respond with a "Y." For example:

```
    ":B:"'CTRL-D'
    "ARE YOU SURE?" 'Y'
**************************************************************************
```

COMMAND COMPLETION

When DBUG 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, DBUG supplies and prints this character for the user.

```
    ":B:"'CTRL-D'
    "ARE YOU SURE?" 'Y'
    ">"
**************************************************************************
```

APPENDIX 9-A: - DEBUG ERROR MESSAGES

The following error messages are generated by DBUG. In addition to these errors, other errors may be detected by the HDOS Operating System itself. These errors are discussed in detail in Chapter Three, System Optimization, Appendix 3-A:, Error Messages, page 3-33.

BELL ..... The console terminal's bell is sounded when you type an illegal character for the current command. DBUG sends the ASCII bell code to the terminal instead of echoing the illegal character.

LWA < FWA ..... The second address specified in this command must be the same or larger than the first address specified.

FORMAT ERROR .. The file you have attempted to load is not of the FILE proper type. The LOAD command will only load BINARY files.

ATTEMPT TO .... A file may not be loaded into the same memory locations LOAD OVER DBUG occupied by DBUG.

NO ROOM ..... You have attempted to specify more breakpoints than DBUG has room for. DBUG currently allows a maximum of eight breakpoints to be simultaneously specified.

****

APPENDIX 9-B: - DBUG COMMAND SUMMARY

MEMORY COMMANDS

Memory Display Form:

FORMAT CONTROL range blank

Memory Alter Form:

FORMAT CONTROL range =

FORMAT	RESULT
<null><null>	byte octal
F<null>	word octal
<null>A	byte ASCII
FA	word ASCII

APPENDIX 9-B: DBUG COMMAND SUMMARY (Cont)

Memory Alter Form: (Cont)

FORMAT CONTROL range =

FORMAT	RESULT
<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:

RANGE FORM	DESCRIPTION
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 (i.e. count) memory locations starting at location ADDR. NOTE: cnt is a decimal integer less than 255.
#-ADDR	Range specifies the memory locations starting at the beginning of the previous range and ending at ADDR.
#/cnt	Range specifies cnt (count) memory locations starting at the beginning of the previous range. NOTE: cnt is a decimal integer less than 255.
<NULL>/cnt	Range specifies cnt (count) memory locations starting at the address following the last address of the previous range. NOTE: cnt is a decimal integer less than 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.

APPENDIX 9-B: - DBUG COMMAND SUMMARY (Cont)

REGISTER COMMANDS

All Registers:

```
FORMAT CTRL-R
```

Single Register:

```
REG REG-NAME blank
```

Altering Register:

```
REG REG-NAME=
```

EXECUTION CONTROL

Single-stepping:

```
STEP ADDR/cnt
```

Breakpointing:

```
BKPT ADDR1/cnt1, . . . . . , ADDRn/cntn
```

Breakpoint Display:

```
BKPT DSPLY
```

Clearing Breakpoints:

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

GO:

```
GO(ADDR)     (Starts at PC value if ADDR not specified.)
```

Execute:

```
EXEC SADDR-ADDR1, . . . . ., ADDRn  (Combines GO and BKPT.)
```

PROGRAM LOADING AND DUMPING

```
LOAD<Fspec>
DUMP<Fspec>,saddr-eadder
```

****

INDEX

HDOS SOFTWARE REFERENCE MANUAL

HDOS DISK OPERATING SYSTEM

VERSION 3.0

CHAPTER 10

HEATH TEXT EDITOR

EDIT

HEATH DISK OPERATING SYSTEM

SOFTWARE REFERENCE MANUAL

VERSION 3.0

 Through the years it continued to be improved by successive revisions which included 1.5, 1.6, and finally 2.0. It was entered into public domain on 19 July 1989 per letter by Jim Buszkiewicz, Managing Editor, Heath Users' Group, P.O. Box 217, Benton Harbor, MI 49022-0217 (616)982-3463. A copy of this letter is available for public inspection.

This manual is indicative of further improvements and provides for the latest revision, HDOS 3.0 and HDOS 3.02. Revision 3.0 is detailed in chapters 1, 2, and 3, while chapters 4 ,5, 6, 7, 8, 13 and 14, are related to revision 3.02. Chapters 9 through 12, with minor improvements, are essentially picked up from the original HDOS 2.0 manual. Indeed, HDOS is still alive and well!

SPECIAL DISCLAIMER: The Heath Company cannot provide consultation on either the HDOS Operating System or user-developed or modified versions of Heath software products designed to operate under the HDOS Operating System. Do not refer to Heath for questions.

Instead, you are invited to direct any questions concerning the Heath Disk Operating System (HDOS) to Mr. Kirk L. Thompson, Editor "Staunch 89/8" Newsletter, P.O. Box 548, #6 West Branch Mobile Home Village, West Branch, IA 52358.

TABLE OF CONTENTS

INTRODUCTION

The Heath Text Editor (EDIT) converts your system into a very sophisticated typewriter. This typewriter is not only capable of generating text, but also has powerful editing capabilities. With these capabilities, even if you are a poor typist, you can create error-free text, organized as you desire.

EDIT is a very powerful utility program with many uses. You can use it to enter and edit assembly language programs and BASIC programs, as well as to create and edit reports, letters, and manuscripts. Note that EDIT supports the ASCII TAB character to conserve space in the text files.

You can do your editing by command, referencing the desired line, and text is stored in a section of memory called the BUFFER. When the buffer is full, you can transfer the text to a disk file. Additional text can be read in from previously created files, or you can place text into the buffer from the terminal keyboard. EDIT's file handling capabilities allow it to edit large files on a piecemeal basis.

All of the RAM in your system that is not used by HDOS or the Edit program is available in the buffer.

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

* Fifteen commands for text editing versatility
* Terminal control of output and input operations
* Command completion and command error analysis

CHARACTER SET

EDIT supports the entire 96-character ASCII set, including the lower case characters, form feed, and tabs. You may enter text and commands in lower case. With the exception of TAB [CTRL-I] and FORM FEED [CTRL-L], you may not use control characters in the text.

EDIT is called as follows:

```
>EDIT<RTN>                NOTE: The prompt for EDIT is a
 EDIT ISSUE #103.00.00            double dash.
 --
```

****

EDITOR MODES OF OPERATION

Two modes of operation called the "Command Mode" and the "Text Mode" are available in EDIT. 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. The carriage return will be indicated throughout this reference manual by: <RTN>. In actual use, no symbol is displayed on the terminal when the <RTN> key is pressed on the keyboard. The 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. Once a source file has been created,

it can be stored on a file. Later, you can use this file as a source of text to be read into the buffer.

Type CTRL-C when you wish to return from the Text Mode to the Command Mode. This performs two functions. First, it discards the line of text which it was on when the keys were struck. Second, it returns the Text Editor to the Command Mode. To preserve the last line of text, press <RTN> to generate a new line before striking CTRL-C.

****

THE COMMAND STRUCTURE

The basic commands for EDIT have the following form:

[<range>] [<verb>] [<qualifier string>] [<option>] [<parameter>]

RANGE ..... Indicates what lines in the buffer the command affects.

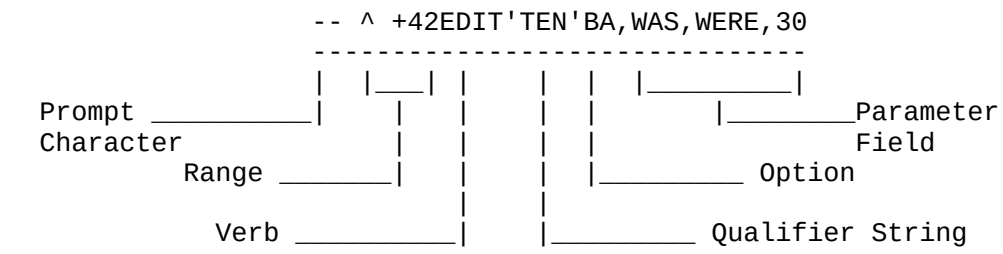
VERB ..... The basic command.

QUALIFIER STRING Limits the command to those lines containing a given string.

OPTION ..... Permits you to view the line before or after (or both).

PARAMETERS ..... Contains specific instructions for some commands.

For example, the command:



is read as follows:

The lines starting with the first line (^)(i.e.,up arrow) and ending with the 43rd line (+42) are to be edited (EDIT). The EDIT command replaces one string with another. The affected lines are limited to those containing the string "TEN," the optional QUALIFIER STRING. The option (BA) indicates that you wish to view the lines before and after editing. And in this particular case, the PARAMETER FIELD specifies the old and new strings, and the number of times the old string is to be replaced by the new string. In this example, the word WAS is to be replaced by the word WERE a maximum of 30 times. NOTE: The EDIT string used in the RANGE EXPRESSION or QUALIFIER STRINGS must be inclosed in a single quote ['']. Each of the various fields of this command is discussed below.

THE COMMAND STRUCTURE (Cont)

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 multiple line expression
- A null
- A blank
- An equal sign

These different Range Expressions are explained as follows:

NOTE: Text must be in the buffer before a Range Expression will be accepted. If the text is not in the buffer, a bell code will sound as you try to complete the Range Expression. To use the following examples, use the INSERT command. See page 4-13 for details.

SINGLE LINE EXPRESSIONS

Any of the following expressions may be used to specify a single line:

THIS SYMBOL	SPECIFIES
^	The first line.
$	The last line.
+n	The "nth" line beyond the current line pointer.
-n	The "nth" line preceding the current line pointer.
+'string'	The first line in the text buffer beyond the current line pointer which contains the designated 'string'.
-'string'	The first line in the text buffer preceding the current line pointer which contains the designated 'string'.

THE COMMAND STRUCTURE (Cont)

SINGLE-LINE EXPRESSIONS (Cont)

EDIT has a "pointer" which is always pointing to some line of text. After you have inserted text or invoked a text file, the EDIT pointer always points to the first line of text.

The effect of the various range commands is to reposition the pointer. Once you have used a range expression to specify a line, EDIT repositions the current pointer at the line indicated by the range expression. For example, suppose the buffer contains:

```
--INSERT<RTN>
THIS IS THE FIRST LINE<RTN>
MARY HAD A LITTLE LAMB<RTN>
THIS IS THE THIRD LINE<RTN>
ITS FLEECE WAS WHITE AS SNOW<RTN>
THIS IS THE FIFTH LINE<RTN>
AND EVERY WHERE THAT MARY WENT<RTN>
THIS IS THE SEVENTH LINE<RTN>
THE LAMB WAS SURE TO GO<RTN>
THIS IS THE LAST LINE<RTN>
USER TYPES CTRL-C
```

If you type

```
--+1PRINT<RTN>
```

EDIT will reposition the pointer from the first line to the second line and print:

```
MARY HAD A LITTLE LAMB
```

Since the current pointer is now positioned at the second line, the command:

```
--+1PRINT
```

indicates to EDIT that it should reposition the pointer to the line that is one line past the current pointer and then print that line:

```
THIS IS THE THIRD LINE
```

Therefore, once you have specified a line, your future range commands should take into account the distance from the current pointer to the line you want to manipulate.

THE COMMAND STRUCTURE (Cont)

MULTIPLE-LINE EXPRESSION

Use a multiple-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 ^, $, +, +n, -n, + '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 multiple-line expression.

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

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

```
--^+2,+3PRINT<RTN>
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH LINE
AND EVERY WHERE THAT MARY WENT
```

```
--^+'FLEECE',+1PRINT<RTN>
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH LINE
```

```
--$-'THAT'+'SURE'PRINT<RTN>
AND EVERY WHERE THAT MARY WENT
THIS IS THE SEVENTH LINE
THE LAMB WAS SURE TO GO
```

```
--$-'THAT',+2PRINT<RTN>
AND EVERY WHERE THAT MARY WENT
THIS IS THE SEVENTH LINE
THE LAMB WAS SURE TO GO
--
```

THE COMMAND STRUCTURE (Cont)

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<RTN>
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 EVERY WHERE 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 keyboard.

THE NULL

The NULL expression (the absence of any range expression) causes EDIT to apply the command to whatever text is indicated by the current line pointer. The position of the current pointer is unchanged.

For example:

```
--^+2,+'FIFTH'PRINT<RTN>
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
THIS IS THE FIFTH LINE
--PRINT<RTN>
THIS IS THE THIRD LINE
--
```

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

THE COMMAND STRUCTURE (Cont)

THE NULL (Cont)

```
--^+2,+1PRINT<RTN>
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
--,+2PRINT<RTN>
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 position of the current pointer, and the "+2" directs EDIT to print two additional lines. Again, the position of the current pointer in this example remains unchanged.

THE EQUAL [=]

The equal [=] expression sets the range of the previous command. For example:

```
--$'MARY',+'GO'PRINT<RTN>
AND EVERYWHERE THAT MARY WENT
THIS IS THE SEVENTH LINE
THE LAMB WAS SURE TO GO
--=PRINT<RTN>
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 EDIT condition. For example, an attempt to PRINT an empty buffer is meaningless, and the PRINT verb will be rejected.

THE COMMAND STRUCTURE (Cont)

THE VERB (Cont)

For example, when you type the P key, the Text Editor knows that only one command begins with a P. Therefore, the "P" is immediately followed by "RINT." However, if you type the "N," the Editor 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 the command verbs and their specific actions and limitations follows in the "Commands" section.

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 inclosed 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'<RTN>
    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 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 reworked.

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

THE OPTION FIELD (Cont)

1. B..... The BEFORE option displays the line BEFORE the command is executed.

2. A..... The AFTER option displays the line AFTER the command of execution.

3. BA..... This is a combination of all the previous commands. The line is displayed BEFORE and AFTER the command of execution.

For example, suppose 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 EVERY WHERE 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 means of this command:

```
--^+'RED'EDITBA,RED,WHITE,1<RTN>
ITS FLEECE WAS RED AS SNOW
ITS FLEECE WAS WHITE AS SNOW
```

Note that after the command EDIT, the options A & B 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 meaningness, as the line (or lines) cannot be displayed after deletion.

****

THE PARAMETER FIELD

This is a special field used with the EDIT, 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, qualifier strings, options, and parameter fields (if required) that may be used with this command. NOTE: All possible combinations of range expressions, qualifier strings, options, and parameter fields are not given. You must review the section for each of these expressions to determine all possible command structures.

BLITZ

The BLITZ command discards all text in the working buffer. Because of the drastic action this command takes, BLITZ followed by a <RTN> results in the question, "SURE?" A 'Y' in response to "SURE?" proceeds with BLITZ. You may abort BLITZ at the query "SHURE>" by typing an 'N' or any other character except 'Y'. The range option, qualifier, and parameter fields are ignored by a BLITZ command.

BYE

The BYE command is the normal way to exit from EDIT to the operating system. The BYE command first performs a FLUSH, which causes all remaining text in the buffer and in the input file to be written to the output file. EDIT then closes both input and output files and exits to the operating system.

If no output file has been specified (i.e., no NEWOUT command performed), BYE types the message:

```
NO NEWOUT FILE
```

An output filename should be specified via NEWOUT, and the BYE command retyped. If you wish to exit the editor without writing a file, simply type CTRL-D.

THE COMMANDS (Cont)

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. Once you have used DELETE, the current pointer is reset to the first line of the text buffer. You may not delete the entire buffer. To do this, you would use the BLITZ command.

For example:

```
-- PRINT<RTN>
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 EVERY WHERE THAT MARY WENT
THIS REPLACES THE SEVENTH LINE
THE LAMB WAS SURE TO GO
THIS REPLACES THE OLD LAST LINE
THIS IS AN ADDITIONAL LAST LINE
--


--$DELETE<RTN>
--$PRINT<RTN>
THIS REPLACES THE OLD LAST LINE
--


--^,$-1DELETE<RTN>
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 EVERY WHERE THAT MARY WENT
THIS REPLACES THE OLD SEVENTH LINE
THE LAMB WAS SURE TO GO
--PRINT<RTN>
THIS REPLACES THE OLD LAST LINE
--
```

THE COMMANDS (Cont)

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, but 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 of 65,536 is assumed.

The EDIT command makes full use of all range expressions, qualifier strings, options; 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<RTN>
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 EVERY WHERE 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<RTN>

-- PRINT<RTN>
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 EVERY WHERE THAT MARY WENT
THIS IS THE SEVENTH OF MANY LINES
THE LAMB WAS SURE TO GO
THIS IS THE LAST OF MANY LINES
--
```

THE COMMANDS (Cont)

EDIT (Cont)

NOTE: The delimiting characters may not appear in the old or new strings. For example, if a comma is used as a delimiter, EDIT will not allow you to enter 'MARY,' as either the old or new string. If either string contains a comma, use a slash [/] or some other character which does not appear in the string as a delimiter.

FLUSH

Use the FLUSH command when editing is complete. It causes the working buffer to be written to the output file, and then any remaining text on the input file is copied over to the output file without modification. The FLUSH command then closes both the input and the output files.

The FLUSH command does not use range expressions, options, qualifier strings, or parameter fields. NOTE: After the lines are written, they are deleted from the buffer.

INSERT

The INSERT command places EDIT 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 specified in the range expression. If you use a null in the range expression, the text will be inserted on the line immediately following the text indicated by the current pointer. If the range expression is given in the form "-n," the line or lines are still inserted after the line specified in the range command. The range command " INSERT" is a special case used to

insert a line before the first line in the buffer.

For example:

```
-- PRINT<RTN>
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 EVERY WHERE THAT MARY WENT
THIS IS THE SEVENTH LINE
THE LAMB WAS SURE TO GO
THIS IS THE LAST LINE
--
```

THE COMMANDS (Cont)

INSERT (Cont)

```
--$INSERT<RTN>
THIS IS AN ADDITIONAL LAST LINE<RTN>
<CTRL-C>
--


--^+'FIFTH'INSERT<RTN>
THIS IS A NEW LINE INSERTED AFTER THE FIFTH LINE<RTN>
<CTRL-C>
--


-- PRINT<RTN>
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 EVERY WHERE 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 CTRL-C to return to the command mode. If you strike CTRL-C after inserting a partial line, that line will be lost. Therefore, to preserve the last line of inserted text, press <RTN>; prior to typing CTRL-C. This will create a new blank line in the text buffer.

NEWIN

The NEWIN command opens a disk file for reading. It is not necessary to have an input disk file to do editing. You may create a new file in the buffer by using the INSERT command. The name of the disk file to be opened is contained in the NEWIN command parameter field. The parameter field for the NEWIN command is in the form:

```
<delimiter><fspec><delimiter>
```

The default device for <fspec> is SY0:. There is no default extension.

After the NEWIN command is executed, the file is open for reading. You must use the READ command to read text into the buffer. Note that the NEXT, FLUSH, and BYE commands also cause text to be read. When the end of the file is read, the message "End of File" is typed on the console. EDIT will automatically close the file.

THE COMMANDS (Cont)

NEWIN (Cont)

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 the new file. The reply 'Y', to 'SURE?' instructs NEWIN to proceed to find the new file.

For example:

```
--NEWIN/WORK.TXT/<RTN>

OLD INPUT FILE NOT FINISHED.  ARE YOU SURE?Y<RTN>
```

NEWOUT

Use the NEWOUT command to open an output file. The filename is supplied in the parameter field in the same manner as in the NEWIN command. The form of the parameter field is:

```
<delimiters><fspec><delimiters>
```

EDIT will open the file for WRITE, leaving it ready for text. Use the WRITE, FLUSH, or BYE commands, as appropriate, to write text into the file. Note that any existing file by that name will be overwritten.

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 'Y' reply permits NEWOUT to close the current output file and open a new one. Note that the closing of the previous file will cause any preexisting file by that name to be replaced by the newly closed one.

For example:

```
 --NEWOUT/WORK2.TXT/<RTN>
```

NEXT

The NEXT command performs the following sequence:

```
$WRITE
READ
```

THE COMMANDS (Cont)

NEXT (Cont)

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

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 the section "CHARACTER SET" on page 10-2.

READ

The READ command is used to input text into the working buffer. Lines are read into the buffer until the end of file is reached, or until less than 700 free bytes of buffer space remain. In the first case, EDIT prints the message:

```
END OF FILE
```

In the second case, EDIT prints:

```
NOT ENOUGH RAM
```

Text input by the READ command is appended to the working buffer.

REPLACE

This command causes the eligible line(s) in the command range to be replaced. After you have typed some form of REPLACE and <RTN>, the cursor moves to the beginning of the new line so you can enter the replacement text. Type the replacement lines one at a time, following each with a <RTN>. You may use tabs as part of the replacement text;

THE COMMANDS (Cont)

REPLACE (Cont)

and you may use BACKSPACE, DELETE, and RUBOUT to edit the new line you are entering. However, simply typing <RTN> without entering any text will cause EDIT to replace the old line with a blank line.

After the lines indicated by the range expression have been replaced, EDIT reverts to the command mode. Typing CTRL-C returns the editor to the command mode, erasing the current line.

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

For example:

```
-- PRINT<RTN>
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 EVERY WHERE 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
--


--^+5REPLACE<RTN>
THIS IS A NEW LINE INSERTED AFTER THE FIFTH LINE
THIS LINE IS REPLACED<RTN>
--


--^+6,+'LAST'REPLACE'LINE'<RTN>
THIS REPLACES THE OLD SEVENTH LINE<RTN>
THIS REPLACES THE OLD LAST LINE<RTN>
--

-- PRINT<RTN>
THIS IS THE FIRST LINE
MARY HAD A LITTLE LAMB
THIS IS THE THIRD LINE
ITS FLEECE WAS WHITE AS SNOW
THIS LINE IS REPLACED
AND EVERY WHERE THAT MARY WENT
THIS REPLACES THE OLD SEVENTH LINE
```

THE COMMANDS (Cont)

REPLACE (Cont)

```
    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.

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 into the output file, and more text is added from the input file. The SEARCH stops when the end of the file is reached, 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 sign (=) for the range expression. The SEARCH command uses the range expression, but does not use option or parameter fields.

For example:

```
    -- PRINT<RTN>
    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 EVERY WHERE THAT MARY WENT
    THIS IS THE SEVENTH LINE
    THE LAMB WAS SURE TO GO
    THIS IS THE LAST LINE
    --


    "--"'SEARCH"FIFTHE"<RTN>'
    "--"'=EDITBA,FIFTHE,FIFTH,<RTN>'
    "THIS IS THE FIFTHE LINE"
    "THIS IS THE FIFTH LINE"
```

THE COMMANDS (Cont)

SEARCH (Cont)

NOTE: The given character string is inclosed in double quotes ["], not single quotes ['].

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 by displaying 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.

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 computer employed in this example has 16k RAM 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:
```

NOTE: EDIT requires some room for work space. It refuses to allow more text when unused space in the buffer is down to 200 free bytes. These 200 bytes are its work space.

THE COMMANDS (Cont)

USE (Cont)

```
--  USE<RTN>
LINES = 00015
USED  = 00273
FREE  = 08299
--
```

WRITE

The WRITE command outputs text from the buffer into the output file. It starts at the top of the buffer and continues to the first line of the command range.

Thus, the command:

```
$WRITE<RTN>
```

writes the entire buffer.

This command uses range expressions and options. It has no parameter field.

You can use the WRITE command to output only certain lines of text to the output file. To do this, first specify the lines to be written, and then use the BLITZ command to delete all other lines of text from the buffer. Then enter a BYE or FLUSH command.

For example:

```
-- PRINT<RTN>
10 LINE INPUT "WHAT IS YOUR NAME";B$
30 END
--NEWOUT/SY2:TEST.BAS/<RTN>
--+1WRITE<RTN>
--BLITZ<RTN>
Are You Sure?Y
--BYE<RTN>

End of File
>TYPE SY2:TEST.BAS<RTN>
10 LINE INPUT "WHAT IS YOUR NAME?;B$
20 PRINT "HELLO, ";B$;"."
```

Note that all lines up to and including the line specified in the WRITE command are included in the output file.

****

COMMAND COMPLETION

When EDIT 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, EDIT supplies and prints this character for you.

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

```
--^+'TUESDAY'
```

and EDIT refuses the "P" in PRINT, it means that it found no line containing Tuesday. This check is done before the command is executed, so you can use a backspace to eliminate the TUESDAY and replace this string with a valid string for the text.

NOTE: If no information exists in the text buffer, EDIT 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 no text to print.
****

APPENDIX 10-A: ERROR MESSAGES

<BELL>

EDIT rings the terminal bell when an illegal command character (for the current command) is typed. EDIT also rings the bell if the line expression just entered applies to no line in the text file.

END OF FILE

The end of the file was reached on the input file. The file was automatically closed. This message is informative and does not indicate an error of any kind.

ERROR - xxx ... xxx

The operating system detected the error, "xxx ... xxx." This expression stands for an error description which changes from time to time, depending upon the circumstances. It is normally self-explanatory, and is discussed in the HDOS Operating System Manual, Chapter Three, "System Optimization," Appendix 3-A, page 3-33, "Error Messages."

FIRST <= LAST

The line matching the FIRST LINE expression must occur before, or be the same as, the line matching the "last line" expression.

ILLEGAL FILENAME

The file specification used in a NEWIN or NEWOUT command contains too many characters. The file specification should contain no blanks or spurious characters.

NO OUTPUT FILE

EDIT could not write the text buffer to disk because no output file has been opened. This message can come in response to a WRITE, FLUSH, SEARCH, NEXT, or BYE command -- all of which can cause text to be written to disk.

NOT ENOUGH RAM

There is not enough free RAM to complete the operation. You can free up RAM by deleting some text lines, or by writing some text lines to

APPENDIX 10-A: ERROR MESSAGES (Cont)

NOT ENOUGH RAM (Cont)

the output file. Note that this message is a typical response to the READ or NEXT command, and only means that EDIT ran out of RAM space before it read the end-of-file. EDIT automatically stops reading with several hundred bytes still free to allow room for editing.

NOT FOUND

The line specified in the SEARCH command could not be located.

OLD INPUT FILE NOT FINISHED ARE YOU SURE?

You have attempted to specify a new input file (via NEWIN) before the end of file was read on the old one. Replying with a Y (YES) closes the old input file and opens the new one. Replying with a N (NO) leaves the old file open.

OLD OUTPUT FILE NOT FINISHED ARE YOU SURE?

You have attempted to specify a new output file (via NEWOUT) before the old file was closed. Replying with a Y (YES) closes the old output file and opens the new one. Replying with a N (NO) leaves the old output file open.

APPENDIX 10-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."

COMMAND STRUCTURE

The general format for an editor command is:

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

RANGE EXPRESSION FORMS

NULL - First line in previous command range.

BLANK - The entire working buffer.

= - The previous command range.

A single line expression.

Multiple expressions.

MULTIPLE LINE EXPRESSION FORMS

EXPRESSION	DEFINITION
^	The first line in the buffer.
$	The last line in the buffer.
NULL	The first line in the previous command range.
COMMA	Separates (delimits) multiple line expressions.
+n	Move forward n decimal lines.
-n	Move backward n decimal lines.
+'STRING'	Move forward until 'STRING' is located.
-'STRING'	Move backward until 'STRING' is located.

APPENDIX 10-B: COMMAND SUMMARY (Cont)

VERB (COMMAND) FORMS

BLITZ ..... Discards all text after a Y reply to ARE YOU SURE? (Refer to page 10-12 for details.)

BYE ..... Exit to HDOS Operating System after flushing text and closing files. (Refer to page 10-12 for details.)

DELETE ..... Deletes eligible lines in command range (* except the entire buffer). (Refer to page 10-13 for details.)

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. (Refer to page 10-14 for details.)

FLUSH ..... Writes working buffer, balance of input file, and end-of-file character onto output file. Use when editing is complete. Text is deleted from buffer when complete. (Refer to page 10-15 for details.)

INSERT ..... Add to text buffer from keyboard. CTRL-C returns Editor to command mode. (Refer to page 10-15 for details.)

NEWIN ..... Opens a new file to be read in. Parameter field specifies filename <delimiter><name><delimiter>. (Refer to page 10-16 for details.)

NEWOUT ..... Opens a new file for output. Parameter field specifies filename <delimiter><name><delimiter>. (Refer to page 10-17 for details.)

NEXT ..... Writes working buffer onto output file. Then fills buffer from the input file. (Refer to page 10-17 for details.)

PRINT ..... Print eligible lines on computer screen. (Refer to page 10-17 for details.)

READ ..... Reads text into memory from input file. (Refer to page 10-17 for details.)

REPLACE ..... Replaces eligible lines in command range from keyboard. CTRL-C returns Editor to Command Mode. (Refer to page 10-17 for details.)

SEARCH "string" Searches text buffer and input file after initial line for 'STRING.' Search stops when STRING is found, or end-of-file is found. Command range is set to line containing 'STRING.' (Details are on page 10-20.)

APPENDIX 10-B: COMMAND SUMMARY (Cont)

VERB (COMMAND) FORMS (Cont)

USE ..... Displays number of lines in buffer, bytes used, and bytes free. (Refer to page 10-21 for details.)

WRITE ..... Writes text from the working buffer to output file. Writes start of buffer to first line of common range. After writing, lines are not deleted. (Refer to page 10-22 for details.)

QUALIFIER STRING

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

OPTION

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.

PARAMETER FIELD

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

APPENDIX 10-C: SAMPLE EDITED SOURCE FILE

The following edited source file is typical of one you might create using EDIT. It is intended to show examples of some of EDIT's editing capabilities.

```
    -- PRINT
              ORG     100000A-160Q
    FPNRM     EQU     073173A
              INX     B          TO
              INX     B           EXPONENT
              LDAX    A
              ANA     B          SET CONDX CODE
              RZ
              DCR     A          /2
              JZ      USRI       IF UNDER FLOW
              DCR     A          /2 again (/4)
    USR1      STAX    B
              CALL    FPNRM
              RET

              END     START

    --^+'EQU'INSERTB
    FPNRM     EQU     073173A
    START     INX     B          INC
    --^+'073173'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    A          (A) = ACCX EXP
              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
```

APPENDIX 10-C: TYPICAL EDITED SOURCE FILE (Cont)

```
    --^+'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     1200000A-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                IN CASE 0

              END     START


    --NEWOUT"SY1:TEXT.ASM"<RTN>
    --BYE<RTN>
    END OF FILE
    >              (HDOS PROMPT)
**********************************************************************
```

INDEX

INDEX (Cont)

HDOS SOFTWARE REFERENCE MANUAL

HDOS DISK OPERATING SYSTEM

VERSION 3.0

CHAPTER 11

HEATH ASSEMBLY LANGUAGE

ASM

HEATH DISK OPERATING SYSTEM

SOFTWARE REFERENCE MANUAL

VERSION 3.0

 Through the years it continued to be improved by successive revisions which included 1.5, 1.6, and finally 2.0. It was entered into public domain on 19 July 1989 per letter by Jim Buszkiewicz, Managing Editor, Heath Users' Group, P.O. Box 217, Benton Harbor, MI 49022-0217 (616)982-3463. A copy of this letter is available for public inspection.

This manual is indicative of further improvements and provides for the latest revision, HDOS 3.0 and HDOS 3.02. Revision 3.0 is detailed in chapters 1, 2, and 3, while chapters 4 ,5, 6, 7, 8, 13 and 14, are related to revision 3.02. Chapters 9 through 12, with minor improvements, are essentially picked up from the original HDOS 2.0 manual. Indeed, HDOS is still alive and well!

SPECIAL DISCLAIMER: The Heath Company cannot provide consultation on either the HDOS Operating System or user-developed or modified versions of Heath software products designed to operate under the HDOS Operating System. Do not refer to Heath for questions.

Instead, you are invited to direct any questions concerning the Heath Disk Operating System (HDOS) to Mr. Kirk L. Thompson, Editor "Staunch 89/8" Newsletter, P.O. Box 548, #6 West Branch Mobile Home Village, West Branch, IA 52358.

TABLE OF CONTENTS

INTRODUCTION

The Heath Assembly Language program (ASM) lets you use source (symbolic) programs using letters, numbers, and symbols that are meaningful because they are abbreviated English statements. These source programs must be generated with a text editor, such as PIE, Textpro, EDIT19, or the Heath Text Editor (EDIT). ASM assembles the source program into a listing with an optional cross-reference table and an object program in absolute binary format executable by your computer.

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 ASM.

ASM is designed to produce programs which run on an H8/H89 Computer System. Therefore, it assembles 8080 symbolic assembly code. The H8 uses the 8080 microprocessor, whereas the H89/Z90 computers use the Z80 microprocessor. If Heath decided to assemble only in the Z80 code, the results would only run on the H89/Z90 type of computers. Since they choose to assemble the 8080 code, the results apply to both computers. This is not untypical, since most of the typical user programs are created to run on both codes.

The ASM chapter presumes that you have read the HDOS Operation Manual which came with the H89, Z90, or H8 computer, and are also familiar with the 8080 instruction set, I/O formats, memory formats, (and for H8 or H8/H19 owners only,) front panel configuration. A thorough knowledge of this data is essential to efficient assembly language programming.

****

CONVENTIONS

To clarify the text, statements made by the computer will be set off by quotation marks ["]. Similiarily, responses by the user will be set off by apostrophe marks ['']. NOTE: Since certain ASM commands may also use apostrophes, the reader should be able to discern according to context.

****

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 ASM are listed as follows:

THE CHARACTER SET (Cont)

1. The letters A through Z [NOTE: Lower case letters are acceptable for quoted strings and comments.]

2. The numerals 0 through 9.

3. The characters period [.] and dollar sign [$], which are considered alphabetic.

4. The symbols: : = % # [ ] , ; " ' + - _ ! ?

5. Blanks and tabs.

****

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 separated by BLANKS and TABS. The four fields are:

LABEL OPCODE OPERAND COMMENT

The label on comment fields is 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 to 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 or contain a TAB. 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, each of the following character groups are valid labels:

A A3 . C9D4 START .. $END END$PGM

For example, if the current location counter is set to 042 200 and the statement:

STATEMENTS (Cont)

THE LABEL FIELD (Cont)

```
START   MOV  A,B
```

is the next statement, the assembler assigns the value of 042 200 to the label START. Subsequent references to START refer to location 042 200.

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 or TAB. 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 the program assembly.

The opcode field is terminated by a BLANK, a TAB, or 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 or TAB. 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 or TAB 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 TAB 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 TAB terminates the operand field and begins the comment field.

STATEMENTS (Cont)

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 or TAB. 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 and TAB characters. Format control is primarily used to produce a program which is easily read. Format control has no effect on the assembly process of the source program. The following two statements are interpreted identically. The first one uses BLANKS and the second one uses TABS.

```
    START MOV A,B  THIS IS A COMMENT

    START     MOV A,B     THIS IS A COMMENT
```

****

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.

OPERAND EXPRESSIONS (Cont)

OPERATORS

ASM recognizes 5 operators. They are as follows:

+	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 ohly as the first character of 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 ORIGIN SYMBOL

Each of these tokens has the limitations described in the following paragraphs.

INTEGERS

Decimal integers ranging from 0 to 65,535 are allowed, but no decimal place may be specified. The radix of an integer expression is assumed to be the 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	H	Hexadecimal
O or Q	Octal	S	Offset Octal
D	Decimal		

OPERAND EXPRESSIONS (Cont)

For example:

Expression	Radix	Decimal Value
000 00011B	Binary	3
160Q	Octal (Also 160O)	112
3200	Decimal (Also 3200D)	3200
77000A	Offset Octal	16128
021AH	Hexadecimal	282

Legal Integer	Illegal Integer	Comments
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 "The 8080 Opcodes," Page 11-8. All symbols must consist of legal ASM 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.

OPERAND EXPRESSIONS (Cont)

CHARACTER STRING

A character string consisting of one or two legal characters may be used as a token in an ASM expression. Such a character string is inclosed in a single apostrophe ['']. For example:

```
'A'     The character A (Value 101Q Octal)
'GL'    The character string GL (Value 107 114A)
'"'     The character quotation mark (Value 042Q Octal)
```

THE ORIGIN SYMBOL [*]

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
        EQU    ***
```

defines the symbol A to have the value 100. The first statement, "ORG 10," sets the origin counter to the value of 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 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 a discussion of the format used to describe each instruction.

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.

THE 8080 OPCODES (Cont)

INSTRUCTION TYPES (Cont)

* Logical Group ------ AND, OR, EXCLUSIVE-OR, compare, rotate, or complement data in registers or 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.

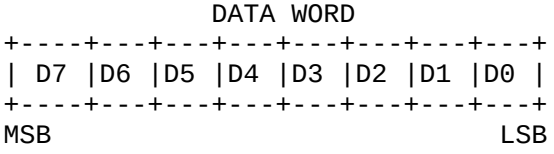
TERMS, SYMBOLS, AND NOMENCLATURE

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 65,535 bytes of memory, which may consist of both read-only memory (ROM) elements and random-access (RAM) elements (read/write memory).

Data in the 8080 is stored in the form of 8-bit binary integers:



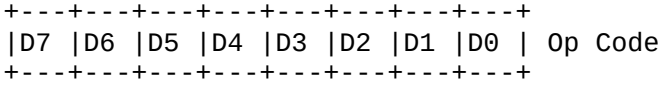
When a register or data word contains a binary number, it is necessary to establish the order in which the bits of the number are written. In the Intel 8080, BIT 0 is referred to as the "Least Significant Bit" (LSB), and BIT 7 of an 8-bit number is referred to as the "Most Significant Bit" (MSB).

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.

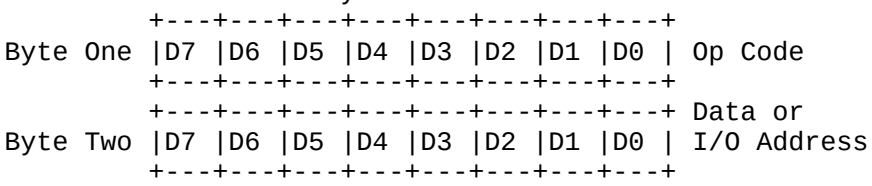
8080 OPCODES (Cont)

INSTRUCTION AND DATA FORMATS (Cont)

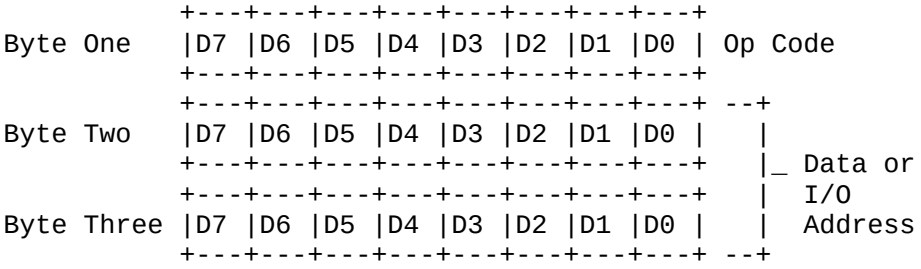
Single-Byte Instructions



Two-Byte Instructions



Three-Byte Instructions



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 Direct -- 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.

8080 OPCODES (Cont)

ADDRESSING MODES (Cont)

* 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 are in the second register.]

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:

1. ZERO
2. SIGN
3. PARITY
4. CARRY
5. 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.

8080 OPCODES (Cont)

CONDITION FLAGS (Cont)

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 of 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 an 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 instructions
byte 3	The third byte of the instructions

8080 OPCODES (Cont)

SYMBOLS AND ABBREVIATIONS (Cont)

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

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,S:

RP	Register Pair
00	B-C
01	D-E
10	H-L
11	SP

8080 OPCODES (Cont)

SYMBOLS AND ABBREVIATIONS (Cont)

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, Cy, AC	The condition flags. Zero, Sign, Parity, Carry, and Auxiliary Carry, respectively.

8080 OPCODES (Cont)

SYMBOLS AND ABBREVIATIONS (Cont)

NOTE: ASM recognizes the E as well as the Z defining the zero bit. Therefore, JZ (jump zero) or JE (jump equal) are both valid opcodes.

( ) The contents of the memory location or registers is inclosed in parentheses.

<-- "Is transferred to"

/\ Logical AND

-\/- Exclusive OR (Note: There is a horizontal bar drawn in the middle of the symbol which the H89 cannot copy.)

\/ Inclusive OR

\+ Addition

\- Two's complement subtraction

* Multiplication

<--> "Is exchanged with"

-- The one's complement (e.g., $\overline{A}$)

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 ASM format, consisting of the opcode and operand fields, is printed in BOLDFACE on the left of the first line.

2. The name of the instruction is inclosed 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.

8080 OPCODES (Cont)

DESCRIPTION FORMAT (Cont)

5. The following line(s) contain the binary fields and patterns that comprise the machine instructions.

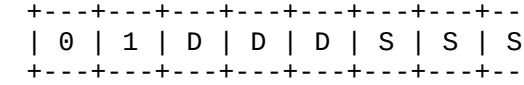
6. The last two lines contain incidental information about the execution of the instruction. The number of machine cycles and states required to execute the instruction are listed first. If the instruction has two possible execution times as a conditional jump, both times will be listed, separated by a slash. Next, any significant data addressing modes (see "Addressing Modes," Page 10) 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)

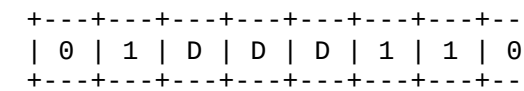
(r1)<--(r2)
The contents of register r2 is moved to register r1.



Cycles: 1 Addressing: register
States: 5 Flags: none

MOV r, M (Move from memory)

(r)<--((H)(L))
The content of the memory location whose address is in registers H and L is moved to register r.



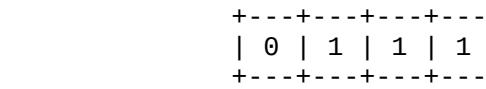
Cycles: 2 Addressing: reg. indirect
States: 7 Flags: none

8080 OPCODES (Cont)

DATA TRANSFER GROUP (Cont)

MOV M, r (Move from memory)

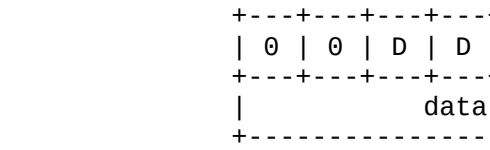
((H)(L))<--(r)
The content of register r is moved to the memory location whose address is in registers H and L.



Cycles: 2 Addressing: reg. indirect
States: 7 Flags: none

MVI r, data (Move to register immediate)

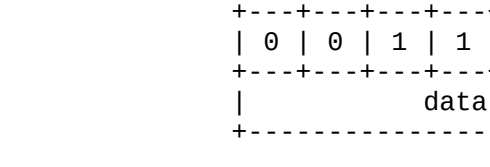
(r)<--(byte 2)
The content of byte 2 of the instruction is moved to register r.



Cycles: 2 Addressing: immediate
States: 7 Flags: none

MVI M, data (Move to memory immediate)

((H)(L))<--(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 Addressing: immed./reg. indirect
States: 10 Flags: none

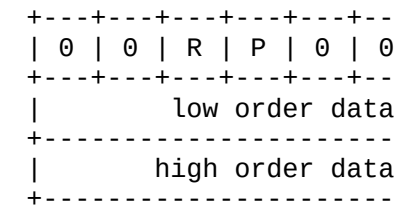
8080 OPCODES (Cont)

DATA TRANSFER GROUP (Cont)

LXI rp, data 16 (Load register pair immediate)

(rh)<--(byte 3),
(rl)<--(byte 2)

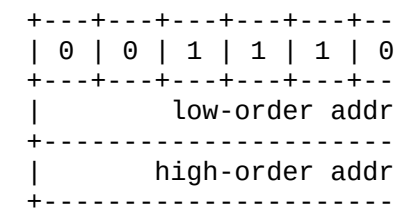
Byte 3 of the instruction is moved into the high-order register (rh) of the register pair rp. Byte 2 of the instruction is moved into the low-order register (rl) of the register pair rp.



Cycles: 3 Addressing: immediate
States: 10 Flags: none

LDA addr (Load Accumulator direct)

(A)<--((byte 3)(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.



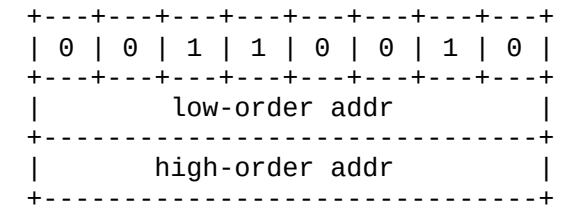
Cycles: 4 Addressing: direct
States: 13 Flags: none

8080 OPCODES (Cont)

DATA TRANSFER GROUP (Cont)

STA addr (Store accumulator direct)

((byte 3)(byte 2))<--(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.

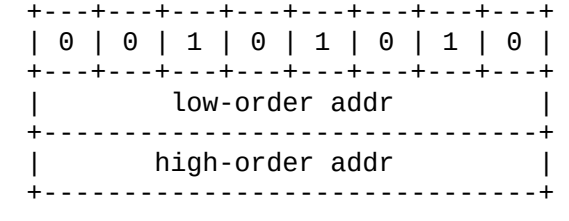


Cycles: 4 Addressing: direct
States: 19 Flags: none

LHLD addr (Load H and L direct)

(L)<--((byte 3)(byte 2))
(H)<--((byte 3)(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 Addressing: direct
States: 16 Flags: none

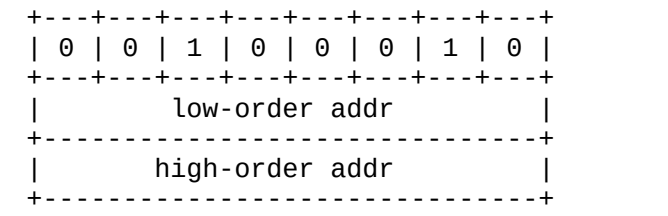
8080 OPCODES (Cont)

DATA TRANSFER GROUP (Cont)

SHLD addr (Store H and L direct)

((byte 3)(byte 2)<--(L)
((byte 3)(byte 2) + 1)<--(H)

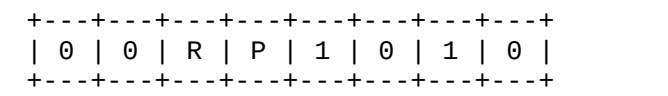
The content of register L is moved to the memory location whose address is specified in byte 2 and byte 3. The content of register H is moved to the succeeding memory location.



Cycles: 5 Addressing: direct
States: 16 Flags: none

LDAX rp (Load accumulator direct)

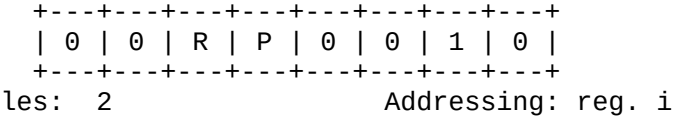
(A)<--((rp))
The content of memory location whose address is in the register pair rp is moved to register A. NOTE: Only register pairs rp = B (registers B and C) or rp = D (registers D and E) may be specified.



Cycles: 2 Addressing: reg. indirect
States: 7 Flags: none

STAX rp (Store accumulator indirect)

((rp))<--(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.



Cycles: 2 Addressing: reg. indirect
States: 7 Flags: none

8080 OPCODES (Cont)

DATA TRANSFER GROUP (Cont)

XCHG (Exchange H and L with D and E)

(H)<-->(D)
(L)<-->(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

8080 OPCODES (Cont)

ARITHMETIC GROUP

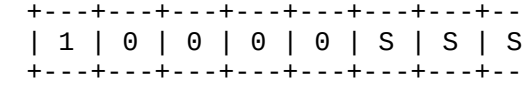
This group of instructions performs arithmetic operations on data in registers and memory.

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)

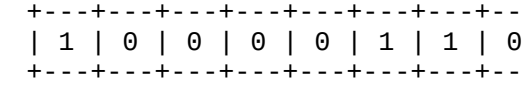
(A)<--(A) + (r)
The content of register r is added to the content of the accumulator. The result is placed in the accumulator.



Cycles: 1 Addressing: register
States: 4 flags: Z,S,P,CY,AC

ADD M (Add memory)

(A)<--(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.



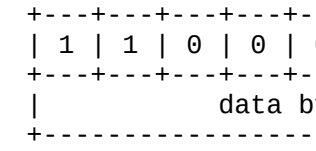
Cycles: 1 Addressing: reg. indirect
States: 7 Flags: Z,S,P,CY,AC

8080 OPCODES (Cont)

ARITHMETIC GROUP (Cont)

ADI DATA (add immediate)

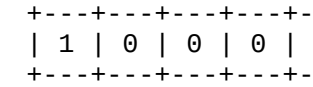
(A)<--(A) + (byte 2)
The content of the second byte of the instruction is added to the content of the accumulator. The result is placed in the accumulator.



Cycles: 2 Addressing: immediate
States: 7 Flags: Z,S,P,CY,AC

ADC r (Add Register with carry)

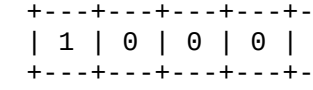
(A)<--(A) + (r) + (CY)
The content of register r and the content of the carry bit are added to the content of the accumulator. The result is placed in the accumulator.



Cycles: 1 Addressing: register
States: 4 Flags: Z,S,P,CY,AC

ADC M (Add memory with carry)

(A)<--(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.



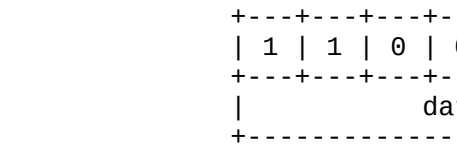
Cycles: 2 Addressing: reg. indirect
States: 7 Flags: Z,S,P,CY,AC

8080 OPCODES (Cont)

ARITHMETIC GROUP (Cont)

ACI data (Add immediate with carry)

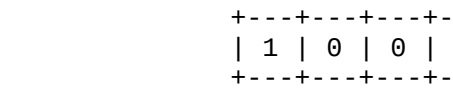
(A)<--(A) + (byte 2) + (CY)
The content of the second byte of the instruction and the content of the CY flag are added to the contents of the accumulator. The result is placed in the accumulator.



Cycles: 2 Addressing: immediate
States: 7 Flags: Z,S,P,CY,AC

SUB r (Subtract Register)

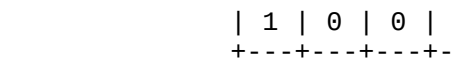
(A)<--(A) -- (r)
The content of register r is subtracted from the content of the accumulator. The result is placed in the accumulator.



Cycles: 1 Addressing: register
States: 4 Flags: Z,S,P,CY,AC

SUB M (Subtract memory)

(A)<--(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.



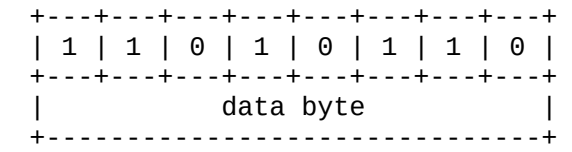
Cycles: 2 Addressing: reg. indirect
States: 7 Flags: Z,S,P,CY,AC

8080 OPCODES (Cont)

ARITHMETIC GROUP (Cont)

SUI DATA (Subtract immediate)

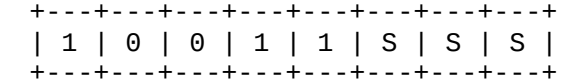
(A)<--(A) -- (byte 2)
The contents of the second byte of the instruction is subtracted from the contents of the accumulator. The result is placed in the accumulator.



Cycles: 2 Addressing: immediate
States: 7 Flags: Z,S,P,CY,AC

SUBB r (Subtract Register with borrow)

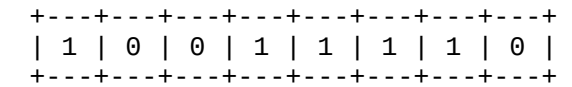
(A)<--(A) -- (r) -- (CY)
The contents of register r and the contents of the CY flag are both subtracted from the accumulator. The result is placed in the accumulator.



Cycles: 1 Addressing: register
States: 4 Flags: Z,S,P,CY,AC

SBB M (Subtract memory with borrow)

(A)<--(A) -- ((H)(L)) -- (CY)
The content of the memory location whose address is contained in the H and L registers, and the contents of the CY flag are both subtracted from the accumulator. The result is placed in the accumulator.



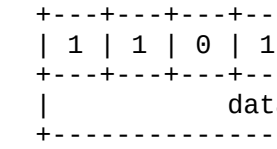
Cycles: 2 Addressing: reg, indirect
States: 7 Flags: Z,S,P,CY,AC

8080 OPCODES (Cont)

ARITHMETIC GROUP (Cont)

SBI DATA (Subtract immediate with borrow)

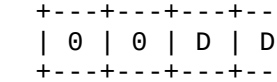
(A)<--(A) -- (byte 2) -- (CY)
The contents of the second byte of the instruction and the contents of the CY flag are both subtracted from the accumulator. The result is placed in the accumulator.



Cycles: 2 Addressing: immediate
States: 7 Flags: Z,S,P,CY,AC

INR r (Increment Register)

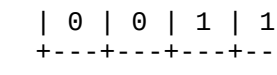
(r)<--(r) + 1
The contents of register r is incremented by one. NOTE: All condition flags except CY are affected.



Cycles: 1 Addressing: register
States: 5 Flags: Z,S,P,AC

INR M (Increment memory)

((H)(L))<--((H)(L)) + 1
The contents 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.



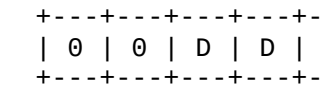
Cycles: 3 Addressing: reg. indirect
States: 10 Flags: Z,S,P,AC

8080 OPCODES (Cont)

ARITHMETIC GROUP (Cont)

DCR r (Decrement register)

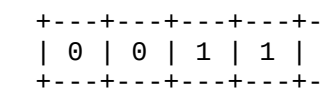
(r)<--(r) -- 1
The contents of register r is decremented by one. NOTE: All condition flags except CY are affected.



Cycles: 1 Addressing: register
States: 5 Flags: Z,S,P,AC

DCR M (Decrement memory)

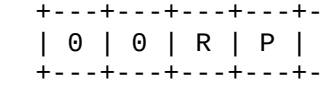
((H)(L))<--((H)(L)) -- 1
The contents of the memory location whose address is contained in the H and L registers is decremented by one. NOTE: All condition flags except CY are affected.



Cycles: 3 Addressing: reg. indirect
States: 10 Flags: Z,S,P,AC

INX rp (Increment register pair)

(rh)(rl)<--(rh)(rl) + 1
The contents of the register pair rp is incremented by one. NOTE: No condition flags are affected.



Cycles: 1 Addressing: register
States: 5 Flags: none

8080 OPCODES (Cont)

ARITHMETIC GROUP (Cont)

DAD rp (Add register pair to H and L)

(H)(L)<--(H)(L) + (rh)(rl)
The contents of register pair rp is added to the contents 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
Status: 10 Flags: CY

DAA (Decimal Adjust Accumulator)

The eight-bit number in the accumulator is adjusted to form two four-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.

0	0	1	0	0	1	1	1

Cycles: 1
States: 4
Flags: Z,S.P,CY,AC

8080 OPCODES (Cont)

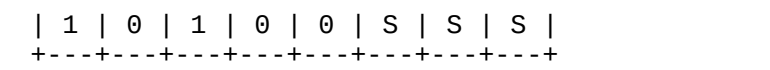
LOGICAL GROUP

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, and Carry flags according to the standard rules.

ANA r (AND register)

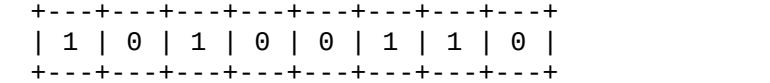
(A)<--(A)/\(r)
The contents of register r is logically ended with the contents of the accumulator. The result is placed in the accumulator. The CY flag is cleared.



Cycles: 1 Addressing: register
States: 4 Flags: Z,S,P,CY,AC

ANA M (AND memory)

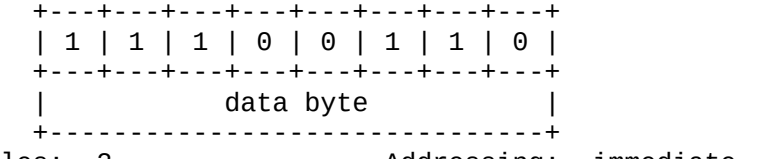
The contents of the memory location whose address is contained in the H and L registers is logically anded with the content of the accumulator. The result is placed in the accumulator. The CY flag is cleared.



Cycles: 2 Addressing: reg. indirect
States: 7 Flags: Z,S,P,CY,AC

ANI data (AND immediate)

(A)<--(A)/\(byte 2)
The contents 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

8080 OPCODES (Cont)

LOGICAL GROUP (Cont)

XRA r (Exclusive OR register)

(A)<--(A)-\/-(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.

1	0	1	0	1	S	S	S

Cycles: 1 Addressing: register
States: 4 Flags: Z,S,P,CY,AC

XRA M (Exclusive OR memory)

(A)<--(A)-\/-((H)(L))
The contents of the memory location whose address is contained in the H and L registers is exclusive-OR'd with the contents 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 Addressing: reg. indirect
States: 7 Flags: Z,S,P,CY,AC

XRI data (Exclusive OR immediate)

(A)<--(A)-\/-(byte 2)
The contents of the second byte of the instruction is exclusive OR'd with the contents 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

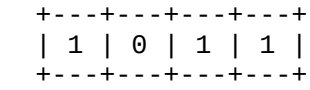
Cycles: 2 Addressing: immediate
States: 7 Flags: Z,S,P,CY,AC

8080 OPCODES (Cont)

LOGICAL GROUP (Cont)

ORA r (OR Register)

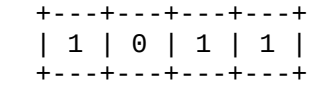
(A)<--(A)\/(r)
The contents of register r is inclusive-OR'd with the contents of the accumulator. The result is placed in the accumulator. The CY and AC flags are cleared.



Cycles: 1 Addressing: register
States: 4 Flags: Z,S,P,CY,AC

ORA M (OR Memory)

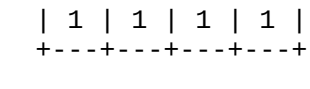
(A)<--((H)(L))
The contents of the memory location whose address is contained in the H and L registers is inclusive-OR'd with the contents of the accumulator. The result is placed in the accumulator. The CY and AC flags are cleared.



Cycles: 2 Addressing: reg. indirect
States: 7 Flags: Z,S,P,CY,AC

ORI data (OR Immediate)

(A)<--(A)\/(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

8080 OPCODES (Cont)

LOGICAL GROUP (Cont)

CMP r (Compare Register)

(A) -- (r)
The contents 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).

1	0	1	1	1	S	S	S

Cycles: 1 Addressing: register
States: 4 Flags: Z,S,P,CY,AC

CMP M (Compare memory)

(A)--((H)(L))
The contents 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)).

1	0	1	1	1	1	1	0

Cycles: 2 Addressing: reg. indirect
States: 7 Flags: Z,S,P,CY,AC

CPI data (Compare immediate)

(A)--(byte 2)
The contents 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).

1	1	1	1	1	1	1	0

Cycles: 2 Addressing: immediate
States: 7 Flags: Z,S,P,CY,AC

8080 OPCODES (Cont)

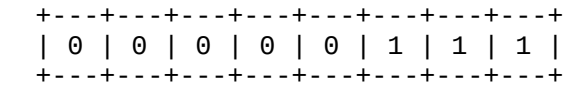
LOGICAL GROUP (Cont)

RLC (Rotate left)

(An+1)<--(An);(A0)<--(A7)
(CY)<--(A7)

NOTE: The following characters are subscripted: n+1, n, 0, 7, 7.

The contents 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.



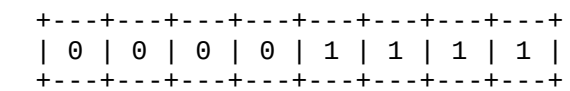
Cycles: 1
States: 4
Flags: CY

RRC (Rotate right)

(An)<--(An-1);(A7)<--(A0)
(CY)<--(A0)

NOTE: The following characters are subscripted: n, n-1, 7, 0, 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.



Cycles: 1
States: 4
Flags: CY

8080 OPCODES (Cont)

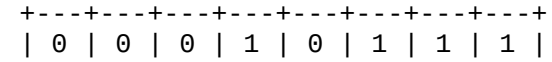
LOGICAL GROUP (Cont)

RAL (Rotate left through carry)

(An+1)<--(An);(CY)<--(A7)
(A0)<--(CY)

NOTE: The following characters are subscripted: n+1, n, 7, 0.

The contents of the accumulator is rotated left one position through the CY flag. The low-order bit is set 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.



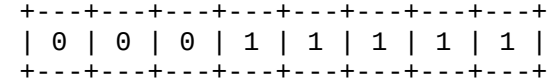
Cycles: 1
States: 4
Flags: CY

RAR (Rotate right through carry)

(An)<--(An+1);(CY)<--(A0)
(A7)<--(CY)

NOTE: The following characters are subscripted: n, n+1, 0, 7.

The contents 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.

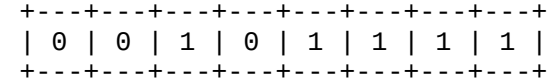


Cycles: 1
States: 4
Flags: CY

CMA (Complement accumulator)

(A)<--(A)

The contents of the accumulator are complemented (zero bits become one, one bits become zero). No flags are affected.



Cycles: 1
States: 4
Flags: none

8080 OPCODES (Cont)

LOGICAL GROUP (Cont)

CMC (Complement carry)

(CY)<--(NOT CY) NOTE: This computer cannot illustrate CY with a "not" bar above the expression.

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)<-- 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

8080 OPCODES (Cont)

BRANCH GROUP

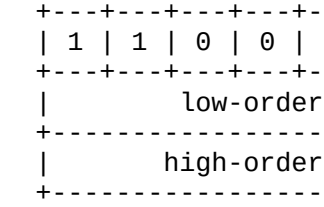
This group of branch instructions alter normal sequential program flow. Condition flags are not affected by any instruction in this group.

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)<--(byte 3)(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

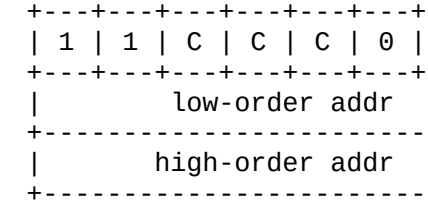
8080 OPCODES (Cont)

BRANCH GROUP (Cont)

JNE JNC JPO JP (Condition jump)
JE JC JPE JM

If (CCC),
(PC)<--(byte 3)(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.

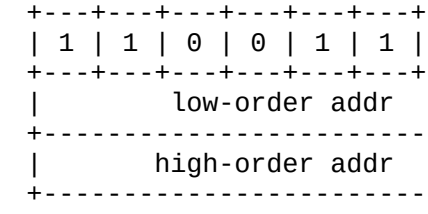


Cycles: 3 Addressing: immediate
States: 10 Flags: none

CALL addr (Call)

((SP)--1)<--(PCH)
((SP)--2)<--(PCL)
(SP)<--(SP)--2
(PC)<--(byte 3)(byte 2)

The high-order 8-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 8-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

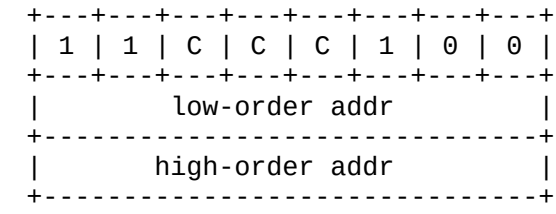
8080 OPCODES (Cont)

BRANCH GROUP (Cont)

CNE CNC CPO CP (Condition call)
CE CC CPE CM

If (CCC)
((SP)--1)<--(PCH)
((SP)--2)<--(PCL)
((SP)<--(SP)--2
(PC)<--(byte 3)(byte 2)

If the specified condition is true, the actions specified in the CALL instructions (see above) are performed. Otherwise, control continues sequentially.

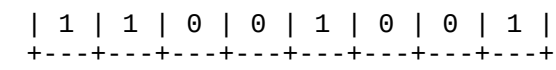


Cycles: 3/5
States: 11/17
Addressing: immediate/reg. indirect
Flags: none

RET (Return)

(PCL)<--((SP)):
(PCH)<--((SP)) + 1);
(SP)<--(SP) + 2;

The content of the memory location whose address is specified in register SP is moved to the low-order 8-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 8-bits of register PC. The content of register SP is incremented by 2.



Cycles: 3
States: 10
Addressing: reg.indirect
Flags: none

8080 OPCODES (Cont)

BRANCH GROUP (Cont)

RNE RNC (Conditional return)
RE RC

If (CCC),
(PCL)<--((SP))
(PCH<--((SP) + 1)
(SP)<--(SP) + 2

If the specified condition is true, the actions specified in the RET instructions (see above) are performed. Otherwise, control continues sequentially.

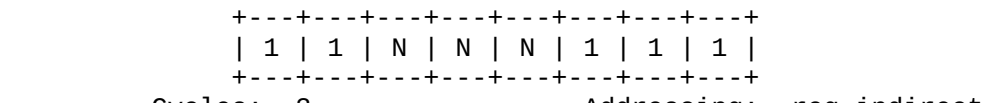
Cycles: 1/3 Addressing: reg. indirect
States: 5/11 Flags: none

RST n (Restart)

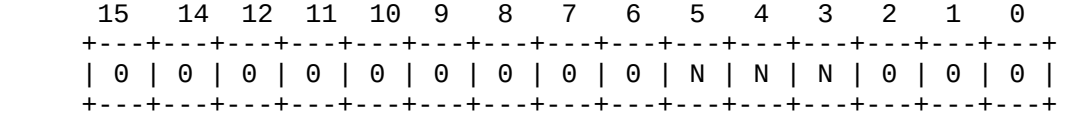
((SP)--1)<--(PCH)
((SP)--2)<--(PCL)
(SP)<--(SP)--2
(PC)<--8 * (NNN)

The high-order 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 8-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.



Cycles: 3 Addressing: reg.indirect
States: 11 Flags: none



Program Counter after Restart

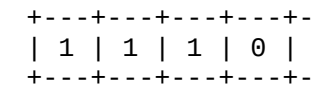
8080 OPCODES (Cont)

BRANCH GROUP (Cont)

PCHL (Jump H and L indirect -- move H and L to PC)

(PCH)<--(H)
(PCL)<--(L)

The content of register H is moved to the high-order 8-bits of register PC. The content of register L is moved to the low-order 8-bits of register PC.



Cycles: 1 Addressing: register
States: 5 Flags: none

****

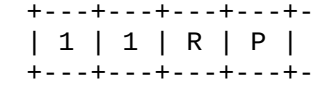
STACK, I/O, AND MACHINE CONTROL GROUP

This group of instructions performs I/O, manipulates the stack, and alters the internal control flags. Unless otherwise specified, condition flags are not affected by any instructions in this group.

PUSH rp (Push)

((SP)--1)<--(rh)
((SP)--2)<--(rl)
(SP)<--(SP)--2

The content of the high-order 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.



Cycles: 3 Addressing: reg. indirect
States: 11 Flags: none

STACK, I/O, AND MACHINE CONTROL GROUP (Cont)

PUSH PSW (Push processor status word)

```
((SP)--1)<--(A)
((SP)--2)0<--(CY),((SP)--2)1<--1
((SP)--2)2<--(P),((SP)--2)3<--0
((SP)--2)4<--(AC),((SP)--2)5<--0
((SP)--2)6<--(Z),((SP)--2)7<--(S)
(SP)<--(SP)--2
```

NOTE: Numbers after the first expression: 0, 2,, 4, 6, are subscript. Also numbers after the third expression: 1, 3, 5, 7 are subscript.

The contents 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)

```
(rl)<--((SP))
(rh)<--((SP) + 1
(SP)<--(SP) + 2
```

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 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 Addressing: reg. indirect
States: 10 Flags: none

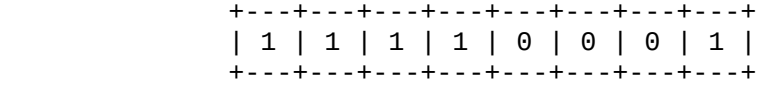
STACK, I/O, AND MACHINE CONTROL GROUP (Cont)

POP PSW (Pop processor status word)

```
(CY)<--((SP))0
(P)<--((SP))2
(AC)<--((SP))4
(Z)<--((SP))6
(S)<--((SP))7
(A)<--((SP) + 1
(SP)<--(SP) + 2
```

NOTE: Numbers after the expressions: 0, 2, 4, 6, 7 are subscript.

The contents of the memory location whose address is specified by the contents 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.



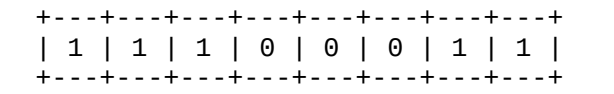
Cycles: 3
States: 10
Addressing: reg. indirect
Flags: Z,S,P,CY,AC

STACK, I/0, AND MACHINE CONTROL GROUP (Cont)

XTHL (Exchange stack top with H and L)

(L)<-->((SP))
(H)<-->((SP) + 1)

The contents of the L register is exchanged with the contents of the memory location whose address is specified by the contents of register SP. The contents of the H register is exchanged with the contents of the memory location whose address is one more than the contents of register SP.

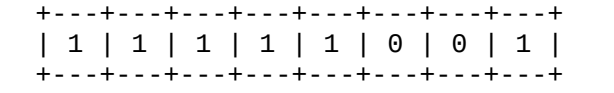


Cycles: 5 Addressing: reg. indirect
States: 18 Flags: none

SPHL (Move HL to SP)

(SP)<--(H)(L)

The contents of registers H and L (16-bits) are moved to register SP.

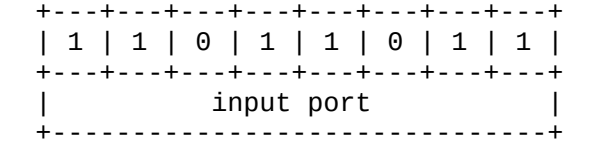


Cycles: 1 Addressing: register
States: 5 Flags: none

IN port (Input)

(A)<--(data)

The data placed on the 8-bit bidirectional data bus by the specified port is moved to register A.



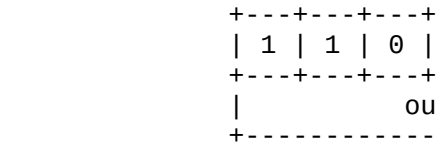
Cycles: 3 Addressing: direct
States: 10 Flags: none

STACK, I/0, AND MACHINE CONTROL GROUP (Cont)

OUT port (Output)

(data)<--(A)

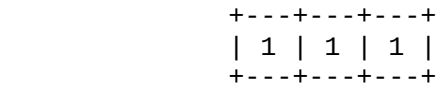
The contents of register A is placed on the 8-bit bidirectional data bus for transmission to the specified port.



Cycles: 3 Addressing: direct
States: 10 Flags: none

EI (Enable interrupt)

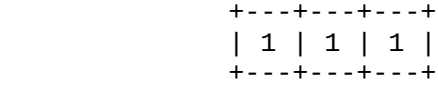
The interrupt system is enabled following the execution of the next instruction.



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

STACK, I/0, AND MACHINE CONTROL GROUP (Cont)

HLT (Halt)

The processor is stopped. The registers and flags are unaffected.

0	1	1	1	0	1	1	0

Cycles: 1
States: 7 Flags: none

NOP (No op)

No operation is performed. The registers and flags are unaffected.

0	0	0	0	0	0	0	0

Cycles: 1
States: 4 Flags: none

****

PSEUDO OPCODES/ASSEMBLER DIRECTIVES

The Heath Assembly Language supports several assembler directives, more commonly known as '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 ASM and are not translated as instructions. Some pseudo ops 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

PSEUDO OPCODES/ASSEMBLER DIRECTIVES (Cont)

DEFINE BYTE, DB (Cont)

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 inclose 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 recognizes 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.

PSEUDO OPCODES/ASSEMBLER DIRECTIVES (Cont)

DEFINE WORD, DW

The DW pseudo defines word constants. The form of the DW pseudo is:

```
LABEL    DW    iexpl, . . . . . ,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 with 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. ASM 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
```

PSEUDO OPCODES/ASSEMBLER DIRECTIVES (Cont)

CONDITIONAL ASSEMBLY PSEUDO OPERATORS (Cont)

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 PROGRAM, 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, an error is flagged, and ASM uses 042 200A.

DEFINE LABEL, EQU

The equate statement is used to assign an arbitrary value to a symbol. The form of the equate statement is:

```
    LABEL    EQU    iexp
```

PSEUDO OPCODES/ASSEMBLER DIRECTIVES (Cont)

CONDITIONAL ASSEMBLY PSEUDO OPERATORS (Cont)

DEFINE LABEL, EQU (Cont)

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.

PSEUDO OPCODES/ASSEMBLER DIRECTIVES (Cont)

CONDITIONAL ASSEMBLY PSEUDO OPERATORS (Cont)

ORIGIN STATEMENT, 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     42200 A
```

The program is started at location 042 200 (offset octal) and the label BEGIN is assigned the offset octal value 040 200. This is the lowest address the user (programmer) should use.

```
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 STATEMENT, 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.

PSEUDO OPCODES/ASSEMBLER DIRECTIVES (Cont)

XTEXT STATEMENT, XTEXT

The XTEXT statement is used to include the contents of another file in the assembly. The form of the XTEXT statement is:

```
    XTEXT   <fname>
```

When the assembler encounters the XTEXT pseudo operation, it locates the specified file <fname>. <Fname> must reside upon a disk device and should contain assembly language statements. Note that it may not contain an END statement nor another XTEXT statement. The statements in <fname> are assembled into the program where the XTEXT statement was encountered. The XTEXT statement itself is normally listed, but the included statements from <fname> normally are not. The C listing control option is provided to cause them to be listed (see LON and LOF pseudo operations).

The file specification <fname> may specify a device code and an extension. If no extension is specified, ASM assumes the extension .ACM. The only device codes that you may specify are codes for disk devices which have been mounted. If no device is specified and XTEXT default devices were specified on the command line, ASM will search those default devices for the file in the order that they were specified. If it does not find the named file on the default devices, or if no default devices were specified, ASM will search for the file on the device where the main program resides. If it still cannot find the named file, ASM will search for the file on device SY0:. If the file still cannot be located, ASM will flag the XTEXT statement with a "U" error.

The XTEXT statement is normally used to include files containing symbol definitions and commonly-used subroutines. For example, Heath provides a file intended to be used with XTEXT, "HDOS.ACM." HDOS.ACM contains symbolic definitions for various operating system function requests. For example, the symbol .EXIT is defined to have the value of 0 (zero). A program including the file HDOS.ACM can use this symbol in generating system requests. This is not only self-documenting, but should a future system revision change the system function codes, the programmer can convert over by simply changing the definitions in HDOS.ACM, and reassembling all of his programs, since they all make use of the same definition file, HDOS.ACM.

You can also use XTEXT to include commonly-used assembly language subroutines into a program. In this way a programmer can avoid having to rewrite and redebug the same routine for each of his programs. An assembly language programmer will soon build an extensive library of utility subroutines, ready to be XTEXTed into any assembly language program.

PSEUDO OPCODES/ASSEMBLER DIRECTIVES (Cont)

LISTING CONTROL

ASM 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 '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 ASM processes an EJECT pseudo, the output device is instructed to move to the start of a new page during the listing.

PSEUDO OPCODES/ASSEMBLER DIRECTIVES

LISTING CONTROL (Cont)

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 were encountered.

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 in the listing. If more than three bytes are generated by a statement, new lines are generated in the listing to display these bytes. NOTE: The DB 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.

PSEUDO OPCODES/ASSEMBLER DIRECTIVES (Cont)

LISTING CONTROL (Cont)

C Lists XTEXT-included lines. When this option is enabled, all lines included via the XTEXT pseudo operator are listed.
DEFAULT MODE: XTEXT lines are not listed.

R Lists referenced labels. When this option is enabled, lines which reference labels in the operand field are included in the cross-reference table.
DEFAULT MODE: Lines which reference labels are included.

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.

NOREF

The NOREF pseudo causes references to the defined symbols to be omitted from the cross-reference listing. The form of the NOREF pseudo is:

```
    NOREF   symbol 1, symbol 2, . . . . . symbol n
```

Note that the specified symbols are included in the cross-reference table until NOREF pseudo is encountered.

ERRxx

ASM contains four conditional error pseudo operators. These are of the form:

```
    ERRZR   iexp
    ERRNZ   iexp
    ERRPL   iexp
    ERRMI   iexp
```

PSEUDO OPCODES/ASSEMBLER DIRECTIVES (Cont)

LISTING CONTROL (Cont)

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. 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   Assumes 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 warns of the mistake.

****

GENERATING THE ASSEMBLER

Before you can use the assembler, it must first be copied to your system disk. You can do this by copying the files:

```
ASM .ABS
XREF.ABS
   *.ACM
```

The file ASM.ABS is the assembler, and XREF.ABS is the cross-reference table generator. Both files are necessary before you can run the assembler. The files with filename extensions of .ACM are Assembler Common files. The use of these files will be discussed later.

USING THE ASSEMBLER

In order to use the assembler, you must prepare source code using a text editor such as EDIT.ABS. To get you started, Heath has prepared some short assembly language programs which may be found in Appendix A.

When the source program is ready, type ASM in response to the HDOS system prompt. HDOS interprets this command as RUN SY0:ASM.ABS. If the assembler is on SY1:, then type RUN^SY1:ASM. [Note: The ^ mark indicates to make a space.] In either case, the assembler will type:

```
HDOS Assembler Issue #104.00.00
*
```

Note that the issue number may be different, but an issue number will be shown. The asterisk (*) is the Assembler's prompt, asking you to enter a command line in the form:

```
<binary fname>,<listing fname>,<XREF temp fname>=
<source fname>,<XTEXT devices>[/SWITCHa . . . /SWITCHn]
```

The <binary fname> specification tells ASM where to put the generated binary program. The default extension is .ABS. If you do not wish to generate a binary file, omit the filename, but not the following comma.

The <listing fname> specification tells ASM where to put the assembly listing. The default extension is .LST. If you specify no listing file, ASM will not generate one. In that case, any program statements that contain errors will be listed on the system console.

The <XREF temp fname> specification tells ASM where to store a temporary file which will contain information used in generating the cross-reference table. The default extension is .TMP. If no .TMP file is specified, ASM will not generate a cross-reference table. The temporary file is automatically deleted after the cross-reference table has been listed.

GENERATING THE ASSEMBLER (Cont)

USING THE ASSEMBLER (Cont)

The <source fname> specification tells ASM which file contains the assembly language source program. The default extension is .ASM. You must specify this file; it cannot be omitted. The device specified must be a mounted disk drive.

The <XTEXT devices> specification tells ASM where to search for XTEXT files. You may specify from 0 to 5 device names, each separated by a comma. The devices must be disk drives, and the disks must have been mounted. For further information, see the paragraphs on XTEXT.

SWITCHES

There are several switches that you may specify at the end of the command line. These switches are all optional, and you can combine any number of them. The legal switches are:

/LARGE

This switch tells ASM that the program you wish to assemble is large, and it should use all the available memory. Normally, when assembling, ASM speeds itself up by letting the operating system use a portion of RAM. However, if your program is so large that the assembler runs out of RAM, you will have to assemble using the /LARGE switch. This switch causes ASM to use all the available RAM for itself, with a slightly slower assembly as a result. For systems with only 32k RAM, ASM will automatically use all of available memory; specifying /LARGE will have no effect.

/ERR

This switch causes ASM to write all program lines with errors in them to the console. Of course, the lines are also written in the normal fashion to the listing file. If no listing file is specified, error lines will automatically be written to the console regardless of the /ERR switch.

GENERATING THE ASSEMBLER (Cont)

SWITCHES (Cont)

/PAGE:nn

ASM writes the source listing file formatted into pages so the program can be listed neatly on a printer or a hard-copy terminal. The /PAGE switch tells the assembler how many lines are to appear on a page. Note that this is not the size of the page itself, since you will want to leave several lines to form a gap between the pages. Thus, for the standard page size of 66 lines, a specification of /PAGE:60 is about right. This is the default value, so only users with non-standard paper (i.e., other than 8.5 x 11 inches) size need to specify /PAGE.

/WIDE

The /WIDE switch informs the cross-reference program that the listing is to be printed on 132-column paper. Default paper width is 80 columns.

/FORM:nn

When the assembler is told to start a new page for the listing file, it writes an ASCII form-feed character into the listing file. This causes an eject to a new page. If your hard-copy device will not respond to a form-feed in this manner, you can use the /FORM:nn switch to have the assembler generate the proper number of line-feeds to cause the paper to eject. The "nn" field is the size of a page (or "form") for your hard-copy device. This must be larger than the specified /PAGE:nn value.

The standard size for most computer forms is 66 lines per page; thus, /FORM:66 should be specified. If, for example, you had paper that held 40 lines per page, and you wished to print only the top half of each page, you could specify /FORM:40/PAGE:20. This tells ASM that you want to print 20 lines per page, and that each page is 40 lines long. When the /FORM:nn switch is specified, the assembler writes the proper number of carriage return line-feeds to the listing file instead of the form-feed character.

GENERATING THE ASSEMBLER (Cont)

SWITCHES (Cont)

/LON:ccc

The /LON switch is used to override the listing options specified (via the LON and LOF pseudo instructions) in the assembly language code. The "ccc" switch represents one or more listing options, discussed in the paragraph on the LON pseudo instruction. A listing option selected by the /LON switch cannot be deselected by a LOF pseudo instruction for the program. Note: LOF is not a switch; it is a pseudo op.

/LOF:ccc

The /LOF switch is used to override the listing options specified (via the LOF and LOF pseudo instructions) in the assembly language source code. "ccc" represents one or more listing options, discussed in the paragraph on the LOF pseudo instruction. A listing option deselected by the /LOF switch cannot be selected by a /LON pseudo instruction in the program.

COMMAND LINE EXAMPLES:

This section shows several example assembly command lines with a brief discussion of each. These lines all show assembly of a sample program, DEMO.ASM.

"*" 'DEMO,DEMO=DEMO<RTN>'

This command causes the file SY0:DEMO.ASM to be assembled, with the listing file written to SY0:DEMO.LST and the binary file written to SY0:DEMO.ABS. Note that form-feed characters will be used to separate the pages of the listing file.

"*" 'DEMO.DEMO,TEMP=DEMO,SY2:<RTN>'

This command causes the file SY0:DEMO.ASM to be assembled, with the binary file written to SY0:DEMO.ABS. The listing file will be written to file SY0:DEMO.LST, and will include a cross-reference table. The file SY0:TEMP.TMP will be used as a temporary work file for generating the cross-reference table. Device SY2: will be the first device to be searched for any files given on XTEXT lines in the source file which do not specify a device.

GENERATING THE ASSEMBLER (Cont)

COMMAND LINE EXAMPLES (Cont)

"*" 'DEMO.XXX,TT:=DEMO.ASM/FORM:66<RTN>'

This command causes the file SY0:DEMO.ASM to be assembled, with the listing file written directly to the console terminal device, TT:. The binary file will be written to file SY0:DEMO.XXX. This example assumes that the console terminal is a Decwriter II, without the form-feed option. Thus, the /FORM switch was specified so the assembler would space the paper correctly.

"*" ',LP:=DEMO/LOF:L<RTN>'

This command causes the SY0:DEMO.ASM file to be assembled, with the listing file written directly to the line printer, device LP:. Since no /FORM switch was specified, and the /PAGE switch was defaulted to /PAGE:60, the assembler will write pages of 60 lines (or less) to the line printer, separated by form-feed characters. The user in this case wanted a listing of just the errors in his program, without listing all the correct statements. His use of the /LOF:L switch specified that no lines were to be listed. Since lines containing errors are always listed on the listing file, the result will be a listing on the printer showing only lines with errors.

"*" '=DEMO<RTN>'

This final example shows the user assembling the program SY0:DEMO.ASM, and producing no binary or listing files. This form is useful to check a program for assembly errors since, in the absence of a listing file, all assembly errors are printed on the console. Note that no binary file will be generated.

ERRORS

All errors detected by the Heath Assembly Language assembler 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 is placed in column 3.

GENERATING THE ASSEMBLER (Cont)

ERRORS (Cont)

CHARACTER	ERROR DESCRIPTION
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, such as a register pair instruction which refers to a single register.
D	Label is double-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 8-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 instruction.
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. Or too many characters were specified 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 ASM pseudo-op instruction code set. Check for spelling errors or for op-codes used from some other microprocessor instruction sets.
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 then attempt to 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 DBUG. However, this does not result in a correct listing.

****

APPENDIX 11-A: - ASSEMBLY LANGUAGE INTERFACE

INTRODUCTION

The HDOS Operating System offers a powerful and yet simple interface to assembly language programs. This section discusses the fundamental system commands necessary to execute a simple assembly language program. The advanced features and facilities of HDOS will be discussed in the HDOS System Programmers' Guide, Chapter 13.

HDOS provides what is called the "environment" for an assembly language program. It loads the program into memory, sets up the stack, handles console and disk device I/O, and provides other services for the program. In return, a programmer must always remember that his program is not the only one running in the computer -- the HDOS program is also running in the same machine. A programmer must:

* Be careful not to write into memory locations reserved for HDOS.

* Be sure his program does not destroy the program stack by loading the stack pointer.

* Be sure his program does not turn off interrupts via the DI instruction (except for very short periods of time).

Finally, it is important that assembly language programs use the support and facilities of HDOS, rather than "doing it themselves." Using HDOS whenever possible serves two functions: first, it makes the program much more useful and flexible. For example, if your program uses the HDOS console driver rather than communicating directly to the console itself via IN and OUT instructions, your program automatically takes advantage of the features of the HDOS console system (i.e., CTRL-S, CTRL-O, CTRL-U, RUBOUT, etc) without any extra programming effort from you. Later, if a new version of HDOS supports new devices and/or new features, your program will automatically be able to take advantage of any new feature without having to be modified.

The second reason for using HDOS functions is system compatability. As mentioned above, new releases of HDOS will be made available periodically. These new versions will fix known bugs, support new devices, and contain powerful new features. Programs which properly use HDOS functions will be able to run under the new version of HDOS after being reassembled. Programs that "do it themselves" may fail to work under new HDOS releases.

NOTE: The symbol [^] indicates to type a space.
The symbol [>] indicates the HDOS system prompt.
The symbol [*] indicates the Assembler prompt.

APPENDIX 11-A: - ASSEMBLY LANGUAGE INTERFACE (Cont)

WRITING YOUR PROGRAM

In order to successfully run your assembly language program under HDOS, you must follow the simple format shown in Figure 1. Your program must start with the three lines:

```
    TITLE   "some descriptive title"
    XTEXT   HDOS
    ORG     USERFWA
```

The TITLE statement causes an appropriate title to be printed on the assembly listing. The title you use is not important as long as it is meaningful to you. The XTEXT statement prepares the assembler for the HDOS commands you will be including in your program. These are discussed later in this appendix. Finally, the ORG statement tells the assembler to assemble your program into the user memory area.

You may write your program after these three lines; however, the last line in the program must be:

```
    END     xxx
```

where xxx is a label in your program. When you run your program, via the RUN command, execution will begin at the label specified in the END statement.

```
+-----------------------------------------------------------------+
|         TITLE   "some meaningful title"                         |
|         XTEXT   HDOS                                            |
|         ORG     USERFWA                                         |
|                                                                 |
| XXX     (first line of meaningful code)                         |
|         (your program goes here, see Figures 2, 3, and 4        |
|         for examples)                                           |
|                                                                 |
|         END     xxx                                             |
+-----------------------------------------------------------------+
```

Figure 1

ASSEMBLING YOUR PROGRAM

The first thing you must do to run an assembly language program is to assemble it. This process translates the source code language statements into the 8080A binary object codes. A sample source-coded program, "DEMO.ASM," is shown beginning on page 11-70. This listing should be entered using an editor. Once you have this program as a source file, you can then assemble it. In the HDOS command mode, type:

```
">"'RUN^ASM<RTN>'
"*"'DEMO,DEMO=DEMO<RTN>'
```

APPENDIX 11-A: - ASSEMBLY LANGUAGE INTERFACE (Cont)

ASSEMBLING YOUR PROGRAM (Cont)

This command tells the assembler that you want to assemble the file SY0:DEMO.ASM, producing a listing file called SY0:DEMO.LST, and producing a binary file called SY0:DEMO.ABS. It is this binary file that contains the executable program. If you have a hard-copy device, such as a line printer, you can copy the file DEMO.LST onto that device for reference during the remainder of this discussion. If you do not have a hardcopy device, you can refer to the listing of the file DEMO.ASM at the end of this appendix.

Note that the .ASM, .LST, and .ABS extensions are "defaults" provided by ASM. The assembler will use any specified extensions. Since ASM makes use of HDOS facilities for I/O, ASM is also device independent. For example, if you are assembling a program and want to produce the listing output on your "AT:" device, you need not write the listing file to the disk, copy it to "AT:" and then delete it. Instead, type:

```
"*"'DEMO,AT:=DEMO<RTN>'
```

This will cause the listing to be written directly to the "AT:" device.

EXECUTING YOUR PROGRAM

You must specify the starting address, or entry point, of your program in the END statement. Thus, in the program DEMO.ASM, the END statement says that execution is to start at the label ENTRY. When you type:

```
">"'RUN^DEMO<RTN>'
```

HDOS will load the program into memory and start executing it at the label ENTRY.

RETURNING TO HDOS

When your program has finished executing, it must return control to HDOS so you can continue to use the operating system. Your program can do an orderly return to HDOS by executing the two instructions:

```
    XRA     A
    SCALL   .EXIT
```

This will cause control to return to HDOS. The SCALL .EXIT instruction will be the last one your program will execute.

APPENDIX 11-A: - ASSEMBLY LANGUAGE INTERFACE (Cont)

RETURNING TO HDOS (Cont)

The SCALL is a special HDOS assembler operation that generates a special two-byte call to the HDOS Operating System. The symbol .EXIT indicates the particular type of request you want to make. In this case, you are telling HDOS that you are done executing.

Another way to return control to HDOS is to process CTRL-Cs within your program. In your program initialization, set up CTRL-C processing as follows:

```
          .
          .
          .
      LXI  H,EXIT
      MVI  A,003
      SCALL  .CTLC
          .
          .
          .
```

The end of your program will have the exit routine:

```
          .
          .
          .
    EXIT XRA A
    SCALL .EXIT
```

A CTRL-C entered while your program is running will cause a return to HDOS.

If you have not dismounted or reset your system, typing CTRL-Z twice will return you to HDOS immediately. However, if your program has a bug, and cannot respond to either CTRL-C or CTRL-Z, you should reboot the system. This will restart the system. You can then run your program under DBUG and isolate the problem in a controlled environment.

MEMORY USAGE

HDOS uses memory locations both below and above your program. It is important that HDOS should know how much of the user memory area, starting at address 042200 that your program will be using. In order to be as fast as possible, HDOS will use some of the RAM area (that part directly below the resident HDOS code) for a work area, if the running user program is not using it. Thus, if you are not going to use that RAM, HDOS should be informed so that it can use the area. If you are going to use that RAM, HDOS should be informed so that it will not use the same area for itself.

APPENDIX 11-A: - ASSEMBLY LANGUAGE INTERFACE (Cont)

MEMORY USAGE (Cont)

When you type the command:

```
">"'RUN^<fname><RTN>'
```

HDOS automatically computes the size of your program as it was assembled. This means that your program must not write into any memory location that you did not declare during the assembly, using a DB, DW, or DS statement. For example, if your program needs a 500-byte memory area, you should not write your program in the form:

```
            .       .
            .       .
            .       .
    WORK    EQU     *          500 BYTE WORK AREA STARTS HERE
            END     ENTRY
```

-- Example of how not to write your program --

and then use the 500 bytes starting at the label WORK. In this case, HDOS would think that your program ended at the label, WORK. HDOS would have no way of knowing that you had planned to use 500 more bytes. Instead, you should code the program as follows:

```
            .       .
            .       .
            .       .
    WORK    DS      500        500 BYTE WORK AREA
            END     ENTRY
```

-- Example of one correct way to write your program --

In this case, HDOS will know that you will be using the 500 bytes at WORK because you declared them in the DS statement.

TYPING LINES AND CHARACTERS

HDOS provides two commands for writing to the console terminal. These are .PRINT and .SCOUT.

.PRINT

The .PRINT SCALL is used to print a line of text on the system console. Before you issue the .PRINT SCALL, you must load the address of the first byte of the line to be printed in the H and L registers.

APPENDIX 11-A: - ASSEMBLY LANGUAGE INTERFACE (Cont)

TYPING LINES AND CHARACTERS (Cont)

.PRINT (Cont)

For example:

```
            LXI     H,LINE
            SCALL   .PRINT          PRINT THE MESSAGE
             .       .
             .       .
    LINE    DB      12Q,'HI THER','E'+200Q
```

would cause the message
"HI THERE"
to be printed on the system console

You have probably noticed that the DB statement in the above example contains more than just the character string 'HI THERE'. The first of these additions is the 12Q. This tells the assembler to start the message with the ASCII character 012 OCTAL. This is the ASCII "New Line" character. Instead of using the ASCII Carriage Return and Line Feed characters, HDOS uses the "New Line" character. (NOTE: The "New Line" has the same octal code as the Line Feed; since HDOS does not allow Line Feed characters, there is no confusion.) The "New Line" character causes a new line to be started on the output device. The rationale behind the use of "New Line" instead of Carriage Return-Line Feed is beyond the scope of this manual. Suffice it to say that the use of "New Line" gives a device-independent way to cause a new line to be started. The Carriage Return character should not be used; the Line Feed character will be interpreted as a "New Line," since both are represented by 12Q.

The other item to note about the DB statement is the expression "'E'+200Q". The .PRINT command prints the characters whose address is in the H and L registers until it prints a character with the parity (200Q) bit set. This character is the last one printed. Thus, in the example the expression "'E'+200Q" was used to set the high-order bit on the last 'E' in the message so HDOS would stop typing at that point.

.SCOUT

Use the .SCOUT to type a single character on the console device. The character in the A register is printed on the console terminal. For example:

```
            MVI     A,'X'
            SCALL   .SCOUT              PRINT THE CHARACTER 'X'
```

The high-order bit (parity bit) is ignored by .SCOUT.

APPENDIX 11-A: - ASSEMBLY LANGUAGE INTERFACE (Cont)

READING FROM THE CONSOLE

HDOS provides the .SCIN command for reading characters from the console terminal and the one command .CONSL to control character echoing, backspace, and erase-line handling.

.SCIN

The .SCIN command is used to read a single character from the console device. If the 8080 "carry" flag is set after the SCALL instruction, it means that no character has been typed yet. If the carry flag is clear, then a character has been read and is in the A register. It does not matter if the carry flag is set or clear when you execute the SCALL .SCIN. For example:

```
READ        SCALL   .SCIN           READ A CHARACTER, IF ANY
            JC      READ            NO CHARACTER ENTERED, YET
            STA     CHAR            STORE CHARACTER READ IN MEMORY
```

.CONSL

The .CONSL command is used to set the mode of console input. There are two modes of input: line mode, and character mode.

When you are inputting in line mode, HDOS saves up the typed characters until you type a <RTN>. This is done so HDOS can handle RUBOUT (character delete) and CTRL-U (line delete) functions. If HDOS were to give you the characters one by one as they were typed, it wouldn't be able to 'take them away again' if CTRL-U were typed. By saving them all up until you press <RTN>, HDOS can handle any DELETEs and CTRL-Us that are typed. For example, if you were to type the four keys Y, E, S, and <RTN> while your program was executing the example shown above, it would not receive any characters until you pressed the <RTN>. The next four .SCIN commands would each return with one of the characters. The <RTN> key gives the 012Q, "New Line" character code. Thus, the four values read when you type YES <RTN) are 131Q (Y), 105Q (E), 123Q (S), and 012Q (RTN).

Line mode is very useful when you wish to input a line from the console, since HDOS provides the DELETE and CTRL-U functions for you automatically. For programs that need to read each character immediately after it is typed, there is 'character mode'. Inputting in "Character Mode" causes the typed character to be passed to your program immediately. If the user types RUBOUT or DELETE, the RUBOUT code (177Q) is passed to your program. If the user types CTRL-U, the CTRL-U code (025Q) is passed to your program. "Character Mode" is more

APPENDIX 11-A: - ASSEMBLY LANGUAGE INTERFACE (Cont)

READING FROM THE CONSOLE (Cont)

.CONSL (Cont)

flexible than "Line Mode," but it requires your program to handle the RUBOUT and CTRL-U keys.

The .CONSL command also allows you to turn character echoing on and off. If echoing is turned on, each time the user strikes a character, it is typed on the console automatically by HDOS. If echoing is turned off, the character is not typed on the console. If you wish the character to be visible, your program must type it itself, via the .SCOUT command.

To use the .CONSL command, code the following lines:

```
            XRA     A
            MVI     B,xxx           "xxx" value discussed below
            MVI     C,201Q
            SCALL   .CONSL
```

where "xxx" is

```
            000Q FOR LINE MODE WITH ECHO
            001Q FOR CHARACTER MODE WITH ECHO
            200Q FOR LINE MODE WITHOUT ECHO
            201Q FOR CHARACTER MODE WITHOUT ECHO
```

The default mode of HDOS console input is "line mode, with echo". You only need to use the SCALL .CONSL command if you wish some other mode of operation. You can change modes of operation as often as you like.

```
  +-----------------------------------------------------------------+
  |                              NOTE                               |
  |  The HDOS system must be configured to accept tabs in order     |
  |  to run a demonstration program.                                |
  |                                                                 |
  |  For details, see the SET command in Chapter Three, page        |
  |  3-19 of this HDOS manual.                                      |
  |                                                                 |
  +-----------------------------------------------------------------+
**********************************************************************
```

APPENDIX 11-B: - SAMPLE SOURCE CODE LISTING

[1] SAMPLE LISTING 1:

```
042.200             00002        XTEXT   HDOS
                    00020        ORG     USERFWA
                    00021
                    00023 ***    DEMO.ASM - HEATH HDOS ASSEMBLY LANGUAGE
                    00024 *
                    00025 *      DEMO IS A SHORT AND SIMPLE PROGRAM USED
                    00026 *      TO DEMONSTRATE THE HDOS ASSEMBLER AND
                    00027 *      THE HDOS OPERATING SYSTEM
                    00028 *
                    00029 *      THIS PROGRAM SIMPLY PRINTS TWO CODED
                    00030 *      LINES ON THE SYSTEM CONSOLE TERMINAL.
                    00031
042.200 041 221 042 00032  ENTRY LXI      H,MESA  (HL)=ADDRESS OF 1ST MSG
042.203 377 003     00033        SCALL    .PRINT  PRINT FIRS MESSAGE
O42.205 041 250 042 00034        LXI      H,MESB  (HL)=ADDRESS OF 2ND MSG
042.210 377 003     00035        SCALL    .PRINT  PRINT 2ND MESSAGE
                    00036
                    00037 *      SEND A BELL TO THE TERMINAL
                    00038
042.212 076 007     00039        MVI      A,07Q   (A)=ASCII BELL
042.214 377 002     00040        SCALL    .SCOUT  RING TERMINAL'S BELL
                    00041
                    00042 *      RETURN CONTROL TO HDOS OPERATING SYSTEM
                    00043
042.216             00044        XRA      A       EXIT TO OPERATING SYSTEM
042.217 377 000     00045        SCALL    .EXIT
                    00046
                    00047
                    00048 *      MESSAGES FOR .PRINT SCAL'S
                    00049
042.221 012 110 111 00050 MESA   DB   12Q,'HI THERE,SPORTS FANS','!'+200Q
                    00051        LON  G LIST THE BYTES OF THE NEXT MESSAGE
042.250 012 131 117 00052 MESB   DB   12Q,'YOUR SYSTEM WORKS FINE','!'+200Q
        125 122 040
        123 131 123
        124 105 115
        040 127 117
        122 113 123
        040 106 111
        116 105 241
                    00053

                    00054        END  ENTRY    START EXECUTING AT 'ENTRY'
                                               LABEL

00054 Statements Assembled
32420 Bytes Free
No Errors Detected
```

APPENDIX 11-B: - SAMPLE SOURCE CODE LISTING (Cont)

[1] SAMPLE LISTING 1: (Cont)

```
SYMBOL TABLE
 .CONSL  000006    .EXIT   000000    .PRINT  000003    .SCIN  000001
 .SCOUT  000002    ENTRY   042200    MESA    042221    MESB   042250
 STACK   042200    USERFWA 042200


CROSS REFERENCE TABLE

 .CONSL  000006   14E
 .EXIT   000000   10E   45
 .PRINT  000003   13E   33   35
 .SCIN   000001   11E
 .SCOUT  000002   12E   40
 ENTRY   042200   32L   54
 MESA    042221   32    50L
 MESB    042250   34    52L
 STACK   042200   18E

 USERFWA 042200   19E   20

39566 Bytes Free
```

APPENDIX 11-B: - SAMPLE SOURCE CODE LISTING (Cont)

[2] SAMPLE LISTING 2:

```
042.200             00002       XTEXT  HDOS
042.200             00020       ORG    USERFWA
                    00021
                    00023
                    00024 ***   DEMO2.ASM-CONSOLE INPUT DEMO, IN LINE MODE
                    00025 *
                    00026 *     THIS IS A SIMPLE DEMONSTRATION PROGRAM
                    00027 *     THAT INPUTS LINES FROM THE CONSOLE,
                    00028 *     AND TYPES THEM BACK AGAIN.
                    00029 *
                    00030 *     IF THE LAST LINE YOU ENTERED CONTAINED A
                    00031 *     ('.') THEN DEMO2 EXITS TO HDOS AFTER
                    00032 *     TYPING THE LINE.
                    00033
                    00034
                    00035 ***   TO RUN THIS PROGRAM, TYPE THE FOLLOWING:
                    00036 *     (DO NOT TYPE COMMENTS IN PARENTHESIS)
                    00037 *
                    00038 *     >RUN ASM
                    00037 *     *DEMO2,TT:=DEMO2  (WRITES LISTING
                    00038 *                        TO CONSOLE)
                    00039 *     >RUN DEMO2
                    00040 *     HI, I'M DEMO2!    (DEMO2 TYPES THIS)
                    00041 *     ABCD              (YOU TYPE THIS)
                    00042 *     ABCD              (DEMO2 TYPES THIS)
                    00043 *     IS ANYONE THERE?  (YOU TYPE THIS)
                    00044 *     IS ANYONE THERE?  (DEMO2 TYPES THIS)
                    00045 *     BYE BYE.          (YOU TYPE THIS)
                    00046 *     BYE BYE.          (DEMO2 TYPES THIS)
                    00047 *     >                 (DEMO2 EXITS TO HDOS)
042.200 041 236 042 00048 ENTRY LXI    H,DCMOA     EXECUTION STARTS HERE
042.203 377 003     00049       SCALL
                    00050
                    00051 *     LOOP ECHOING LINES
                    00052
042.205 377 001     00053 ECHO  SCALL  .SCIN
042.207 332 205 042 00054       JC     ECHO    NO CHARACTER YET
042.212 376 056     00055       CPI    '.'
042.214 302 222 042 00056       JNE    ECHO1   NOT PERIOD CHARACTER
042.217 062 256 042 00057       STA    ENDFLAG MAKE ENDFLAG NON-ZERO
                    00058                      (A '.', IN FACT)
042.222 377 002     00059 ECHO1 SCALL  .SCOUT  TYPE CHARACTER BACK
042.224 072 256 042 00060       LDA    ENDFLAG
042.227 247         00061       ANA    A
042.230 312 205 042 00062       JZ     ECHO    STILL MORE TO GO
                    00063
                    00064 *     HAVE SEEN '.' WILL RETURN TO HDOS
```

APPENDIX 11-B: - SAMPLE SOURCE CODE LISTING (Cont)

[2] SAMPLE LISTING 2: (Cont)

```
                    00065
042.233             00066       XRA    A
042.234 377 000     00067       SCALL  .EXIT  RETURN TO HDOS
                    00068
042.236 012 110 111 00069 DCMOA DB     12Q.'HI,I'M DEMO2!',212Q
042.256 000         00070 ENDFLAG DB   0      <>0 IF TO EXIT
                    00071
042.257 000         00072       END    ENTRY


00072 Statements Assembled
32401 Bytes Free
No Errors Detected


SYMBOL TABLE

.CONSL  000006   .EXIT  000000   .PRINT  000003   .SCIN   000001
.SCOUT  000002   DEMOA  042236   ECHO    042205   ECHO1   042222
ENDFLAG 042256   ENTRY  042200   STACK   042200   USERFWA 042200


DEMO2.ASM -- CONSOLE READ DEMO, LINE MODE       HEATH XREF #104.06.00
CROSS REFERENCE TABLE                           22-SEP-80  PAGE 4

.CONSL    000006     14E
.EXIT     000000     10E     67
.PRINT    000003     13E     50
.SCIN     000001     11E     54
.SCOUT    000002     12E     59
DEMOA     042236     49      69L
ECHO      042205     54L     55     62
ECHO1     042222     57      59L
ENDFLAG   042256     58      60     70L
ENTRY     042200     49L     72
STACK     042200     18E
USERFWA   042200     19E     20

32513 Bytes Free
```

APPENDIX 11-B: - SAMPLE SOURCE CODE LISTING (Cont)

NOTES ON SAMPLE 2:

Note that although this program DEMO2.ASM appears to be written to echo each character after it is typed, actually it echoes each line after the RETURN has been typed. This is because the program reads characters in line mode. HDOS holds the characters until you press the RETURN key, and then supplies them to the DEMO2 program. Thus, each line typed to this program appears twice: once when HDOS echoes it as it is being typed, and once when DEMO2.ASM types it.

NOTES ON SAMPLE 3:

Note that the program DEMO3.ASM is identical to DEMO2.ASM, except that this program inputs in character mode, rather than line mode. This causes a big difference in the response the program makes when you type input to it. DEMO3.ASM echoes each character immediately after it is typed. This causes each character to be printed twice on the screen: once when HDOS echoes it, and once when DEMO3.ASM types it. As an exercise, modify this program to disable the automatic echoing which is done by HDOS.

APPENDIX 11-B: - SAMPLE SOURCE CODE LISTING (Cont)

[3] SAMPLE LISTING 3:

```
042.200             00002       XTEXT  HDOS
042.200             00020       ORG    USERFWA
                    00021
                    00023
                    00024 ***   DEMO2.ASM-CONSOLE INPUT DEMO,
                    00025 *     IN CHARACTER MODE.
                    00026 *
                    00027 *     THIS IS A SIMPLE DEMONSTRATION PROGRAM
                    00028 *     THAT INPUTS CHARACTERS FROM THE
                    00029 *     THE CONSOLE, AND TYPES THEM BACK AGAIN.
                    00030 *
                    00031 *     IF THE LAST CHARACTER YOU ENTERED CONTAINED
                    00032 *     A ('.') THEN DEMO2 EXITS TO HDOS AFTER
                    00033 *     TYPING THE LINE.
                    00034
                    00035
                    00036 ***   TO RUN THIS PROGRAM, TYPE THE FOLLOWING:
                    00037 *     (DO NOT TYPE COMMENTS IN PARENTHESIS)
                    00038 *
                    00039 *     >RUN ASM
                    00040 *     *DEMO3,TT:=DEMO3  (WRITES LISTING
                    00041 *                        TO CONSOLE)
                    00042 *     >RUN DEMO3
                    00043 *     HI, I'M DEMO3!    (DEMO3 TYPES THIS)
                    00044 *     AABBCCDD  (YOU TYPE ABCD, DEMO3 ECHOS IT)
                    00045 *     XXYY..    (YOU TYPE 'XY.',DEMO3 ECHOS IT)
                    00046 *     >                 (DEMO3 EXITS TO HDOS)
                    00047 *
042.200 041 245 042 00048 ENTRY LXI    H,DEMOA     EXECUTION STARTS HERE
042.203 377 003     00049       SCALL  .PRINT      PRINT 'HI!' MESSAGE
                    00050
                    00051 *     SETUP CHARACTER MODE.  SINCE HDOS WILL ECHO
                    00052 *     THE CHARACTERS, AND THEN DEMO3 WILL TYPE
                    00053 *     THEM.  CHARACTERS WILL BE DOUBLED ON THE
                    00054 *     SCREEN AS THEY ARE TYPED.
                    00055
042.205 257         00056       XRA    A
042.206 006 001     00057       MVI    B,0001Q  CHARACTER MODE WITH ECHO
042.210 016 201     00058       MVI    C,201Q
042.212 377 006     00059       SCALL  .CONSL
                    00060
                    00061 *     LOOP ECHOING LINES
                    00062
042.214 377 001     00063 ECHO  SCALL  .SCIN
042.216 332 214 042 00064       JC     ECHO     NO CHARACTER YET
042.221 376 056     00065       CPI    '.'
042.223 302 231 042 00066       JNE    ECHO1    NOT PERIOD CHARACTER
042.226 062 265 042 00067       STA    ENDFLAG  MAKE ENDFLAG NON-ZERO
```

APPENDIX 11-B: - SAMPLE SOURCE CODE LISTING (Cont)

[3] SAMPLE LISTING 3: (Cont)

```
                    00068                          (A '.', IN FACT)
042.231 377 002     00069 ECHO1 SCALL  .SCOUT   TYPE CHARACTER BACK
042.233 072 265 042 00070       LDA    ENDFLAG
042.236 247         00071       ANA    A
042.237 312 214 042 00072       JZ     ECHO     STILL MORE TO GO
                    00073 *     I HAVE SEEN '.'.  WILL RETURN TO HDOS
                    00074
042.242 257         00075       XRA    A
042.243 377 000     00076       SCALL  .EXIT    RETURN TO HDOS
                    00077
042.245 012 110 111 00078 DEMOA DD     12Q,'HI, I'M DEMO3!',212Q
042.265 000         00079 ENDFLAG DB   0        <>0 IF TO EXIT
                    00080
042.266 000         00081       END    ENTRY


00072 Statements Assembled
32401 Bytes Free
No Errors Detected


SYMBOL TABLE

.CONSL  000006   .EXIT  000000   .PRINT  000003   .SCIN   000001
.SCOUT  000002   DEMOA  042245   ECHO    042214   ECHO1   042231
ENDFLAG 042265   ENTRY  042200   STACK   042200   USERFWA 042200


DEMO3.ASM
CROSS REFERENCE TABLE

.CONSL    000006     14E     55
.EXIT     000000     10E     72
.PRINT    000003     13E     46
.SCIN     000001     11E     59
.SCOUT    000002     12E     64
DEMOA     042245     45      74L
ECHO      042214     59L     60     67
ECHO1     042231     62      64L
ENDFLAG   042265     63      65     75L
ENTRY     042200     45L     77
STACK     042200     18E
USERFWA   042200     19E     20

39508 Bytes Free
********************************************************************************
```

APPENDIX 11-C: - SUPPLEMENTAL REFERENCES

For supplemental information on Heath Assembly Language, refer to the following articles in REMark Magazine, the official Heath/Zenith Publication.

[1] REMark Issue 15, March 1981, Page 4
"A KISS for Assembly Programming - Article 1" - by Bob Ellerton.
3.5 pages. Reviews Assembly Language techniques and presents some neat tricks to help the beginner get started.

[2] REMark Issue 16, April 1981, Page 3
"A KISS for Assembly Programming - Article 2" - by Bob Ellerton.
2.25 pages. A review of EDIT, the HDOS Line Editor, and how it applies to writing Assembly Language programs.

[3] REMark Issue 17, May 1981, Page 4
"A KISS for Assembly Programming - Article 3" - by Bob Ellerton.
3 pages. Further refines the art of programming in assembly language.

[4] REMark Issue 18, June 1981, Page 4
"A KISS for Assembly Programming - Article 4" - by Bob Ellerton.
3.3 pages. Outlines techniques for programming in assembly language for those who use cassette tapes.

[5] REMark Issue 38, March 1983, Page 23
"ASM FOR THE NOVICE," - by Richard A. Martin. 3 Pages.
Defines ASM as a "low level" language, a way to organize and create "object code" by listing source code in a specific way so that the computer understands directly. Gives examples.

[6] REMark Issue 39, April 1983, Page 35
"GETTING STARTED WITH ASSEMBLY LANGUAGE," - by Patrick Swayne. 3.5 pages. Presents basic concepts and techniques for writing ASM source code.

[7] REMark Issue 43, August 1983, Page 37
"PROCESSING H-19/H89 SPECIAL FUNCTION KEYS WITH ASM," - by William R. Rousseau, MD. 3.0 pages. Tells how to implement the terminal's special function keys with ASM.

[8] REMark Issue 44, September 1983, Page 21
"GETTING STARTED WITH ASSEMBLY LANGUAGE," - by Patrick Swayne. 1 page.
Tips on guidelines for programming console I/O.

APPENDIX 11-C: - SUPPLEMENTAL REFERENCES (Cont)

[9] REMark Issue 46, November 1983, Page 57

"MORE ASSEMBLY LANGUAGE PROGRAMMING - HDOS," by P. John Hagan. 8.5 pages. Tells how to manipulate files and how to format output. Provides sample listing as a teaching tool.

INDEX

INDEX (Cont)

HDOS SOFTWARE REFERENCE MANUAL

HDOS DISK OPERATING SYSTEM

VERSION 3.0

CHAPTER 12

EXTENDED BENTON HARBOR BASIC

HEATH DISK OPERATING SYSTEM

SOFTWARE REFERENCE MANUAL

VERSION 3.0

 Through the years it continued to be improved by successive revisions which included 1.5, 1.6, and finally 2.0. It was entered into public domain on 19 July 1989 per letter by Jim Buszkiewicz, Managing Editor, Heath Users' Group, P.O. Box 217, Benton Harbor, MI 49022-0217 (616)982-3463. A copy of this letter is available for public inspection.

This manual is indicative of further improvements and provides for the latest revision, HDOS 3.0 and HDOS 3.02. Revision 3.0 is detailed in chapters 1, 2, and 3, while chapters 4, 5, 6, 7, 8, 13 and 14, are related to revision 3.02. Chapters 9 through 12, with minor improvements, are essentially picked up from the original HDOS 2.0 manual. Indeed, HDOS is still alive and well!

Chapter 12, Benton Harbor BASIC, was modified slightly for use in the HDOS 3.02 environment. This chapter explains all the features of BASIC and tells how to use it.

SPECIAL DISCLAIMER: The Heath Company cannot provide consultation on either the HDOS Operating System or user-developed or modified versions of Heath software products designed to operate under the HDOS Operating System. Do not refer to Heath for questions.

Instead, you are invited to direct any questions concerning the Heath Disk Operating System (HDOS) to Mr. Kirk L. Thompson, Editor "Staunch 89/8" Newsletter, P.O. Box 548, #6 West Branch Mobile Home Village, West Branch, IA 52358.

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont)

INTRODUCTION

Extended Benton Harbor BASIC (Extended BASIC) is a conversational programming language which is an adaptation of Dartmouth BASIC. (BASIC is a registered trademark of the Trustees of Dartmouth College.)

BASIC is an acronym for Beginner's 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. Extended BASIC is an interpretive language, compact enough to run in a Heath 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.

CONVENTIONS

Within this manual, the up-arrow symbol [^] will NOT be used to indicate a required space, in order to prevent confusion with 'exponentiation.'

Statements made by the computer will be set off by quotation marks ["], unless the particular situation is obvious. Similiarily, responses by the user will be set off by apostrophe marks ['].

Benton Harbor BASIC is referred to three ways in the manual: Benton Harbor BASIC, B. H. BASIC, and BASIC. All three forms refer to the Heath version of BASIC.

MANUAL SCOPE

BASIC runs on an H8/H19, H89, or Z90 Computer System, and requires a minimum of 24k bytes of random access memory.

This manual is written for the user who is already familiar with the BASIC programming language. 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 learning BASIC. If you are not familiar with BASIC, we suggest that you obtain the Heathkit Continuing Education course entitled "BASIC Programming," Model EC-1100, or the equivalent, before attempting to start programming with BASIC.

RUNNING BASIC

In order to run Extended BASIC, first copy the file "BASIC.ABS" (42 sectors) from your software distribution disk onto the system disk that

INTRODUCTION (Cont)

RUNNING BASIC (Cont)

you plan to use. If desired, you may use the HDOS 3.02 SYSCMD copy command to place the 42-sector BASIC.ABS on your working disk. If you plan to use your printer, it is necessary to load your printer driver first. Loading LP: is done in the following manner:

'Load LP:<RTN>'.

Once BASIC.ABS is present on disk, you can run BASIC by typing:

">"'RUN DVn:BASIC<RTN>' or simply: ">"'DVn:BASIC<RTN>'

"DVn:" is the device name (SY0:, SY1:, SY2:, SY3:, DK0:, DK1:, DK2:) that contains the file, BASIC.ABS. If you do not type a device name, HDOS assumes the file is on SY0:. For example:

```
">"'RUN BASIC<RTN>'
"EXTENDED BENTON HARBOR BASIC #110.00.00"
```

BASIC uses the asterisk [*] as its prompt character.

Note that the part number that your computer system prints on the screen may be different. However, some part number will be displayed.

****

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. In this expression, both the negative 38 and the positive 37 represent exponents. 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.

BASIC ARITHMETIC (Cont)

NUMERIC DATA (Cont)

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 the power of 10. The exponential form is:

XXE(+/-)NN

where E represents the algebraic statement "times ten to the power of;" XX represents up to a six-digit integer or real number; (+/-) represents plus or minus, and NN represents an integer from 0 to 38. Thus, the number is read as "XX times 10 to the plus or minus 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. [NOTE: This may be changed. See "CNTRL 1," page 12-34.] If the results do not fall within this range, the exponential format is used. BASIC automatically suppresses all leading and trailing zeroes 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 off of leading and trailing zeroes, truncation to six places of accuracy, conversion to exponential notation when necessary, and conversion to decimal notation where permitted.

INPUT NUMBER	OUTPUT NUMBER	COMMENTS
0.1	.1	Leading zero dropped
.0079	7.90000^-03	<.1 converts to exponential
0022	22	leading zeroes dropped
22.0200	22.02	trailing zeroes dropped
999999	999999	format maintained
1000000	1.00000^+06	converted to exponential
100000007	1.00000^+08	truncated to 6 places
-10.1E+2	-1010	converted to decimal format

BASIC ARITHMETIC (Cont)

BOOLEAN DATA

Boolean values are a subclass of numeric values. Values representing the positive integers from 0-65,535 (2 16-1) [In the term "16-1" the "16" is the base number, and the "-1" represents an exponent] 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 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. Any printable ASCII character (with the exception of the quotation mark itself) may appear in another 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.

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 each variable is not acceptable.

ACCEPTABLE VARIABLES	UNACCEPTABLE VARIABLES	REASON FOR UNACCEPTABILITY
C	2C	A digit cannot begin a variable.
A5	AF	A second character in a variable must be a number.
D	3	A single number is not an acceptable variable.
L2	$2	The first character of a variable must be a letter [A thru Z].

Subscripted variables, string variables, and subscripted string variables are permitted. See "Subscripted Variables," page 12-7 and "String Manipulation" on page 12-16.

BASIC ARITHMETIC (Cont)

VARIABLES (Cont)

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" on page 12-46, "PRINT" on page 12-51, and "INPUT AND LINE INPUT" on page 12-62.

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:

An (N1, ... , N8),

where A is the variable letter, n is a number (optional) 0-9, and N1 thru N8 are the integer dimensions of the variable. [In the expression N1, ... , N8, the numbers 1 thru 8 are subscripts.] Subscripted variables provide 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 scaler value of the same name. For example, all four of the following expressions are distinct variables:

A, A(N), A$, A$,(N)

When 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)
```

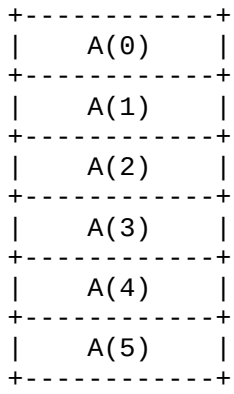
The following are graphic illustrations of simple subscripted variables. In these particular examples, a simple variable (A) is followed by one or two integer expressions in parentheses. For example:

BASIC ARITHMETIC (Cont)

SUBSCRIPTED VARIABLES (Cont)

A(I)

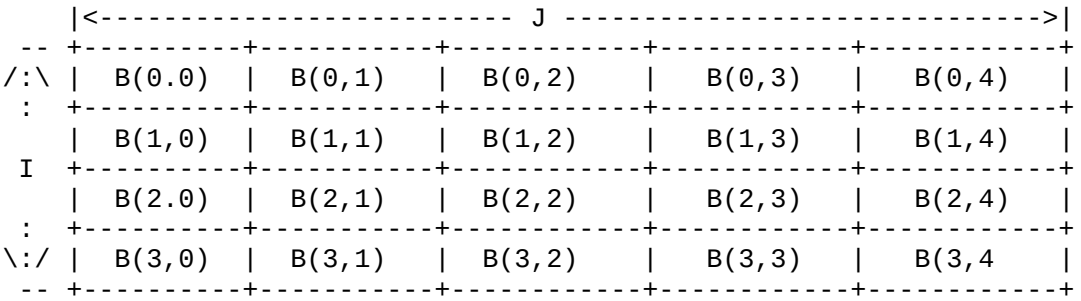
where I may assume the value of 0 to 5, allows reference to each of the six elements: A(0), A(1), A(2), A(3), A(4), and A(5). A graphic representation of this 6-element, single-dimension array is shown below. Each box represents a memory location reserved for the value of the variable of the indicated name. Often the entire array is referred to as A(.



NOTE: Subscripted variables begin at zero. Therefore, the previous example 0 (zero) to 5 defines six elements.

A two-dimensional array B(I,J) allows referral to each of the 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).



NOTE: A variable cannot be dimensioned twice in the same program unless you first clear it with the CLEAR statement.

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 12-35.

BASIC ARITHMETIC (Cont)

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:

ARITHMETIC EXPRESSIONS	BOOLEAN EXPRESSIONS	STRING EXPRESSIONS	DESCRIPTION
1.02	255	"YES"	Data
1.02 + 16	255 OR 003	"YES" + "NO"	Combined
A < B		"YES" < "NO"	Relational

A major feature of Heath's Extended Benton Harbor 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, 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 OPERATORS

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.

OPERATOR	DESCRIPTION
-(Unary)	negation
^	exponentiation
* /	multiplication division
+ -	addition subtraction

BASIC ARITHMETIC (Cont)

THE PRIORITY OF ARITHMETIC OPERATORS (Cont)

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 12-16 for details.

EXPONENTIATION

Exponentiation [^] is used to raise numeric or variable data to a power. For example:

```
A = B ^ 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 ^ 2 should be written as B*B*B. All other powers should use the ^.

MULTIPLICATION AND DIVISION

BASIC uses the asterisk [*] and the slash [/] as symbols to perform the algebra operations of multiplication and division, respectively. Both multiplication and division require numeric data as operands.

BASIC ARITHMETIC (Cont)

MULTIPLICATION AND DIVISION (Cont)

```
"*"'PRINT 2*6<RTN>'
"12"

"*"'PRINT 2/3<RTN>'
".666667"

"*"'PRINT 6/3*2<RTN>'
"4"

"*"
```

NOTE: This last expression evaluates to 4, not 1; because * 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. The following examples use the plus and minus operators:

```
"*"'PRINT 3<RTN>'
"3"

"*"'PRINT 3+5<RTN>'
"8"

"*"'PRINT 10-3<RTN>'
"7"

"*"'PRINT "HEATH" + " " + "COMPUTER"<RTN>'
"HEATH COMPUTER"
```

BASIC ARITHMETIC (Cont)

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 expression illustrates these principles:

```
2 ^ 3 ^ 2
```

The expression is evaluated from left to right:

[1] 2 ^ 3 = 8 (leftmost exponentiation has highest priority).
[2] 8 ^ 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 leftmost 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 the second highest priority)
[3] 36+6 = 42 (addition has the lowest priority; answer)

Parentheses may be nested, (inclosed 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.

BASIC ARITHMETIC (Cont)

SUMMARY (Cont)

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 has 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 a part of a string inclosed 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 false. A false is represented by a zero, and a true is represented by 65535 (2^16-1), where the expression 16-1 is an exponent. The tip-off is the ^ mark. This indicates an exponent.

NOTE: These values are chosen so when they are used as Boolean values, false is all zeroes and true is all ones.

BASIC ARITHMETIC (Cont)

RELATIONAL OPERATORS (Cont)

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.
<*	<=	A<=B	A is less than or equal to B.
>	>	A>B	A is greater than B.
>	>=	A>=B	A is greater than or equal to B.
#	<>	A<>B	A is not equal to B.

NOTES: Under the "ALGEBRAIC SYMBOL" column, two symbols are impossible to represent on the screen using the H89. Therefore, the symbol "<*" is really a "<" symbol with a "bar" that parallels the lower branch of the arrow. Similiarily, the "#" is not really a "#" symbol, but one that has two horizontal slashes but only one vertical slash.

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<RTN>'         (true)
"65535"

"*"'PRINT 4<3<RTN>'         (false)
"0"
```

Benton Harbor BASIC differs from most other implementations of BASIC 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.

BASIC ARITHMETIC (Cont)

BOOLEAN OPERATORS

OR

The operator OR performs a Boolean OR on the last 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:

```
      BASIC STATEMENT:            TRUTH TABLE
      ----------------            -----------
"*"'PRINT 132 OR 255<RTN>'     00000000 10000100    132
"255"                          00000000 11111111    255
                               -----------------
                               00000000 11111111
```

and

```
"*"'PRINT (3>2) OR (4>9)<RTN>'
"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:

```
     BASIC STATEMENT:            TRUTH TABLE
     ----------------            -----------
"*"'PRINT 132 AND 255<RTN>'    00000000 10000100   132
"132"                          00000000 11111111   255
"*"                            -----------------
                               00000000 10000100
```

and

```
"*"'PRINT (3>2) AND (9>7)<RTN>'
"65535"
```

BASIC ARITHMETIC (Cont)

BOOLEAN OPERATORS (Cont)

NOT

The NOT operator performs 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:

```
  BASIC STATEMENT:                    TRUTH TABLE
  ----------------                    -----------
"*"'PRINT NOT O<RTN>'              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, formfeed, and TAB 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$<RTN>'
"HI"
```

NOTE: The string variable B$ is separate and distinct from the variable B.

STRING MANIPULATION (Cont)

STRING VARIABLES (Cont)

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 12-7.

The same variable may 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 legal because 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 that numerical arrays are defined.

STRING OPERATORS

Extended B. H. 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 length of a concatenated string is 255 characters. For example:

```
"*"'PRINT "THE HEATH" + "COMPUTER"<RTN>'
"THE HEATH COMPUTER"
```

STRING MANIPULATION (Cont)

RELATIONAL OPERATORS FOR STRINGS

Relational operators, when applied to strings, indicate alphabetical 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"<RTN>'
"65536"                    (The relation shown is true.)

"*"'PRINT "ABC" >"ABCD"<RTN>'
"0"                        (The relation shown is false. "ABC" is
                              less than "ABCD".)
```

NOTE: In any string comparison, trailing blanks are not ignored. For example:

```
"*"'PRINT "CDE" = "CDE " <RTN>'
"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.

THE COMMAND MODE (Cont)

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 <RTN> is received. For example:

```
"*"'10 PRINT' '"THIS IS A COMPUTER"<RTN>'
```

is not executed when it is entered at the console terminal. However, the statement:

```
"*"'PRINT "THIS IS THE HEATH COMPUTER"<RTN>'
```

After the RETURN key is typed, is immediately executed as:

```
"THIS IS THE HEATH COMPUTER"
```

The command mode of operation is useful in 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.

If you use STOP in this manner, an error message will be printed. This is a normal response and not a programming error on your part. Once BASIC encounters a STOP statement, the program halts. You can examine and change data values using the command mode. The statement:

```
'CONTINUE<RTN>'
```

is used to continue the 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.

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<RTN>'
"2"
"3"
"*"
```

THE COMMAND MODE (Cont)

USING THE COMMAND MODE FOR STATEMENT EXECUTION (Cont)

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<RTN>'
"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 "Illegal Usage" 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 12-23, and "Statements Valid in the Command or Program Mode" on page 12-29.

****

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 <RTN> (a non-printing character). A D12TA statement cannot share a line with other statements. (See page 12-55 "Read and Data Statements" for details.)

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 65,534. It is good programming practice to number lines in increments of 5 or 10 to allow for insertion of forgotten or additional statements.

The length of a BASIC statement must not exceed one line. There is no method to continue a statement to the 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, while:
'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 at the beginning of a line.

Each time you type a statement with a line number, BASIC performs some simple syntactical checks before inserting the line into your program. BASIC checks to see if all the keywords are spelled correctly, and translates them to upper case. It makes sure that all function calls are immediately followed by an open parenthesis "(". BASIC makes several other checks of the line to check for simple syntax errors. If the line is determined to be incorrect, the message:

SYNTAX ERROR

will be displayed, and the line will not be inserted into your program. Note that this preliminary syntax check will not detect all possible errors; BASIC may accept the line when you type it, and then detect an error later when you execute your program.

BASIC STATEMENTS (Cont)

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 that they can appear within a program or be typed directly in the command mode and be 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 could cause the SIN of 3 to be printed by typing:

"*"'PRINT SIN(6/2)<RTN>'

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" on page 12-23, "Statements Valid in the Command or Program Mode" on page 12-29, and "Program Mode Statements," on page 12-61.

To simplify some practical descriptions in these sections and those following, the notations below are used to describe valid 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 in 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.

[4] "linnum" indicates a line number. This must be an unsigned decimal number or the expression LNO (iexp). See the discussion of the LNO function for details.

[5] "sep" indicates a separator. Separators such as the comma and the semicolon are used to delineate certain portions of BASIC statements.

[6] "[]" brackets indicate optional portions of a statement, depending on the exact function desired.

BASIC STATEMENTS (Cont)

STATEMENT TYPES (Cont)

[7] "var" indicates a variable. This may be a numeric or string variable, depending upon the example.

[8] "name" indicates a string used to identify a date, a program, or a language record.

[9] "fname" indicates an HDOS file descriptor (filename). A filename descriptor may include a device specification and a filename and extension. The device specification and extension may be omitted, in which case BASIC will supply a default.

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 an "Illegal Usage" error message.

BUILD

This statement is used to insert or replace many program lines. The form of the BUILD statement is:

```
"*"'BUILD iexp1, iexp2<RTN>'  (where iexp1 = Starting line number of
                               build sequence.)
                              (where iexp2 = Increment by nn lines)
```

When BUILD is executed, the initial line number iexp 1 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. CTRL-C terminates BUILD.

```
"*"'BUILD 100, 10<RTN>'
"*"'100 PRINT "LINE 100"<RTN>'
"*"'110 PRINT "LINE 110"<RTN>'
"*"'120 PRINT "LINE 120"<RTN>'
"*"'130 CTRL-C'                (CTRL-C is typed here)
"*"'LIST<RTN>'
"100 PRINT "LINE 100"'
"110 PRINT "LINE 110"'
"120 PRINT "LINE 120"'
```

BASIC STATEMENTS (Cont)

BUILD (Cont)

BASIC performs a preliminary syntax check on lines entered via BUILD. Should an error be detected, BUILD will give an error message. For example:

```
"*"'BUILD 10,10<RTN>'
"*"'10 PRINT "LINE 10"<RTN>'
"*"'20 PRANT "LINE 20"<RTN>'        (Note the error)
"SYNTAX ERROR"
"*"'20 PRINT "LINE 20"<RTN>'        (Reenter line 20)
"*"'30'
```

BYE

The BYE command is used to terminate BASIC and return to HDOS command mode. BYE will not save your program, close your files, or in any other way clean up for you. If you want to save the program you have written, use SAVE or REPLACE before using BYE. BYE will ask you if you are sure before terminating. For example:

```
"*"'BYE<RTN>'
"SURE?"'YES<RTN>'
```

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 in the program or after a CTRL-C stops the program. CONTINUE may be used to enter a program at 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<RTN>'
```

In the following example, CONTINUE starts the program at a specific line number.

```
"*"'GOTO 100<RTN>'
"*"'CONTINUE<RTN>'          (Start execution at line 100)
```

CONTINUE is also useful for entering a program with a variable or variables set at particular values. For example:

BASIC STATEMENTS (Cont)

CONTINUE (Cont)

```
"*"'A = 23.5<RTN>'          (Program continues execution at Line 230
"*"'GOTO 230<RTN)'          with variable A set to the value of 23.5.
"*"'CONTINUE<RTN>'          regardless of previous program effects
                             on A.)
```

DELETE

The DELETE statement is used to remove several lines from the BASIC source code. The form of the DELETE statement is:

```
"*"'DELETE iexp1, iexp2<RTN>'
```

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 certain number of lines from 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.

LIST

This command lists the program on the console terminal for reviewing, editing, etc. The form of the LIST command is:

```
"*"'LIST [LINNUM1], [LINNUM2]<RTN>'
```

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, BASIC lists that line. You can use a CTRL-O or CTRL-C to abort the listing. If both 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. The following example show how to use LIST:

```
"*"'LET A=5:LET B=6'
"*"'PRINT A, B,A+B'
"*"'LET C=A/B'
"*"'PRINT C'
"*"'END'

'RUN<RTN>'

"5    6   11    .833333"
```

BASIC STATEMENTS (Cont)

LIST (Cont)

```
"*"'END AT LINE 50'

"*"'LIST<RTN>'

"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<RTN>'

"*"'20 PRINT A,B,A+B'
"*"'LIST 20,40<RTN>'

"20 PRINT A,B,A+B"
"30 LET C=A/B"
"40 PRINT C"
"*"
```

OLD

The OLD command is used to read some preexisting program into BASIC. OLD performs a SCRATCH command, destroying the previous program before reading in the new one. The format for the OLD command is:

```
"*"'OLD "fname"<RTN>'
```

where "fname" is the filename of the program to be loaded. Note that "fname" must be inclosed in quotation marks [" "]. If no device code is specified, BASIC assumed SY0:. If no extension is specified, BASIC assumes .BAS. For example:

```
"*"'OLD "DEMO"<RTN>'

"*"'OLD "SY1:STARTREK.GAM"<RTN>'
```

If you want to load a new program without disturbing your variables and their values, use the CHAIN command.

BASIC performs a preliminary syntax check on lines read in via the OLD command, just as it would for lines you type yourself on the console. Should the OLD command detect any such syntax errors in the lines being

BASIC ARITHMETIC (Cont)

OLD (Cont)

read, it will insert the characters *ERR* at the spot in the line where the error was detected. This should never occur with programs you have entered and modified with BASIC, since BASIC will not let you type lines with such errors. However, such errors could occur if you used a text editor such as 'EDIT' to modify or create a BASIC program.

You can detect such occurrences by listing the program and looking for the *ERR* symbol. Executing a line with the *ERR* symbol in it will generate a syntax error.

REPLACE

The REPLACE command enables you to replace a file that has previously been stored on the disk. The syntax for the REPLACE command is:

"*"'REPLACE "FNAME"<RTN>'

The default device is SY0:; the default extension is .BAS. Note that you can use the REPLACE command to obtain a printed copy of a program that is currently in memory. For example, if you had a configured line printer, the command:

"*"'REPLACE"LP:"<RTN>'

would cause BASIC to write the source for the program to the line printer, thus giving you a hard-copy listing. The SAVE command cannot be used to obtain hard-copy listings in this way, since SAVE opens the file specified for read to see if it already exists. If you typed 'SAVE "LP:,"' BASIC would print an error message, since the file LP: exists.

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<RTN>'

BASIC ARITHMETIC (Cont)

RUN (Cont)

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 then rerun. This process is continued until the program runs properly without producing any error messages. See "Errors" (page 12-80) 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 gain control of the program by typing CTRL-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.

SAVE

The SAVE command is used to save a BASIC program as an HDOS file. The file can then be listed or copied onto different devices, edited by a text editor, and reread by BASIC (via the OLD command). The SAVE command is the normal method of saving a program that you might want to use again. The format of the SAVE command is:

```
"*"'SAVE "FNAME"<RTN>'
```

where "FNAME" is the name of the file which is to be written. Note that "FNAME" must be inclosed in quotes [" "]. If no device is specified, BASIC assumes SY0:. If no extension is supplied, BASIC assumes .BAS. NOTE: The FNAME must not already exist on the specified device. BASIC will not allow you to replace a file with the SAVE command. This is done so you will not accidentally use the same name for two programs and inadvertently destroy one of them. If you wish to store an updated version of a program, you can delete the old version via UNSAVE, or you can use the REPLACE command. For example:

```
"*"'SAVE "SY1:INCOMTX"<RTN>'       (NOTE: Example for multiple drives.)
"*"'LIST 10<RTN>'
"*"'00010 PRINT "HI THRER"'        (note the error)
"*"'10 PRINT "HI THERE"<RTN>'      (error corrected)
"*"'SAVE "SY1:INCOMTX"<RTN>'       (attempt to replace program)

"! ERROR - FILE ALREADY EXISTS"
"*"'REPLACE "SY1:INCOMTX"<RTN>'     (replace program)
"*"
```

BASIC STATEMENTS (Cont)

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 keyboard to insure that old program lines are not mixed with new program lines. It also assures a clear symbol table. The form of the SCRATCH command statement is:

```
"*"'SCRATCH<RTN>'
```

Before destroying stored information, the user is asked "SURE?" A "Y" reply causes SCRATCH to proceed. Any other response cancels SCRATCH. For example:

```
"*"'SCRATCH<RTN>'          (SCRATCH statement entered)
   "SURE?"'Y<RTN>'         (BASIC query.  Response is Y (Yes))
"*"                        (BASIC is ready for a new entry)
```

STATEMENTS VALID IN THE COMMAND OR PROGRAM MODE

You may use the statements in this section in either the command or 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.

CHAIN

The CHAIN command is used to start the execution of another BASIC program. The format of the CHAIN command is:

```
"*"'CHAIN sexp<RTN>'

    or

"*"'CHAIN sexp,linnum<RTN>'
```

where "sexp" is a string expression containing the filename of the program to be executed. If no device is specified, BASIC assumes SY0:. If no extension is specified, BASIC assumes .BAS.

The CHAIN command causes the current program text to be deleted, the new program(s) to be read in, and the execution to begin. If a line number is specified, execution begins at that line number. Note the

BASIC ARITHMETIC (Cont)

CHAIN (Cont)

GOSUB and FOR loop tables are cleared by the CHAIN process, but no data values (numeric and string variables and arrays) are affected by the CHAIN. However, the data pointer is reset to the top of the data statements.

You can use the CHAIN command in the command mode as a quick way to load and execute a program. For example:

```
"*"'CHAIN "DEMO"<RTN>'

"HI, I'M A BASIC DEMO PROGRAM:"
               (etc)
```

You can use the CHAIN command in the execution mode to start a different program executing, while maintaining any open files and data values. Thus, a program that is too large to fit in memory all at once can be written in several sections, with each section chaining to the next one when ready. As an example, assume we have written a payroll maintenance program that is too large to all fit into memory. This program can perform 5 different functions upon the payroll file. One of these functions may be "add an employee," another one "print monthly checks," and so forth. Because the entire program will not fit into memory at one time, we have split it into five pieces, each of which performs one of the five functions. A section of the program might look like:

```
"*"'00020 DIM A$(4)'
"*"'00030 A$(0)="SY1:PAYROLL1.BAS"'

"*"'00040 A$(1)="SY1:PAYROLL2.BAS"'
.
.
.

"*"'02000 INPUT "WHAT FUNCTION (1-5)",F'
"*"'02010 CHAIN A$(F-1)'
```

This program inputs a number from the operator, indicating which function is to be performed, and then CHAINs to the appropriate program. The value of A$ and the values of all other variables are preserved during the CHAIN. In this example the individual service programs CHAIN back to the master program with a statement:

```
"*"'CHAIN "PAYROLL",2000<RTN>'
```

so the PAYROLL program does not start over at the beginning, but instead starts at line 2000.

BASIC STATEMENTS (Cont)

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:

```
CLEAR
CLEAR varname
```

All variables, arrays, string buffers, etc., are cleared before a 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<RTN>'          (Clears variable A)
"*"'CLEAR A$<RTN>'         (Clears the string variable A$)
"*"'CLEAR A(<RTN>'         (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.

CLOSE

The CLOSE statement is used to close an HDOS file. To read or write to a file, three things must be done in sequence:

1. The file must be opened (see OPEN).
2. The I/O is performed (via "INPUT #chan" or "PRINT #chan").
3. The file must be closed.

The format of the CLOSE statement is:

```
"*"'CLOSE #chan1<RTN>'
"*"'CLOSE #chan1, . . . ,#chann<RTN>'
```

BASIC STATEMENTS (Cont)

CLOSE (Cont)

where "#chan" is the channel number assigned to the file when it was opened. The CLOSE command performs three tasks:

1. If the file was OPENed for writing, the new file is entered into the disk's directory. If the file is not closed, it, and all the information written to it, is lost!

2. The BASIC channel number is freed so a different file may be OPENed on that channel.

3. If there are no open channels with numbers higher than the one being closed, the buffer space in the FILE table (see the FREE command) is freed up. That is, if channels 1 and 2 are open, and you close channel 1, then no FILE table space is freed. When you later close channel 2, then the FILE table space for both channels 1 and 2 is freed.

If your program blows up without closing its channels, you may want to type CLEAR to discard the partially written files. If you want to save any partial files, use CLOSE in command mode to close the files.

If the channel number(s) listed in the CLOSE command have not been opened or have already been closed, they are ignored.

CNTRL

CNTRL is a multi-purpose command used to set various options and flags. The form of the CNTRL statement is:

"*"'CNTRL iexp1, iexp2<RTN>'

where, as usual, iexp1 and iexp2 signify integer expressions.

The various CNTRL options are as follows:

	iexp1	iexp2
CNTRL	0,	nnn
CNTRL	1,	n
CNTRL	2,	n
CNTRL	3,	n
CNTRL	4,	n

BASIC STATEMENTS (Cont)

CNTRL 0 (ZERO)

The CNTRL 0, nnn command sets up a GOSUB routine to process CTRL-B characters. The line number of the routine is specified as "iexp2." When a CTRL-B is entered on the keyboard, 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:

```
"*"'00010 CNTRL 0,500'
"*"'00020 FOR A=1 TO 9'
"*"'00030 PRINT A,A*A,A*A*A'
"*"'00040 NEXT A'
"*"'00050 END'
"*"'00500 PRINT "THAT TICKLES"'
"*"'00510 RETURN'
"*"'RUN<RTN>'

    "1            1          1"
    <CTRL-B> "2        4          8"

"THAT TICKLES"
    "3            9          27"
    "4          16<CTRL-B>       64"
"THAT TICKLES"
    "5          25           125
     6          36           216"

    <CTRL-B>THAT TICKLES
    "7          49  <CTRL-B> 343"
"THAT TICKLES"

    "2          64           512
     9          81           729"
"*"'END AT LINE 50'
"*"
```

During the execution of the program containing these three statements, a CTRL-B from the keyboard momentarily interrupts 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.

BASIC STATEMENTS (Cont)

CNTRL 1

The CNTRL 1, n command sets the number of digits permitted before the exponential notation is used. Normal mode N = 6. For example:

```
"*"'CNTRL 1,2<RTN>'    (Numbers >/ 100 are to be in exponential form)

"*"'PRINT 101<RTN>'
"1.01000E+02"
```

CNTRL 2 [Applies only to the H8 Computer.]

The CNTRL 2, n command controls the H8 front panel LED display mode. The control functions are:

```
"*"'CNTRL 2,0<RTN>'  (Turn display off (Normal Mode).)

"*"'CNTRL 2,1<RTN>'  (Turn display on without update. (For writing
                      into a display see the example under "The Segment
                      Function," SEG (narg) on page 12-72.)

"*"'CNTRL 2,2<RTN>'  Turn display on with update (to monitor a register
                     or memory location).
```

NOTE: The CNTRL 2, n command has no effect on an H89, since there is no front panel LED display on this model.

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<RTN>'
"*"'PRINT 1,2,3,4,3,2,1,0<RTN>'
       "1    2    3    4    3    2    1    0"
```

CNTRL 4

NOTE: CNTRL 4 applies only to HDOS version 2.0 and below. It is included in this manual to provide continuity concerning programs written for HDOS version 2.0 and earlier versions.

BASIC STATEMENTS (Cont)

CNTRL 4 (Cont)

The CNTRL 4 command is used to control the HDOS Operating System's overlay handling. Part of the HDOS 2.0 system does not reside permanently in RAM, but is kept on the disk in SY0:. When it is needed, it is read into memory temporarily. The programs that comprise these temporary disk files are called "overlays."

The statement:

"*"'CTRL 4,1<RTN>'

will cause these HDOS overlays to remain in memory permanently. This will greatly speed up the execution of the RUN, SAVE, UNSAVE, OLD, REPLACE, OPEN, and CLOSE statements, at the cost of about 2.5k bytes of free RAM.

Executing the statement:

"*"'CNTRL 4.0<RTN>'

restores HDOS to its normal mode, and allows BASIC to make use of that 2.5k bytes of RAM. When you first run BASIC, it starts up in the CNTRL 4, 0 mode. Users with sufficient free space will find a significant speed increase by using the CNTRL 4,1 command.

NOTE: The CNTRL 4,n command cannot be executed as a program statement. If you want to "lock" the overlays in memory, do so before executing the program. Good programming practice dictates that you do a CNTRL 4, n command prior to putting the program into memory.

NOTE: CNTRL 4 applies only to versions of HDOS over 2.0 and earlier.

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 (. . .)<RTN>'

The expressions "iexp1" through "iexpn" are integer expressions specifying the bounds of each dimension. Dimensions are 0 to to "expn." So, for example, the statement:

"*"'DIM A(5,5)<RTN>'

BASIC STATEMENTS (Cont)

DIM (DIMENSION) (Cont)

reserves an array 6x6 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)<RTN>'
```

dimensions the following arrays:

VARIABLE	SIZE	
A6	4 by 3	12 elements
B	6 by 6	36 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<RTN>'

"*"'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<RTN>'
     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"
"*"
```

BASIC STATEMENTS (Cont)

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 scaler 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<RTN>'
    "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<RTN>'
    "2  4  6  8  10  12  14  16  18  20"
"*"
```

Prior to the execution, the variable A is assigned the value of 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>/ "nexp2," the loop terminates. For example, the program:

```
"*"'LIST<RTN>'

"10 FOR J=2 TO 18 STEP 4"
"20 J=18"
"30 PRINT J;:NEXT J"
"40 END"
```

BASIC STATEMENTS (Cont)

FOR AND NEXT (Cont)

```
"*"'RUN<RTN>"
"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<RTN>'

"20"
"*"
```

is executed only once, as the initial value exceeds the terminal value. However, it this example is modified to read:

```
"*"'FOR A=20 TO 2 STEP -2:PRINT A;:NEXT A<RTN>'
"20  18  16  14  12  10  8  6  4  2"
"*"
```

The negative step allows a 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 not terminate, execution is transferred to the statement following NEXT.

The expressions in the FOR statement can be any acceptable BASIC numeric expression.

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<RTN>'
     "2  3  4  5  6  7  8  9  10"
"*"
```

BASIC STATEMENTS (Cont)

FOR AND NEXT (Cont)

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:

LEGAL NESTING

Two-Level Nesting

```
LOOP A :--- FOR A=1 TO 50
FIELD->:
       : :- FOR B=1 TO 10
LOOP B : :
FIELD--->:_ NEXT B
       :
LOOP C : :- FOR C=1 TO 20
FIELD--->:
       : :_ NEXT C
       :
       :--- NEXT A
```

Three-Level Nesting

```
LOOP A  :-------- FOR A=1 TO 10
FIELD-->:
        :  :----- FOR B=1 TO 5
LOOP B  :  :
FIELD----->:
        :  :  :-- FOR C=1 TO 30
LOOP C  :  :  :
FIELD-------->:
        :  :  :-- NEXT C
        :  :
        :  :
        :  :   :- FOR D=1 TO 40
LOOP D  :  :   :
FIELD--------->:
        :  :   :_ NEXT D
        :  :
        :  :----- NEXT B
        :
        :-------- NEXT A
```

ILLEGAL NESTING

Two-Level Nesting

```
LOOP A   :----- FOR A=1 TO 100
FIELD--->:
         : :--- FOR B=1 TO 10
         : :
         :-:--- NEXT A
LOOP B     :
FIELD----->:
           :
           :--- NEXT B
```

Three-Level Nesting

```
LOOP A  :-------- FOR A=1 TO 3
FIELD-->:
        :  ------ FOR B=1 TO 10
LOOP B  :  :
FIELD----->:
        :  :  --- FOR C=1 TO 5
LOOP C  :  :  :
FIELD-------->:
        :  :  :
        :  :  :__ NEXT C
        :  :
        :  :  :-- FOR D=1 TO 30
LOOP D  :  :  :
FIELD-------->:
        :  :  :
        :  :  :__ NEXT D
        :  :
        :__:_____ NEXT A
           :
           :_____ NEXT B
```

BASIC STATEMENTS (Cont)

FOR AND NEXT (Cont)

Note that both columns of nesting illustrations shown on the previous page are shown in two-level and three-level forms. However, the right-hand columns are not truly nesting, but a crossover of FOR and NEXT loops, and therefore 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.

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 insures 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

The FREE statement displays the amount of memory used by BASIC and any program material. It also displays the total amount of free space left, which is dependent upon 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<RTN>'

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)
    WORK = nnnn     (Bytes used by expression and
                       function evaluation)
    STRN = nnnn     (Bytes used by strings)
    TSTR = nnnn     (Bytes used by temporary strings)
    FILE = nnnn     (Bytes used by file buffers)
    FREE = nnnn     (Total number of free bytes)
```

BASIC STATEMENTS (Cont)

FREE (Cont)

For example, in running the program:

```
"*"'10 GOSUB 10<RTN>'
```

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<RTN>'
--------------
  "TEXT = 9
   SYMB = 0
   FORL = 0
   GSUB = 0
   WORK = 0
   STRN = 0
   TSTR = 0
   FILE = 0
   FREE = 34320"
"*"'10 GOSUB 10<RTN>'
---------------------
"*"'RUN<RTN>'
-------------

"! ERROR - Out of RAM Space At Line 10"
"*"'FREE<RTN>'
 --------------
  "TEXT = 9
   SYMB = 0
   FORL = 0
   GSUB = 32928
   WORK = 0
   STRN = 0
   TSTR = 0
   FILE = 0
   FREE = 1392"
"*"
```

Note that the file table never contains less than 283 bytes when a channel is open. The file table contains the disk file buffers necessary to read and write files. The 283 bytes are required for BASIC's internal buffer, which it uses for such commands as OLD, SAVE, and REPLACE.

BASIC STATEMENTS (Cont)

FREE (Cont)

You can compute the amount of space used by the FILE table with the formula:

```
bytes = N * 256
```

where N is the number of the highest-numbered channel that is open. Thus, when your program opens files, it should open them on the lowest numbered channel first. If you open a file on channel 3, space is reserved for the buffers for channels 1 and 2, even if they are never opened.

FREEZE

The FREEZE command is used to store BASIC, your program, and all of your program's variables on any mounted disk. The format for the command is:

```
"*"'FREEZE "FNAME"<RTN>'
```

where "FNAME" is the filename under which the "frozen" program will be stored. If no device is specified, BASIC assumes SY0:. If no extension is specified, BASIC assumes .BAF (for BASIC Frozen).

The FREEZE command allows you to suspend work temporarily; perhaps to power-down overnight or to allow some more important work to interrupt. This command is not intended as a general-purpose, program-save command. The SAVE and REPLACE commands are provided for normal program saving. The file created by the FREEZE command is in absolute binary format, and cannot be displayed or edited. Its sole use is to be unfrozen with the UNFREEZE command.

The file is quite large because it contains all of the BASIC interpreter in addition to your program and variables. Frozen programs are non-transferrable, meaning they cannot be unfrozen by different versions of BASIC than the one they were frozen with.

NOTE: All files must be closed before a program is saved via FREEZE.

BASIC STATEMENTS (Cont)

GOSUB AND RETURN

A subroutine is a section of a 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 the data statements, if any.

Program execution begins and continues until a GOSUB statement is encountered. The form of the GOSUB statement is:

"*"'GOSUB LINNUM<RTN>'

where LINNUM is the line number of 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<RTN>'

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<RTN>'

RETURN causes the program control to return to the statement following the original GOSUB statement. A subroutine must always be terminated by a RETURN.

BASIC STATEMENTS (Cont)

GOSUB AND RETURN (Cont)

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.

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 LINNUM<RTN>'
```

When this statement is executed, program control transfers to the line number specified by LINNUM. For example:

```
"*"'10 LET A=1'
"*"'20 GOTO 40'
"*"'30 LET A=2'
"*"'40 PRINT A'
"*"'50 END'
"*"'RUN<RTN>'
     --------
"1"
"END AT LINE 50"
"*"
```

BASIC STATEMENTS (Cont)

GOTO (Cont)

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 upon the results of a relation test. The forms of the IF statement are:

```
                |THEN|
IF expression <    > LINNUM<RTN>  or
                |GOTO|

IF expression THEN statement<RTN>
```

The expression frequently consists of two variables combined by the relational operators described in "Relational Operators" (page 12-13). In the first form, if the result of the expression is true, control passes to the specified line number (LINNUM).

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. 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<RTN>'

"A=B  10"

"END AT LINE 70"
```

BASIC STATEMENTS (Cont)

IF THEN (IF GOTO) Cont)

"*"'CONTINUE<RTN>'

"A< >B 5"

"END AT LINE 40"

"*"'CONTINUE<RTN>'

"A< >B 20"

"END AT LINE 40"

"*"

NOTE: The expression can be an arbitrarily complex expression. For example:

```
IF (A<3) AND NOT (B>C) THEN 33
```

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 or a string variable "var$." The expression may be either an arithmetic "nexp" or a string expression "sexp." 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.

```
LET A=B=3<RTN>
```

causes A to be set to 65,535 (true) if B89e is equal to 3, or 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 keyword LET if you prefer. For example, the following two statements produce identical results:

BASIC STATEMENTS (Cont)

```
"*"'10 LET A = 6'
```

or

```
"*"'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 is lost when this statement is executed.

LOCK

The LOCK statement protects your program by preventing execution of the following command mode statements:

BUILD	CLEAR	SCRATCH
BYE	DELETE	UNFREEZE
CHAIN	RUN	

It also prevents the entry or deletion of program text. Variables can be changed, but not deleted. The form of the LOCK statement is:

```
"*"'LOCK<RTN>'
```

A lock error (LOCK) is generated if you attempt to enter a "locked out" command mode statement, such as RUN. Use the UNLOCK statement to abort the LOCK mode.

ON. . . GOSUB

The ON. . . GOSUB statement allows you to program a completed GOSUB. When you use the ON. . . GOSUB statement, use a RETURN at the end of

BASIC STATEMENTS (Cont)

ON. . . GOSUB (Cont)

the subroutine to return program control to the statement following the ON ... GOSUB statement. The form of the ON ... GOSUB statement is:

"*"'ON iexpi GOSUB LINNUM1, ..., LINNUMn<RTN>'

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 LINNUM1 through LINNUMn. If the expression "iexp1" evaluates to 1, for example, control is passed to line number "LINNUM1." If the expression "iexp1" evaluates to 3, for example, control is passed to line number "LINNUM3." 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 computer GOTO. The form of the ON ... GOTO statement is:

"*"'ON iexpi GOTO LINNUM1, ... , LINNUMn<RTN>'

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 LINNUM1 through LINNUMn. If the expression "iexp1" evaluates to 1, for example, control is passed to the line number given by the expression "LINNUM1." If the expression "iexp1" evaluates to 3, for example, control is passed to line number given by the expression "LINNUM3." If the expression "iexp1" evaluates to 0 or to an index greater than the number of statement numbers listed, control is passed on the the next program statement.

OPEN

The OPEN command is used to open HDOS files so that they can be read or written from BASIC. The format of the OPEN command is:

"*"'OPEN sexp FOR READ AS FILE #iexp <RTN>' or
"*"'OPEN sexp FOR WRITE AS FILE #iexp <RTN>'

The first form is used to open files for reading via the INPUT command. The second form is used to open files for writing via the PRINT command. "sexp" is a string value that contains the HDOS filename. If no device is specified, BASIC assumes SY0:. If no extension is

BASIC STATEMENTS (Cont)

OPEN (Cont)

specified, BASIC assumes .DAT. Any legal device may be used. "iexp" represents the channel number that is to be assigned to the file. BASIC has five channels: 1 through 5. This means that you can have a maximum of five files open at one time. You can close a file and then reuse its channel for some other file. After the OPEN statement, the only way to refer to the file is by its channel number; the filename is no longer needed. For example:

```
OPEN "TEAP" FOR WRITE AS FILE #3
OPEN "SA1:RALPH.WRK" FOR READ AS FILE #1
OPEN A$ FOR WRITE AS FILE #1
OPEN "TT:" FOR WRITE AS FILE #2
```

To print or output to the "alternate terminal" device:

```
"*"'00010 OPEN "AT:" FOR WRITE AS FILE #1'
"*"'00020 FOR I=1 TO 10'
"*"'00030 PRINT #1,I,SQR(I)'
"*"'00040 NEXT I'
"*"'00050 CLOSE #1'
"*"'00060 STOP'
"*"'00070 END'
```

NOTES:

1. Although five channels are available, 1, 2, 3, 4, and 5, you should use the lowest-numbered channel available when opening a file in order to minimize the amount of memory space required. See the FREE command discussion (page 12-40) for more information.

2. Although files may be opened to any legal device, including the console terminal (device TT:), you should use the regular INPUT and PRINT statements for communicating with the console. Due to the requirements of HDOS device I/O, BASIC saves up the data you write to a file via PRINT until there are 256 bytes of data, and then writes the 256 bytes all in one group.

Likewise, when reading, BASIC reads ahead a 256-byte block of data and then supplies it as needed to the INPUT #chan statements. Thus, if you write to the console via a channel opened on the device TT:, the lines will not appear on the screen when you PRINT them, but when BASIC has accumulated 256 bytes worth (or when the file is closed).

BASIC STATEMENTS (Cont)

OUT

The OUT statement is used to output binary numbers to an output port. The form of the OUT statement is:

"*"'OUT iexp1, iexp2<RTN>'

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. The form of the pause statement is:

```
PAUSE [iexp]
```

If the optional expression iexp is omitted, PAUSE suspends execution until you type a carriage return. If the expression iexp is present, PAUSE delays 2* iexp milliseconds, and then allows execution to resume. The maximum value for iexp is 30,000, allowing a maximum delay of about 60 seconds.

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 the information scrolls off the screen.

BASIC STATEMENTS (Cont)

STATEMENTS VALID IN THE COMMAND OR PROGRAM MODE (Cont)

POKE

CAUTION *** **** *** CAUTION *** ***** *** CAUTION *** **** *** CAUTION

Prevent possible damage to the computer operating system!

The POKE function gives an experienced BASIC user direct control of virtually all of the features of the computer. However, subtle misuse of POKE can interfere with the operating system and cause it to cease correct functioning. Therefore, CAUTION in using POKE is advised.

**** *** *** *** *** *** *** *** *** *** *** ****

The POKE statement is used to place a value in a particular memory location. The form of the POKE statement is:

```
POKE Location, Value
```

The "Location" is a decimal integer in the range of 0 to 65,535. This references any individual byte of a memory location. The "Value" is also an integer expression lying in the range of 0 to 255. You can examine the contents of a memory location by using the PEEK function, described on Page 12-69.

PRINT

The PRINT statement is used to output data to the console terminal or to an HDOS file. The form of the PRINT statement is:

```
PRINT [nexp1,sep1, . . . . .[,nexpn, sepn]]     or
PRINT #chan, [nexp1, . . . . . [,nexpn,sepn]]
```

The first form shown is for writing text and values to the console terminal. The second form is for writing values and text to an HDOS file. 'chan' is the channel number of a file which must have been previously opened for WRITE. See the discussion of the OPEN and CLOSE command for more information. Except for the destination of the data, both forms of the command are otherwise identical.

BASIC STATEMENTS (Cont)

PRINT (Cont)

The expression and separators contained within the brackets are optional. When used without these optional expressions and separators, the simple

```
PRINT          or
PRINT #CHAN
```

statement outputs a blank line.

PRINT: 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<RTN>'

"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 Comma [,] and Semicolon [;], page 6-53. 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.

BASIC STATEMENTS (Cont)

PRINT: 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 inclosing them in quotation marks ["]. For example:

```
"*"'10 PRINT "THIS IS A HEATH COMPUTER"'
"*"'RUN<RTN>'

"THIS IS A HEATH COMPUTER"

"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<RTN>'

"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.

PRINT: Use of the comma [,] and semicolon [;]

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:

BASIC STATEMENTS (Cont)

PRINT: Use of the comma [,] and semicolon [;] (Cont)

```
"*"'10 A=5,55555:B=2'
"*"'20 PRINT A,B,A+B,A*B,A-B,B-A'
"*"'30 END'
"*"'RUN<RTN>'

"   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 A,B,A+B,,A*B,A-B,B-A'
"*"'30 END'
"*"'RUN<RTN>'

   "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<RTN>'

"1              2"
"3"

"END AT LINE 50"
"*"
```

BASIC STATEMENTS (Cont)

PRINT: Use of the Comma [,] and Semicolon [;] (Cont)

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 semicolon is inserted in place of the comma. A semicolon does not move the next output to the next PRINT zone, but simply prints the next variable, including its leading and trailing blanks. 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<RTN>'

"1  2  3"

"   2  3"

"      4"

"END AT LINE 50"
"*"
```

NOTE: If either a comma or a semicolon is the final character of 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 is:

```
READ var1, . . . , varn
READ val1, . . . , valn
```

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:

BASIC STATEMENTS (Cont)

READ AND DATA (Cont)

```
"*"'05 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<RTN>'

"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 value in the DATA statement must match the type of

corresponding variable in the READ statement. When the DATA statement has been 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 the DATA statement contains more expressions than the READ statement has variables.

The data values in a DATA statement must be separated by commas. If the value is to be read into a numeric variable or array, it must be a number. If the value is to be read into a string variable or array, no specific format is required. If the value is inclosed in quotation marks ["], the quoted characters are assigned to the string variable. If the variable is not inclosed in quotes, BASIC uses the characters until a comma or the end of the line is reached. Thus, if you wish to read a comma as part of the value, you MUST inclose the value in quotes.

You may not include a quote character in the value. For example:

```
"*"'10 READ A$,B$,C$'
"*"'20 PRINT A$,B$,C$'
"*"'30 DATA HI THERE, "HI, THERE", YES'
"*"'RUN<RTN>'

"HI THERE          HI, THERE       YES"
```

A field in the DATA statement may be left null by means of two adjacent commas. This causes the associated variable to retain the old value. For example:

BASIC STATEMENTS (Cont)

READ AND DATA (Cont)

```
"*"'10 A=1:B=1:C=1'
"*"'20 READ A,B,C'
"*"'30 PRINT A,B,C'
"*"'40 DATA 3,4'
"*"'50 END'
"*"'RUN<RTN>'

"3          1          4"


"END AT LINE 50"
"*"
```

If a DATA statement appears on a line, it must be the only statement on that 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, the location of DATA statements is arbitrary, as long as the values contained within the DATA statements appear in the correct order. However, good programming practice dictates that all DATA statements occur near the end of the program. This makes it easy for the programmer to modify the DATA statements if necessary.

If a value contained in a DATA statement is incorrect, 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 on the screen.

If the number of values 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

BASIC STATEMENTS (Cont)

REM (REMARK) (Cont)

statement inserts comments into the program without affecting execution, it does 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<RTN>'

"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.

BASIC STATEMENTS (Cont)

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<RTN>
```

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<RTN>'

"*"'STEP 3<RTN>'
"1          2          3"
"NXT= 30"

"*"'STEP<RTN>'
"NXT= 40"

"*"'STEP<RTN>'
"NXT= 50"

"*"'STEP<RTN>'
"1          2          3"
"NXT= 60"

"*"'STEP 2<RTN>'

"END AT LINE 70"
"*"
```

BASIC STATEMENTS (Cont)

UNFREEZE

The UNFREEZE command is used to restore a program that has been frozen with the FREEZE command. (See "FREEZE" on page 12-42 for more information.) The format of the UNFREEZE command is:

```
UNFREEZE "fname"<RTN>
```

where "fname" is the name of the previously frozen file. If no device is specified, BASIC assumes SY0:. If no extension is specified, BASIC assumes .BAF.

UNLOCK

The UNLOCK statement aborts the LOCK mode and restores the use of all command mode statements. The form of the UNLOCK statement is:

```
*UNLOCK<RTN>
```

UNSAVE

The UNSAVE command is used to delete programs and files from the disk. The form of the UNSAVE command is:

```
UNSAVE "fname"<RTN>
```

where "fname" is the name of the file to be deleted. If no device is specified, BASIC assumes SY0:. If no extension is specified, BASIC assumes .BAS. Unless the file on the disk is write-protected, you can use UNSAVE to delete any file: a BASIC program, a data file, or anything else.

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 12-55) 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 mode.

BASIC STATEMENTS (Cont)

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 proceeds.

```
"*"'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<RTN>'

"9          3          25          5"

"END AT LINE 40"

"*"
```

BASIC STATEMENTS (Cont)

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 of line 65,535.

INPUT AND LINE INPUT

The INPUT and LINE INPUT statements are used when data is to be read from the console terminal, or from an HDOS file. The form of the INPUT statement is:

```
INPUT prompt;var1, . . . . . , varn              or
INPUT #chan, prompt;var1, . . . . . , varn
```

The #chan specification (shown in the second example) is optional, and if present specifies the channel number of the file (which must have been previously OPENed for INPUT) to be read from. An INPUT statement with a file channel number specified works just like a regular INPUT statement, except that a line is read from the file rather than the console. Values are read from the line in exactly the same way as they would be from a line typed at the keyboard. If necessary, BASIC will read more lines from the file to satisfy the INPUT statement. Any unused values on the line are discarded.

If the first element following the INPUT statement is a string, INPUT assumes it is a prompt and types the string instead of the question mark [?]. If you do not want a prompt string but the first variable is a string variable, a leading semicolon is required. For example:

```
INPUT ;S3$(2)
```

tells BASIC that the data read from the console terminal is to be placed in the third element of the string array S3$. Note that a prompt is meaningless when inputting from HDOS files.

The data line input from the console or read from the HDOS file is identical in format to the DATA statement except that the DATA keyword is omitted.

BASIC STATEMENTS (Cont)

INPUT AND LINE INPUT (Cont)

String values need not be inclosed in quotes unless they contain the comma [,] character. Multiple data values on the same line must be separated by commas.

As in the DATA statement, null fields (two commas in a row) cause the variable to retain its previous value. If the user response (or the line read if you are inputting from an HDOS file) does not supply sufficient data to complete the INPUT statement, another "?" prompt is issued (if you are inputting from the console) and another line is read from the console or the data file. CAUTION: If you supply too much data or there is too much on a line read from a file, it will be ignored. The next INPUT statement issues a fresh read to the terminal or file.

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, except that the supplied line is read in its entirety into the string variable, regardless of commas [,] or quotation marks ["]. For example:

```
LINE INPUT "YES OR NO?";A$            or
LINE INPUT #2,;A$
```

Note that the channel number in the second example is must be followed by a comma; the following semicolon tells BASIC that A$ is a variable name, not a prompt.

LINE INPUT, unlike READ and INPUT, allows you to read a string containing a quote ["] character. Note that you should NOT inclose your reply in quotes, since these will be accepted as part of the string.

BASIC STATEMENTS (Cont)

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<RTN>'
"1          2          3"
"END AT LINE 30"

"*"'15STOP<RTN>'

"*"'RUN<RTN>'

"*"'STOP AT LINE15'
"*"'PRINT A<RTN>'

"1"
"*"'*15<RTN>'                (STOP deleted)

"*"'RUN<RTN>'
"1          2          3"

"END AT LINE 30"
"*"
```

****

PREDEFINED FUNCTIONS

INTRODUCTION

There are 31 predefined functions in B. H. BASIC. These functions perform standard mathematical operations such as square roots, logarithms, string manipulations, 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 string expressions (sexp).

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. [Note: The term 10-38 means 10 with a negative exponent of 38. Also the term 10+37 means 10 with a positive exponent of 37.]

The predefined functions may be used in either the command or program mode.

ARITHMETIC AND SPECIAL FEATURE FUNCTIONS

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)<RTN>'

"5.5"                          or

"*"'PRINT ABS(SIN(3.5))<RTN>'

".350783"

"*"
```

NOTE: The sine of 3.5 radians is -.350783.

ARITHMETIC AND SPECIAL FEATURE FUNCTIONS (Cont)

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"<RTN>'

"45.0001 DEGREES"

"*"'*PRINT 4*ATN(1)<RTN>'

"3.14159"

"*"                                NOTE: Pi = 3.14159
```

THE CHARACTER INPUT FUNCTION, CIN(chan)

The CIN function is used to read a character from any open file, or from the console terminal (if chan=0). If the value returned is positive, then it is the next byte read from the file, or the next character read from the console (if chan=0). If the value returned is negative, then an end-of-file has been detected on the file, or no line has yet been entered on the console (if chan=0). For example:

```
"*"'*PRINT CIN(0)<RTN>'

"-1"

"*"
```

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)<RTN>'

".500003"

"*"
```

ARITHMETIC AND SPECIAL FEATURE FUNCTIONS (Cont)

THE EXPONENTIAL FUNCTION, EXP(nexp)

The EXPONENTIAL function returns the value e nexp. If "nexp" exceeds 88, an overflow 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))<RTN>

"2.71828          7.38905          1.64873"
```

[Note: The terms "e nexp" mean that "e" has an exponent of "nexp." Also the number "10" has an exponent of "38." This relationship is impossible to illustrate with the H89 Computer without requiring a note of explanation.]

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 LINE NUMBER FUNCTION, LNO(iexp)

BASIC statements that refer to line numbers (such as GOTO, GOSUB, and so forth) do not allow the line number to be expressed as a numeric expression. The LNO function is provided to convert an integer expression into a line number. For example:

```
    GOTO 20<RTN>
                        and
    GOTO LNO(2*10)<RTN>
```

ARITHMETIC AND SPECIAL FEATURE FUNCTIONS (Cont)

THE LINE NUMBER FUNCTION, LNO (iexp) (Cont)

Both cause a jump to statement number 20. You can use the LNO function anywhere a line number is required; it provides a very powerful "computed GOTO" facility. A program can compute the line number it wishes to jump to (or call, via GOSUB) by using the LNO function. Some more examples:

```
GOSUB LNO(2*Y+100)

ON I GOTO 20,30,LNO(I),LNO(I*2)

IF (A=B) THEN GOTO LNO(A)
```

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:

```
LOGa N = LOGeN/LOGe a
```

Where "a" is a subscript representing the desired base, and "e" is also a subscript. The last "a" is shown on the main line. 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"<RTN>'

"A POWER RATIO OF 2 IS 3.0103 DECIBELS"

"*"
```

THE MAXIMUM FUNCTION, MAX (nexpi, . . . , nexpn)

The MAXIMUM function returns the maximum value of all the expressions which are arguments of the function. For example:

```
"*"'LET A=1'
"*"'20 PRINT MAX(COS(A),SIN(A)/COS(A))'
"*"'30 END'
"*"'RUN<RTN>'

"*"
```

The expression containing the maximum value is the expression for the tangent of 1 radian, 1.55741.

ARITHMETIC AND SPECIAL FEATURE FUNCTIONS (Cont)

THE MINUMUM FUNCTION, MIN(nexpi, . . . ,nexpn)

The MINIMUM function returns the lowest value of all expressions contained in the argument. For example:

```
"*"'PRINT MIN(1,2,3,4, . . . ,5)<RTN>

".5"
"*"
```

THE PAD FUNCTION, PAD(0) [Applies only to the H8 Computer.]

The PAD function returns the value of the keypad pressed on the H8 front panel. For example:

```
"*"'PRINT PAD(0)<RTN>'

"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 Functions," page 12-72, for an additional example.)

NOTE: The PAD function must be completely executed before any other function will respond. Therefore, CTRL-C, etc, will not work until you press an H8 front panel key.

The PAD function is intended for use on an H8 Computer, where front panel access is necessary. On an H89 Computer, there is no front panel. If a BASIC program using the PAD function is run on an H89, a zero (0) will be returned as soon as the PAD(0) is executed. The CTRL-C function is not disabled on the H89.

THE PEEK FUNCTION, PEEK(iexp)

The PEEK function returns the numeric value of the byte at memory location iexp, where iexp is in decimal.

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)<RTN>'
```

ARITHMETIC AND SPECIAL FEATURE FUNCTIONS (Cont)

THE PIN FUNCTION, PIN(iexp) (Cont)

where "A" now contains the data that was at port #38 (46 octal).

THE POSITION FUNCTION, POS(chan)

The POSITION function returns the current terminal printhead (cursor) position. The argument "chan" specifies the I/O channel number (see the OPEN statement) you wish to interrogate. BASIC maintains a separate cursor address for each I/O channel in use. Channel 0 is always the console channel, and is always considered "open." Thus, use POS(0) to read the position of the console cursor. The value returned is a decimal number indicating the column number of the printhead (cursor) postion. For example:

```
"*"'PRINT POS(0), POS(0), POS(0); POS(0); POS(0)<RTN>'

"1             14             28   32   36"

"*"
```

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 this 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 program may continuously return the same number while debugging 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:

[Note: The example will be found on the next page.]

ARITHMETIC AND SPECIAL FEATURE FUNCTIONS (Cont)

THE RANDOM FUNCTION, RND(narg) (Cont)

```
"*"'10 RUN FOR A=0 to 2'
"*"'20 PRINT RND(1)'
"*"'30 NEXT'
"*"'40 END'
"*"

"*"'RUN<RTN>'
-------------
".93677"
".566681"
".53128"

"END AT LINE 40"

"*"'20PRINT RND(0)<RTN>'
------------------------

"*"'RUN<RTN>'

".332306"
".332306"
".332306"

"END AT LINE 40"
"*"'20PRINT RND(-1)<RTN>'
-------------------------

"*"'RUN<RTN>'
-------------
"6.25305E-02"
"6.25305E-02"
"6.25305E-02"

"END AT LINE 40"
"*"'20PRINT RND(-5)<RTN>'
-------------------------

"*"'RUN<RTN>'
-------------
".460968"
".460968"
".460968"

"END AT LINE 40"
----------------
"*"
```

ARITHMETIC AND SPECIAL FEATURE FUNCTIONS (Cont)

THE SEGMENT FUNCTION, SEG(narg) [Applies to the H8 Computer only.]

The SEG function returns a numeric value which is the correct 8-bit binary number to display the digit on the H8 front panel LEDs. The argument must be an integer between 0 and 9. The following program demonstrates the use of PAD, POKE, and SEG in Extended Benton Harbor BASIC:

```
    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<RTN>'
```

When the program is executed, the H8 front panel LEDs respond to the H8 keypad numeric entries. To escape from the program, you would type CTRL-C, and then press any key on the H8 front panel.

NOTE: The SEG function is not useful on the H89 Computer. Running this sample program on your H89 will produce no results. Type CTRL-C to exit.

ARITHMETIC AND SPECIAL FUNCTIONS FEATURES (Cont)

THE SIGN FUNCTION, SGN(narg)

The SIGN function returns the value +1 if "narg" is a positive value, and 0 if "narg" is 0, and -1 if "narg" is negative. For example:

```
"*"'PRINT SGN(5.6)<RTN>'

"1"

"*"'PRINT SGN(-500)<RTN>'

"-1"

"*"'PRINT SGN(12-12)<RTN>'

"0"

"*"
```

THE SINE FUNCTION, SIN(nexp)

The SIN function returns the sine of the argument (nexp) expressed in radians. For example:

```
"*"'PRINT SIN(30/57.297)<RTN>'

".499999"

"*"
```

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<RTN>'

    "12              14              600"

"*"
```

ARITHMETIC AND SPECIAL FUNCTIONS FEATURES (Cont)

THE 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. Positive examples follow:

```
"*"'FOR A=0 TO 5:PRINT A,SQR(A),A*A:NEXT<RTN>'

"0         0              0"
"1         1              1"
"2         1.41421        4"
"3         1.73205        9"
"4         2              16"
"5         2.23607        25"

"*"
```

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<RTN>'

                               60          70"

"*"
```

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)<RTN>'

".999996"

"*"
```

STRING FUNCTIONS

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 a string, generate a character string corresponding to a given number, or generate a number for a given string.

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 a comparison to must ASCII tables. See "Appendix B" for details. For example:

```
"*"'PRINT ASC("ABC")<RTN>'

"65"

"*"'PRINT CHR$(65)<RTN>'

"A"

"*"
```

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 B" for details. For example:

```
"*"'PRINT CHR$(65)<RTN>'          NOTE: If iexp = 0, the generated
"A"                                  string is null.
"*"'PRINT CHR$(70)<RTN>'
"F"

"*"
```

STRING FUNCTIONS (Cont)

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)<RTN>'

"HELLO, THI"

"*"
```

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?")<RTN>'

"23"

"*"
```

THE MATCH STRING FUNCTION, MATCH (sexp1,sexp2,iexp)

The MATCH function searches the string sexp1 for any substrings matching sexp2. The search starts with character iexp in the string sexp1. If iexp = 1, the search starts at the first character in sexp1. If iexp = 2, the search starts at the second character in sexp1, and so forth. MATCH returns the character number of the start of the substring in sexp1, if one was found, and a 0 if it was not found. For example:

```
"*"'PRINT MATCH("THIS IS A RATHER LONG STRING","TH",2)<RTN>'

"13"

"*"
```

Note that MATCH found the TH in RATHER, not in THIS. Since the MATCH call specified a search to start with the second character, BASIC started searching at the "HIS IS . . . ", thereby ignoring the T in "THIS."

STRING FUNCTIONS (Cont)

THE MIDDLE STRING FUNCTION, MID$(sexp, iexp1, [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)<RTN>'

"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 (".0032E-1")<RTN>'

"3.00000E-04"

"*"
```

THE RIGHT STRING FUNCTION, RIGHT$ (sexp,iexp)

The RIGHT STRING function returns the rightmost "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)<RTN>'

"IS A TEST"

"*"
```

STRING FUNCTIONS (Cont)

THE STRING FUNCTION, 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)<RTN>'

"100"                          (STR$ Function)

"*"'PRINT "100"<RTN>'

"100"                          (Normal string printing)

"*"
```

GENERAL TEXT RULES

BLANKS AND TABS

BASIC programs are generally "free format." That is, blanks (spaces and TABS) may be included freely with the following restrictions.

1. Variable names, keywords, and numeric constants may not contain imbedded blanks or tabs.

2. Blanks or tabs 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:

GENERAL TEXT RULES (Cont)

LINE INSERTION (Cont)

```
"*"'100PRINT "HEATH BASIC"'
"*"'LIST<RTN>'
---------------------------
"*"'100 PRINT "HEATH BASIC"'

"*"
```

Each time you type a statement with a line number, BASIC performs some simple syntactical checks before inserting the line into your program. BASIC checks to see if all the keywords are spelled correctly, and translates them into upper case. It makes sure that all function calls are immediately followed by an open parenthesis ["("]. BASIC also makes checks of the line for simple syntax errors. If the line is determined to be incorrect, the message:

```
SYNTAX ERROR
```

will be typed on the screen, and the line will not be inserted into your program. Note that this preliminary syntax check will not detect all possible errors. BASIC may accept the line when you type it and then detect an error later when you execute your program.

LINE LENGTH

A line in Extended Benton Harbor BASIC is restricted to 100 characters.

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 <RTN>. You can leave blank lines by pressing the space bar once for each blank line before pressing <RTN>.

GENERAL TEXT RULES (Cont)

USING A LINE PRINTER WITH BASIC

You can access LP: (or whatever other two-letter name you have assigned to your line printer driver) from within BASIC in order to obtain program listings or printed program output. For example, if you wanted computations or other such results of programs to be listed on the line printer instead of the console terminal, you would use the command OPEN "LP:" FOR WRITE AS FILE #. The general format of this command is outlined in the sample application as follows:

```
"*"'10 OPEN "LP:" FOR WRITE AS FILE#1'
"*"'20 FOR I=1 TO 10'
"*"'30 PRINT#1, I'
"*"'40 NEXT I'
"*"'50 CLOSE#1'
'60 END'

"*"
```

This program will print the numbers 1 through 10 on LP:.

To obtain a listing of a BASIC program that is currently in memory, type REPLACE "LP:".

****

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 on the screen. 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<RTN>'
"*"'! ERROR - ATTEMPTED DIVIDE BY ZERO'
"*"'*10PRINT 1/0<RTN>'
"*"'RUN<RTN>'
"! ERROR - ATTEMPTED DIVIDE BY ZERO AT LINE 10"
"*"
```

ERRORS (Cont)

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, simply type the line number, and follow it with a <RTN>.

RECOVERING FROM ERRORS

When it detects an error, 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 the program text is modified in any way, the GOSUB and FOR stacks are purged. If an error occurred in a GOSUB routine for a FOR-loop, the entire program must be restarted.

ERROR MESSAGES

The following Table 12-1, Error Messages, describes the error messages generated by Extended Benton Harbor BASIC. This error table discusses only those errors which are detected directly by BASIC. They are printed in the BASIC error message format described. For details on the HDOS 3.02 system error messages, refer to Chapter 3, "System Optimization," Appendix 3-A:, page 3-33.

****

TABLE 12-1: - ERROR MESSAGES

AN ILLEGAL CHARACTER WAS ENCOUNTERED
This message indicates a syntax error in the line. BASIC saw a character that was illegal to have in a BASIC statement.

ATTEMPTED DIVIDE BY ZERO
Your program tried to divide a number by zero.

CAN'T FIND VARIABLE NAME MENTIONED IN NEXT STATEMENT
BASIC has not seen a matching FOR-loop for the variable named in the NEXT statement. This error can be caused by improper FOR-loop nesting.

CTL-B STRUCK
The CTRL-B key was struck and no CTRL-B line number has been set up. Refer to the CNTRL 0, n command for more information.

CTL-C STRUCK
The CTRL-C key was struck, interrupting the program.

DATA EXHAUSTED
A READ statement was executed when there was no data remaining in DATA statements to satisfy the READ. You either have too many READ requests, or too few DATA statements.

DATA LOCK ENGAGED
This operation is illegal when BASIC is in the LOCKed state. See the LOCK and UNLOCK commands for more information.

END
Your program executed an END statement. This is a normal way of terminating execution, and not an error. If your program has no END statement, BASIC will invent one at line 65535.

FILE ALREADY EXISTS
You tried to SAVE to a filename which is already present on that device. Either SAVE to a different filename, UNSAVE (delete) the existing filename, or use the REPLACE command.

FILE IS NOT OPEN
You tried to do file I/O(PRINT #chan, or INPUT #chan) to a channel which has no open file associated with it. You must OPEN a file before it can be used for INPUT or PRINTing.

TABLE 12-1: - ERROR MESSAGES

FLOATING POINT OVERFLOW (Number too large)
An arithmetic calculation produced a number larger than 1 X 10^37.

ILLEGAL FORMAT FOR FILE NAME
A filename specified in an OPEN statement contained too many characters to be valid. There should be no blanks or extraneous characters in a filename string.

ILLEGAL NUMBER VALUE
A number appeared in an illegal format or syntax. If this error occurs during a READ or INPUT statement, check the value being READ or INPUTted to see if it is valid.

ILLEGAL OR UNKNOWN STATEMENT NUMBER
A reference was made to a statement that does not exist, or to an illegal statement number. Statement numbers must be between 1 and 65534.

ILLEGAL USAGE
This statement may not be used in this mode. You have tried to use an "execution mode only" command in immediate mode, or an "immediate mode only" command in an executing program.

NO CORRESPONDING GOSUB FOR THIS RETURN STATEMENT
A RETURN statement was encountered when the GOSUB stack was empty.

OUT OF RAM SPACE
There is insufficient free RAM space to continue with this program. This error usually occurs when you DIMension a large array. If you cannot determine the cause of the memory overflow, use the FREE command to display the amounts being used by the various tables.

FOR HDOS 2.0 AND EARLIER: If you have specified CNTRL 4,1, you can free up some RAM space by specifying CNTRL 4,0.

STOP
A STOP statement was encountered. This is a normal condition, and not an error.

STRING LENGTH EXCEEDS 255 CHARACTERS
The maximum length of a string in BASIC is 255 characters.

SUBSCRIPT OUT OF RANGE
The program specified a subscript value larger than the declared limit for that dimension. Either your array was declared too small, or your program incorrectly computed the subscript.

TABLE 12-1: - ERROR MESSAGES

SYNTAX ERROR

There is an error in the statement's syntax.

TOO MANY OR TOO FEW ARGUMENTS SPECIFIED

An incorrect number of arguments was specified for a call to a built-in function, or a user-defined function.

TOO MANY OR TOO FEW SUBSCRIPTS SUPPLIED

The number of subscripts in the array reference do not match the number of dimensions declared.

TYPE CONFLICT (Illegal mix of string and number values)

The program attempted an operation illegally mixing string and number values, or supplied a numeric argument to a function requiring a string argument, or vice-versa. This error can also occur if you try to INPUT or READ a string value into a numeric variable.

UNDEFINED FUNCTION

This user-defined function has not been defined. Your program must execute the DEF FN statement before you attempt to call that function.

NOTE: BASIC may also call up some HDOS Operating System Error Messages. For details concerning the system error messages, refer to Chapter 3, System Optimization, Appendix 3-A:, "Error Messages."

APPENDIX 12-A: - SUMMARY OF B.H. BASIC

NUMERIC DATA

NOTE: The terms "10^-38" and "10^+37" shown below represent the number of 10 to the negative 38th power and 10 to the positive power, respectively. It is impossible to show this relationship per standard methods using the H89 Computer, therefore a brief explanation of the arbitrary technique used herein is required.

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 .... (+ or -)X.XXXXXE(+ or -)NN
```

BOOLEAN DATA

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

Data is all printed in ASCII characters plus the BELL, BLANK, TAB, and FORM-FEED, with the following characteristics:

```
Maximum string length . 255 characters
Inclosure ............. Quotation marks (") on both ends
Multiple lines ........ Not allowed for a single string
```

VARIABLES

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.

SUBSCRIPTED VARIABLES

Subscripted variables are named like variables, but are followed by dimensions in parentheses. Subscripted variables are of the form:

An(N1,N2, . . . Nx) 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.

APPENDIX 12-A: - SUMMARY OF B.H. BASIC (Cont)

ARITHMETIC OPERATORS

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.

SYMBOL	EXPLANATION	
-	Unary negation logical complement	
^	Exponentiation	
* /	Multiplication	Division
+ -	Addition	Subtraction

RELATIONAL OPERATORS

SYMBOL	EXPLANATION
=	Equal to
<	Less than
< =	Less than or equal to
>	Greater than
> =	Greater than or equal to
< >	Not equal to

BOOLEAN OPERATORS

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

String variables may be either subscripted or nonsubscripted. They take the same form as Booleran 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).

APPENDIX 12-A: - SUMMARY OF B.H. BASIC (Cont)

STRING OPERATORS

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.

THE COMMAND MODE

The command mode does not use line numbers. Statements are executed when a <RTN> is typed.

LINE NUMBERS

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.

MULTIPLE STATEMENTS ON ONE LINE

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.

COMMAND MODE STATEMENTS:

COMMAND	FORM	DESCRIPTION
BUILD	BUILD iexp1, iexp2	Automatically generates program line numbers starting at iexp1 in steps of iexp2.
BYE	BYE	Exits BASIC, returns to the HDOS command mode.
CONTINUE	CONTINUE	Resumes program execution.
DELETE	DELETE [iexp1,iexp2]	Deletes program lines between iexp1 and iexp2.
LIST	LIST [iexp1][,iexp2]	Lists the entire program on the console terminal. Lists the line iexp1, or the range of lines iexp1 through iexp2.

APPENDIX 12-A: - SUMMARY OF B.H. BASIC (Cont)

COMMAND MODE STATEMENTS: (Cont)

COMMAND	FORM	DESCRIPTION
OLD	OLD "fname"	Loads file "fname" into BASIC. Clears variables.
REPLACE	REPLACE "fname"	Saves current program as file "fname." Replaces "fname" if it already exists.
RUN	RUN	Starts execution of current program. Preclears all variables, stacks, etc.
SAVE	SAVE "fname"	Saves current program as file "fname." Will not replace any pre-existing "fname."
SCRATCH	SCRATCH SURE?Y<RTN>	Clears all program and data storage area. Any response to "SURE?" but "Y" cancels SCRATCH.

COMMAND AND PROGRAM MODE STATEMENTS:

COMMAND	FORM	DESCRIPTION
CHAIN	CHAIN "fname"[,linnum]	Loads new program "fname" into BASIC and continues execution at linnum. If no line number is specified, starts execution at first line number. Does not affect variables or open files.
CLEAR	CLEAR[varname]	Clears all variables, arrays, string buffers, etc. Optionally clears named variable [varname]. Specifies functions and arrays as V(.
CLOSE	CLOSE #chan 1 [,#chan n]	Close an HDOS file. "#chan" is the number assigned to the opened file.
CONTROL	CNTRL iexp1,iexp2	CNTRL 0 sets a GOSUB to line iexp2 when a CTRL-B is typed. CNTRL 1 sets iexp2 digits before exponential format is used.

APPENDIX 12-A: - SUMMARY OF B.H. BASIC (Cont)

COMMAND AND PROGRAM MODE STATEMENTS (Cont)

COMMAND	FORM	DESCRIPTION
CONTROL (Cont)	CNTRL iexp1,iexp2 (Cont)	CNTRL 2 controls the H8 front panel. If iexp2=0, display off; if iexp2=1, display on without update; if iexp2=2, display on with update. (NOTE: Has no effect on the H89.) CNTRL 3 sets the width of a print zone to iexp2 columns. CNTRL 4 controls the state of the HDOS operating system overlay: iexp2=0, swap overlay, iexp=1, keep overlay in memory. (Use in Command Mode only.)
DIMENSION	DIM varname(iexp1 [, ... iexpn]) [,varname2(...)]	Defines the maximum size of variable arrays.
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.
FREE	FREE	Displays the amount of memory assigned to tables and text.
FREEZE	FREEZE"fname"	Saves BASIC interpreter, current program, and data values on file "fname." All files must be closed before FREEZE is executed.
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.
GOTO	GOTO iexp	Unconditionally transfers the program execution sequence to the line iexp.

APPENDIX 12-A: - SUMMARY OF B.H. BASIC (Cont)

COMMAND AND PROGRAM MODE STATEMENTS (Cont)

COMMAND	FORM	DESCRIPTION
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.
LET	LET var=nexp LET var$ = sexp	Assigns the value nexp (or sexp in the case of strings) to the variable var (or var$). LET keyword is optional.
LOCK	LOCK	Protects your program by preventing you from executing the BUILD, BYE, CHAIN, UNFREEZE, DELETE, RUN, SCRATCH, and CLEAR command mode statements. Also prevents the entry or deletion of program text.
ON/GOSUB	ON iexp1 GOSUB iexp2, ... ,iexpn	Permits a computed GOSUB. Iexp1 is evaluated and acts as an index to line numbers iexp2 through iexpn, each pointing to a different subroutine.
ON/GOTO	ON iexp1 GOTO iexp2,...,iexpn	Permits a computed GOTO. Iexp1 is evaluated and acts as an index to line numbers iexp2 through iexpn.
OPEN	OPEN sexp FOR READ AS FILE #iexp OPEN sexp FOR WRITE AS FILE #iexp	Opens file for read/write operations. "sexp" is a string expression for the filename. "#iexp" is the channel number assigned to the file to be opened.
OUT	OUT iexp1, iexp2	Outputs a number iexp2 to output port iexp1.
PAUSE	PAUSE(iexp)	Ceases program execution until a console terminal key is typed. Ceases program execution for 2 X iexp mS.
POKE	POKE iexp1,iexp2	Writes a number iexp2 into memory at location. CAUTION: This command could damage or destroy your operating system if used improperly!

APPENDIX 12-A: - SUMMARY OF B.H. BASIC (Cont)

COMMAND AND PROGRAM MODE STATEMENTS (Cont)

COMMAND	FORM	DESCRIPTION
PRINT	PRINT [#chan.] (nexp1 sep1 ... nexpn (sepn)	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 on a new line. #chan specifies channel to write line to HDOS file. If no #chan is specified, line goes to console terminal screen.
READ/DATA	READ var1,...,varn DATA exp1,..,expn	The READ statement assigns the values exp1 through expn in the data to the variables var1 through varn.
REMARK	REM	Text following the REM is not executed and is used for commentary only.
RESTORE	RESTORE	Causes the program to reset the DATA pointer, thus reusing data at the first DATA statement.
STEP	STEP iexp	Executes iexp lines of the program. Then returns BASIC to the command mode.
UNFREEZE	UNFREEZE "fname"	Restores BASIC program and variables from previously created FREEZE file.
UNLOCK	UNLOCK	Aborts the LOCK mode and restores the use of all command mode statements.
UNSAVE	UNSAVE "fname"	Deletes programs or files from the disk.

APPENDIX 12-A: - SUMMARY OF B.H. BASIC (Cont)

PROGRAM MODE STATEMENTS:

COMMAND	FORM	DESCRIPTION
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[#chan,] prompt;var1,..,varn	Reads data from the console terminal, or from the HDOS file open on channel "chan," if #chan is specified. String data must be inclosed in quotes if it contains any commas.
LINE INPUT	LINE INPUT [#chan,] prompt; stringvar	Reads string data from the console terminal, or from the HDOS file open on channel "chan," if #chan is specified. Data should not be inclosed in quotes; entire line is read into string variable.
STOP	STOP	Causes BASIC to enter the command mode when the statement containing STOP is executed.

PREDEFINED FUNCTIONS

FUNCTION	DEFINITION
ABS (nexp)	Returns the absolute value of nexp.
ASC (sexp)	Returns the ASCII code for the first character in the string, sexp.
ATN (nexp)	Returns the arctangent of nexp in radians.
CHR$ (iexp)	Returns the ASCII character iexp.
CIN (chan)	Reads a character from any open file, or from the console terminal (if chan=0). If the value returned is positive, a character was read. If the value is negative, an end of file, or no line was read.

APPENDIX 12-A: - SUMMARY OF B.H. BASIC (Cont)

PREDEFINED FUNCTIONS (Cont)

FUNCTION	DEFINITION
COS (nexp)	Returns the cosine of nexp in radians.
EXP (nexp)	Returns e nexp, where "nexp" is an exponent.
INT (narg)	Returns the integer value of narg.
LEFT$(sexp,iexp)	Returns the left iexp characters of the string sexp.
LEN (sexp)	Returns length of string expression sexp.
LNO (iexp)	Converts iexp to a line number.
LOG (nexp)	Returns the natural logarithm of nexp.
MATCH (sexp1,sexp2,iexp)	Finds the first occurrence of the substring sexp2 in sexp1, starting at the iexp th character in sexp1. Returns index of start of substring if found, 0 if not found.
MAX (nexp1,..,nexpn)	Returns the maximum value of expressions nexp1 through nexpn.
MID$ (sexp,iexp1) [,iexp2]	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.
MIN (nexp1,...,nexpn)	Returns the minimum value of expressions nexp1 through nexpn.
PAD (0)	Returns the value of the H8 front panel key pressed. Includes key debounce. Returns a 0 on an H89.
PEEK	Returns the numeric value at memory location iexp.
PIN (iexp)	Returns the data input from port iexp.
POS (chan)	Returns the current file or console printhead (cursor) position by column number.

APPENDIX 12-A: - SUMMARY OF B.H. BASIC (Cont)

PREDEFINED FUNCTIONS (Cont)

FUNCTION	DEFINITION
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.
RIGHT$ (sexp,iexp)	Returns the right iexp characters of the string sexp.
SEG (narg)	Returns the correct eight-bit number to display narg (0-9) on the H8 LEDs. (Has no effect on the H89.)
SGN (narg)	Returns +1 if narg is positive. Returns -1 if narg is negative. Returns 0 if narg is zero.
SIN (nexp)	Returns the sine of nexp in radians.
SPC (iexp)	Positions printhead (cursor) iexp columns to the right.
SQR (narg)	Returns the square root of narg.
STR$ (narg)	Returns narg encoded into ASCII with leading and trailing blanks, as in the print statement.
TAB (iexp)	Position printhead (cursor) to the iexp th column.
TAN (nexp)	Returns the tangent of nexp in radians.
VAL (sexp)	Returns the numeric value of the number encoded in the string.

APPENDIX 12-A: - SUMMARY OF B. H. BASIC (Cont)

ALPHABETICAL LISTING OF FUNCTIONS AND STATEMENTS

APPENDIX 12-A: - SUMMARY OF B. H. BASIC (Cont)

ALPHABETICAL LISTING OF FUNCTIONS AND STATEMENTS (Cont)

APPENDIX 12-A: - SUMMARY OF B. H. BASIC (Cont)

ALPHABETICAL LISTING OF FUNCTIONS AND STATEMENTS (Cont)

APPENDIX 12-B: - ASCII CODES
DECIMAL TO OCTAL TO HEX TO ASCII CONVERSION

DECIMAL	OCTAL	HEX	ASCII	CTRL-n	DECIMAL	OCTAL	HEX	ASCII
0	000	00	NUL	CTRL-@	48	060	30	0
1	001	01	SOH	CTRL-A	49	061	31	1
2	002	02	STX	CTRL-B	50	062	32	2
3	003	03	ETX	CTRL-C	51	063	33	3
4	004	04	EOT	CTRL-D	52	064	34	4
5	005	05	ENQ	CTRL-E	53	065	35	5
6	006	06	ACK	CTRL-F	54	066	36	6
7	007	07	BEL	CTRL-G	55	067	37	7
8	010	08	BS	CTRL-H	56	070	38	8
9	011	09	HT	CTRL-I	57	071	39	9
10	012	0A	LF	CTRL-J	58	072	3A	:
11	013	0B	VT	CTRL-K	59	073	3B	;
12	014	0C	FF	CTRL-L	60	074	3C	<
13	015	0D	CR	CTRL-M	61	075	3D	=
14	016	0E	SO	CTRL-N	62	076	3E	>
15	017	0F	SI	CTRL-O	63	077	3F	?
16	020	10	DLE	CTRL-P	64	100	40	@
17	021	11	DC1	CTRL-Q	65	101	41	A
18	022	12	DC2	CTRL-R	66	102	42	B
19	023	13	DC3	CTRL-S	67	103	43	C
20	024	14	DC4	CTRL-T	68	104	44	D
21	025	15	NAK	CTRL-U	69	105	45	E
22	026	16	SYN	CTRL-V	70	106	46	F
23	027	17	ETB	CTRL-W	71	107	47	G
24	030	18	CAN	CTRL-X	72	110	48	H
25	031	19	EM	CTRL-Y	73	111	49	I
26	032	1A	SUB	CTRL-Z	74	112	4A	J
27	033	1B	ESC	CTRL-[	75	113	4B	K
28	034	1C	FS	CTRL-\	76	114	4C	L
29	035	1D	GS	CTRL-]	77	115	4D	M
30	036	1E	RS	CTRL-^	78	116	4E	N
31	037	1F	US	NOTE 1	79	117	4F	O
32	040	20	SPACE		80	120	50	P
33	041	21	!		81	121	51	Q
34	042	22	"		82	122	52	R
35	043	23	#		83	123	53	S
36	044	24	$		84	124	54	T
37	045	25	%		85	125	56	U
38	046	26	&		86	126	56	V
39	047	27	'		87	127	57	W
40	050	28	(		88	130	58	X
41	051	29	)		89	131	59	Y
42	052	2A	*		90	132	5A	Z
43	053	2B	+		91	133	5B	[
44	054	2C	,		92	134	5C	\
45	055	2D	-		93	135	5D	]
46	056	2E	PERIOD		94	136	5E	^
47	057	2F	/		95	137	5F	_

APPENDIX 12-B: - ASCII CODES
DECIMAL TO OCTAL TO HEX TO ASCII CONVERSION (Cont)

DECIMAL	OCTAL	HEX	ASCII	DECIMAL	OCTAL	HEX	ASCII
96	140	60	`	112	160	70	p
97	141	61	a	113	161	71	q
98	142	62	b	114	162	72	r
99	143	63	c	115	163	73	s
100	144	64	d	116	164	74	t
101	145	65	e	117	165	75	u
102	146	66	f	118	164	76	v
103	147	67	g	119	167	77	w
104	150	68	h	120	170	78	x
105	151	69	i	121	171	79	y
106	152	6A	j	122	172	7A	z
107	153	6B	k	123	173	7B	{
108	154	6C	l	124	174	7C	|
109	155	6D	m	125	175	7D	}
110	156	6E	n	126	176	7E	~
111	157	6F	o	127 NOTE 2	177	7F	DELETE

NOTES:

NOTE 1: DECIMAL 31: Use the CTRL-SHIFT-HYPHEN keys with the H89.

NOTE 2: DECIMAL 127: Use the DELETE key with the H89.

NOTE 3: The following notes explain the abbreviations under the ASCII column of the table shown on page 12-95. Many of the expressions are carryovers from the old days when it was common to use TWX machines and teletypes to transmit information.

NOTE 4: The data included in the Decimal to Octal to Hex to ASCII Table enables one to convert from one kind of notation to another and retain the same values. For example, if you have Hex 9, the equivalent Octal and ASCII code are 11 and CTRL-I, respectively. (Ignore the HT code in the same line.) Similiarily, CTRL-V is equivalent to Octal 26 and Hex 16, respectively. (Ignore the SYN code.) Notice that you can go in either direction.

From time to time one has a need for notation conversion. For example, one of the major uses of this conversion table is to obtain "control codes" for your printer. These codes are obtained from your printer manual. Once you decide what you want to do, you determine the codes to use to accomplish your objective, such as making a title print out in "double high" or "emphasized," type, etc. Then you simply type the control codes into your manuscript in accordance with the particular needs of the editor you are using. For example, EDIT19 requires control codes to be input in HEX notation, while TXTPRO requires codes to be input in ASCII notation. By consulting this table, it should be obvious as to how to perform the aforementioned task.

APPENDIX 12-B: - ASCII CODES
DECIMAL TO OCTAL TO HEX TO ASCII CONVERSION (Cont)

ASCII CODE EXPLANATIONS

NULL -- Null, Tape Feed
SOH --- Start of Heading; Start of Message
STX --- Start of Text; End of Address
ETX --- End of Text; End of Message
EOT --- End of Transmission; Shuts off TWX Machines
ENQ --- Inquiry; WRU
ACK --- Acknowledge; RU
BEL --- Rings Terminal Bell
BS --- Backspace; Format Effector
HT ---- Horizontal TAB
LF ---- Line-Feed or Space (New Line)
VT ---- Vertical TAB
FF ---- Form-Feed
CR ---- Carriage Return
SO ---- Shift Out
SI ---- Shift In
DLE --- Data Link Escape
DC1 --- Device Control 1; Reader On
DC2 --- Device Control 2; Punch On
DC3 --- Device Control 3; Reader Off
DC4 --- Device Control 4; Punch Off
NAK --- Negative Acknowledge; Error
SYN --- Synchronous Idle (SYNC)
ETB --- End of Transmission Block; Logical End of Medium
CAN --- Cancel (CANCL)
EM ---- End of Medium
SUB --- Substitute
ESC --- Escape
FS ---- File Separator
GS ---- Group Separator
RS ---- Record Separator
US ---- Unit Separator

APPENDIX 12-C: - SUPPLEMENTAL REFERENCES

For additional information on Heath Extended Benton Harbor BASIC, refer to the following articles in REMark Magazine, the official Heath/Zenith publication.

[1] REMark Issue 7, Published Quarterly, No Date, 1979, Page 3

"ORGANIZATION! Confusion with A Structured Defintion," by Gene Bellinger. 6 pages. Presents a logical method of conceptualizing the structure of composing a BASIC program. Discusses all the factors involved. Attempts to outline a logical approach.

[2] REMark Issue 7, Published Quarterly, No Date, 1979, Page 12

"A Menu for BASIC Programs," by Douglas H. McNeill, M.D. 0.5 pages. Describes a program which prints a menu of all the BASIC programs on your disk, and then allows you to select a number which loads and runs the program of your choice.

[3] REMark Issue 19, August 1981, Page 7

"Menu Driven Demo Program," by Gene Sevin. 2 pages.
Demonstrates a technique in Extended Benton Harbor BASIC for selecting menu-driven program options, using H89/H19 graphics, function keys, cursor addressing, the 25th line, and the HDOS type-ahead buffer.

[4] REMark Issue 19, August 1981, Page 18

"Using the HDOS Type-Ahead Buffer," by Patrick Swayne. 2 pages. Shows the user how to POKE commands into the HDOS type-ahead buffer from a BASIC program. The effect, of course, is the same as if the commands had been typed from the console.

[5] REMark Issue 29, June 1982, Page 5

"IMPROVEMENTS TO B.H. BASIC," by Patrick Swayne. 8.5 pages
Provides a means of calling machine language subroutines from B.H. BASIC, provides a means of inputting single characters without hitting the RETURN, and provides a patch for FREEZE and UNFREEZE commands so that they load programs in a compressed format, thus speeding up the performance.

[6] REMark Issue 30, July 1982, Page 28

"AN EDITOR FOR B.H. BASIC," Patrick Swayne. 4.5 pages
Developes the topics from REMark 29, and adds an assembly coded line editor for B.H. BASIC.

[7] REMark Issue 39, April 1983, Page 11

"A FASTER B.H. BASIC," by Dahl B. Metters. 6.5 pages with listing
Tells how to increase execution speed by a factor of 200%.

APPENDIX 12-C: - SUPPLEMENTAL REFERENCES (Cont)

[8] REMark Issue 41, June 1983, Page 23

"USING BINARY FILES WITH B.H.BASIC," by David A. Sandage.
Developes the use of the .CIN command to incorporate binary files.

[9] REMark Issue 46, November 1983, Page 66

Letter, Shows how to shorten the technique required to send data to printer, disk, and screen simultaneously. Randall Stokes

NOTE

In case the reader desires to order copies of REMark Magazine, contact "Heath Users' Group, P.O. Box 217, Benton Harbor, MI 49022-0217 (616)982-3463.

INDEX

INDEX (Cont)

INDEX (Cont)

INDEX (Cont)

INDEX (Cont)

HDOS SOFTWARE REFERENCE MANUAL

HDOS DISK OPERATING SYSTEM

VERSION 3.02

CHAPTER 13

HDOS PROGRAMMERS' REFERENCE MANUAL

HEATH DISK OPERATING SYSTEM

SOFTWARE REFERENCE MANUAL

VERSION 3.02

 Through the years it continued to be improved by successive revisions which included 1.5, 1.6, and finally 2.0. It was entered into public domain on 19 July 1989 per letter by Jim Buszkiewicz, Managing Editor, Heath Users' Group, P.O. Box 217, Benton Harbor, MI 49022-0217 (616)982-3463. A copy of this letter is available for public inspection. Indeed, HDOS is still alive and well!

This manual is indicative of further improvements and provides for the latest revision, HDOS 3.0 and HDOS 3.02. Revision 3.0 is detailed in chapters 1, 2, and 3, while chapters 4, 5, 6, 7, 8, 13 and 14, are related to revision 3.02. Chapters 9 through 12, with minor improvements, are essentially picked up from the original HDOS 2.0 manual.

Chapter 13, The HDOS Programmers' Reference Manual, is intended for the advanced programmer. This chapter calls out and describes all of the new .SCALLS for HDOS 3.02. You will find some references to HDOS 2.0 and its overlays in this chapter. These should be ignored.

SPECIAL DISCLAIMER: The Heath Company cannot provide consultation on either the HDOS Operating System or user-developed or modified versions of Heath software products designed to operate under the HDOS Operating System. Do not refer to Heath for questions.

Instead, you are invited to direct any questions concerning the Heath Disk Operating System (HDOS) to Mr. Kirk L. Thompson, Editor "Staunch 89/8" Newsletter, P.O. Box 548, #6 West Branch Mobile Home Village, West Branch, IA 52358.

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont)

PART 1 - INTRODUCTION

Purpose

This manual describes the advanced features of HDOS version 3.0 and 3.02 that are necessary for a user program to interface with HDOS at the assembly language level. This information is provided for the advanced programmer. References to overlays pertain to HDOS 2.0.

Data and information provided in this manual applies equally to the H8, H89, or Z90 family of computers.

Background

Chapter 11, ASM, of this "HDOS Software Reference Manual" documents the various system commands and BASIC statements used to generate and maintain files at the higher language level. At this level, the novice or average programmer need not be concerned about the involved details of interfacing his programs with HDOS or the disk drives.

Preface

HDOS provides a full run-time support environment for assembly language programs. Communications with file-oriented devices, console communications, memory allocation, and other such services are provided by the HDOS system. Since the H8 does not afford any hardware protection, assembly language routines must be "polite," in that they should not damage the H8 running environment. This subject will be discussed in greater detail further on in this document.

File Names

Since many SCALLs require file names as arguments, this is a good time to discuss HDOS file names.

In general, when you supply a file name as an argument to a SCALL, you point to an ASCII string containing the file descriptor just as the user would have typed it. The line should be terminated with a delimiter of some sort, usually a comma, blank, or zerozero (00) character. For example, the following are examples of valid file names:

```
        DB      'SY0:MYFILE.TMP',0
        DB      'TMP',0
        DB      'BASIC.SAM,'
```

PART 1 - INTRODUCTION (Cont)

File Names (Cont)

NOTE: The expression ',' delimits the file name.

Of course these names are all shown being assembled into the program. You might just as well have read them from the user's console, or generated the names on the keyboard. They must not have imbedded 00 bytes or blanks in the names.

Also note that some of the examples shown do not specify an extension or a device. All SCALLs that take file names as arguments also require a default block. This block is a 6-byte area containing the default device specification and a default extension specification. A typical default block is:

```
         DB      'SY0 TMP'
```

which yields a default device of SY0: and extension of TMP. Another common block is:

```
         DB      'SY0',0,0,0
```

which indicates that there is no default extension. File descriptors not specifying a name will generate a file with a null extension.

****

PART 2 - RUN-TIME ENVIRONMENT

Memory Layout

HDOS contains many useful general-purpose subroutines which may be called by user programs. These, together with the system services provided, make assembly language programming under HDOS very convenient.

When you type "RUN fname," HDOS will load your program into memory and run it. This section will discuss the initial run-time environment of the program. Refer to the memory map in the HDOS Software Reference Manual, Chapter 8, Appendix 8-A, and Appendix 13-D, for further data that will help you to understand the information provided in this chapter.

The first 64 bytes of RAM from 040000 to 040100 are used by PAM-8. The PAM-8 source listing documents their use.

PART 2 - RUN-TIME ENVIRONMENT (Cont)

Memory Layout (Cont)

NOTE

Although the PAM-8 ROM will be referenced throughout this manual, the general-purpose routines of the MTR -88, MTR-89, MTR-90, PAM-8-GO, and XCON-8 ROMs all have common entry points. For specific information, refer to the data in the manual covering the ROM that you are using.

The next 295 bytes are used by HDOS and the disk device driver for work cells. These cells are in low memory, so that HDOS can reference them without having to compute relocation factors (HDOS is relocatable in low memory). Some of the contents of these cells are of interest to assembly language programmers and are indirectly available through HDOS system calls. You should refrain from accessing them directly, since their position may change with future releases. Use of the proper HDOS symbols and system calls in assembly language programs will make it possible to transport your programs to future Heath CPUs executing an advanced HDOS system. There are a few cells that may be of interest to the programmer, and these are documented in Part 6. They may be read, but never written to.

Following the work cell area is a 279-byte stack area. When a user program is executed, the stack pointer is set to the symbol STACK, which is 042200A. Note that you may not set your stack pointer below that address and then use the area below 42200A for code or data (other than that stored by a normal PUSH). You may make the stack larger, setting SP to a value larger than 042200A. Calls to the HDOS system will preserve this larger stack.

The user program area starts at 042200A, immediately after the top of the stack. The user program extends until the last byte loaded by the RUN command. Note that the assembler generates a dummy 00 byte as the last statement in a program, so that trailing DS declarations will be contained in the size of the running program. There is a system call which requests access to more memory. You must issue the call first, since HDOS may be using that area for its own code.

Any active device drivers are loaded immediately before the resident HDOS code. A device driver is loaded when a file is opened on a device whose driver is not yet in memory. The TT: device driver is automatically loaded during the boot process, and never needs to be loaded. Since the SY: driver is permanently loaded into memory when the system is first booted up, it never needs to be loaded.

PART 2 - RUN-TIME ENVIRONMENT (Cont)

Memory Layout (Cont)

Finally, the HDOS system resides in low memory. When the system is booted up, HDOS initially loads at a fixed lower address. After sizing memory, HDOS moves its permanently resident parts into low memory. This section contains the SCALL Dispatcher and the handlers for all standard SCALL functions.

I/O Environment

TABLE 13-1
PORT ASSIGNMENTS FOR THE H89 AND H8

PORT	H89 COMPUTER **** BOTH ***** H8 COMPUTER
170-173Q (078-078H)	---- H37, H47, or H67 Disk Drive Systems ----
174-177Q (07C-07FH)	---- H17, H47, or H67 Disk Drive Systems ----
300-307Q (0C0-0C7H)	---------- Reserved ----------
320-327Q (0D0-0D7H)	H88-3 ------- Alternate Terminal ------- H8-4
330-337Q (0D8-0DFH)	H88-3 ---------- Modem ---------- H8-4
340-347Q (0E0-0E7H)	H88-3 ---------- Line Printer ---------- H8-4
350-357Q (0E8-0EFH)	CPU ---------- Console Terminal -------- H8-4
360-361Q (0F0-0F1H)	---- Reserved ---- --- H8 Front Panel ---
370-371Q (0F8-0F9H)	H88-5 ---------- Cassette ---------- H8-5
372-373Q (0F2-0FBH)	---- Reserved ---- ----- Console Terminal
374-375Q (0FC-0FDH)	---- Reserved ---- --- Alternate Terminal
376-377Q (0FE-0FFH)	---- Reserved ---- ------ Reserved -----

HDOS has a vested interest in the I/O ports being used by the device drivers currently in memory. These ports should not be disturbed when HDOS or a device driver may be trying to use them. The ports are listed in Table 13-1.

Since the TT: and SY: drivers are permanently resident, it is vital that you do not disturb the TT: and SY: ports. Disturbing the SY: port will destroy your system disk. Disturbing the TT: port will damage the console driver package. The console driver package communicates with the console device at interrupt time, so you will not be able to detect character entry by examining the console status bits. HDOS provides you with a facility to test the presence of a console character.

Interrupt Environment

HDOS is an interrupt-driven system, so be careful how you handle interrupts. Your program must not turn off interrupts via the DI

PART 2 - RUN-TIME ENVIRONMENT (Cont)

Interrupt Environment (Cont)

instruction for other than very short periods of time. The H17 device driver makes use of the front panel clock interrupts on the H8 computer, so you must not disable them, either directly via port 360Q or by the PAM-8 control word. Likewise, console interrupts are used by the system console handler, and should not be disturbed. HDOS does not currently support any interrupt-driven device drivers except the H37 device driver, but programs may still make use of interrupts. There are two major trouble areas in this: choosing a vector and discontinuing the interrupts.

Interrupt Vectors

Of the eight interrupt vectors available in an 8080A microprocessor (sometimes abbreviated "processor") HDOS makes use of six or seven of them.

```
0 ----- Master Clear.  Returns control to PAM-8.
1 ----- Clock Interrupts.
2 ----- Single-Step.  Used by DBUG.  May be used by a user
          program when not running DBUG.
3 ----- Console Interrupts.
4 ----- Reserved for H37.
4 ----- Reserved for H14 Printer.
5 ----- Reserved for modems.
5 ----- Available for user programs.
6 ----- Available for user programs.
7 ----- HDOS SCALL vector.
```

Set the vectors by storing a JMP to your interrupt service routine in the PAM-8 ".UIVEC" area, as discussed in the PAM-8 manual.

Discontinuing Interrupts

When a user program causes a device to start issuing interrupts, it must somehow turn off that device before control returns to the system. HDOS will not alter the interrupt vector (JMP) in PAM-8's "UIVEC," and an interrupt occurring after your program has been removed from the system will be tragic. Also note that as a user, you must be careful of typing CTRL-Z, as this can kill your program before it can shut down any interrupting devices.

PART 2 - RUN-TIME ENVIRONMENT (Cont)

Discontinuing Interrupts (Cont)

NOTE

You MUST turn off the device interrupts before surrendering control to HDOS. Simply replacing your interrupt vector with EI and RET instructions will cause disaster, since the interrupting device will continue to request interrupts until it is serviced, and HDOS does not know how to service it. Your computer will then hang in an interrupt service loop.

CPU Environment

After loading your program, HDOS transfers control to the program's entry point. This is the address specified in the END (assembler) pseudo.

Channel Environment

HDOS allows user programs to communicate with file-oriented devices via "channels." These channels are discussed in Part 3. In all cases, channel -1 (377Q) is open for read access on the device and file that the program was loaded from. This is done so you can conveniently load overlays without having to know under what name and disk drive your program was run from. If your program was run in response to a RUN command, all other channels will be closed. If your program was run in response to a .LINK SCALL, then the other channels will remain as they were set up by the program which issued the .LINK.

****

PART 3 - I/O CHANNELS

All file I/O in the HDOS system is done via I/O channels. "File I/O" refers to normal input/output done to HDOS devices via HDOS device drivers. Naturally, a program may control its "private devices" (those not suitable for device drivers) in any way it pleases.

In general, the sequence for doing file I/O is to issue an "open" SCALL (i.e., .OPENR, .OPENW, or .OPENU) to HDOS, supplying HDOS with the file descriptor as an ASCII string. HDOS will parse the string, load the device driver (if necessary), and open the file. When you issue the "OPEN" SCALL, you supply a channel number from -1 (i.e., 377Q) to 5.

PART 3 - I/O CHANNELS (Cont)

This channel number must not already be in use. This means that you may open a maximum of seven files simultaneously.

Once the file has been opened, you can perform I/O by using the .READ, .WRITE, and .POSIT SCALLs. Make these requests by supplying HDOS with the channel number of the file you want read or written. After the initial open, you no longer need the file descriptor string. Should you suddenly need that file name (for example, to issue an error message), HDOS provides the .NAME SCALL to recall the file name used when that channel was opened.

All disk file I/O is done in multiples of 256 bytes, the HDOS system sector size. As many bytes as desired may be transferred at one time, so long as the count is an integer multiple of 256. HDOS normally performs I/O in a sequential fashion. For example, if your program is reading from a disk file one sector (256 bytes) at a time, the first read will return sector 0, the next read sector 1, etc. For each open channel, HDOS maintains a "sector cursor," which indicates which sector in the file is next to be read or written. HDOS does provide the facility, via .POSIT, to randomly read and write sectors to or from a disk file by changing the value of this "sector cursor."

When you are done with the file, use the .CLOSE SCALL, once more supplying the channel number. HDOS will close the file and thus make that channel available for another open command.

NOTE

Although channel -1 may be used as a general purpose I/O channel, its use should normally be avoided. It is already open when your program is started, but you must close it before you can open a file on it. Also, channel -1 will be cleared (see the .CLEAR SCALL for details) if you use the .LINK SCALL. Thus, any file open for write on channel -1 at that time will be lost.

****

PART 4 - PRECAUTIONS

We have stated earlier in this document that the HDOS system does not provide any hardware protection, and thus is vulnerable to errors in assembly language programs. In order to help minimize this problem, this segment discusses the "Do's and Dont's" of assembly language programming in greater detail.

PART 4 - PRECAUTIONS (Cont)

Memory Precautions

The two most important areas of memory precautions are: respect for the user program area and maintenance of the stack.

User Memory Area

A user program must never write into memory outside of its domain. This "domain" consists of the memory area from 042200A (USERFWA) to the LWA of the user program area. When your program is first loaded, this LWA is set up to the end of your program and its declared data areas via DS, DW, or DB; not EQU. The ".SETTOP" SCALL is available to adjust this limit. User programs may adjust this limit as often as they like (for details, refer to the .SETTOP SCALL documentation). Note that HDOS may use all memory after this limit for a storage area, which is going to cause trouble if your routine also tries to use it.

Stack Maintenance

Since the HDOS system uses interrupts and requires interrupts to handle the console, the H17 disk, and the H37 disk, your program may be interrupted at any time. You must always maintain a valid stack pointer with at least 64 free bytes on the top of that stack. If you plan to fill the system stack area, then you should ORG your program above 042200A and set the stack pointer higher, giving yourself and HDOS a bigger stack. HDOS does not use a separate stack; it uses the top of the user program stack.

I/O PRECAUTIONS

As we discussed earlier, I/O precautions consist of keeping your INs and OUTs to yourself. Don't disturb the H17, the H37, and the H47, and don't disturb the console ports! Also, be careful what you do with the front panel ports on the H8, either directly or indirectly via PAM-8. These ports control the clock interrupts, which are necessary for the H17 device driver.

INTERRUPT PRECAUTIONS

When you are using interrupts, you must use only the available vectors, which are 4 (if you are not using the H37), 5 (if you are not

PART 4 - PRECAUTIONS (Cont)

INTERRUPT PRECAUTIONS (Cont)

using a modem), and 6. You may also use 2, if you will not be using DBUG. Before you enable your interrupting device, install the service vector in the appropriate ".UIVEC" location.

Most importantly, turn off the interrupting device so it cannot issue any more interrupts before you either return control to HDOS or CTL-Z out of the program. If an interrupt occurs when your program is no longer there to service it, the operating system and possibly the information on your diskettes will be destroyed.

Since console and clock interrupts may occur at any time, your program should not turn off interrupts via DI, except for very short periods of time.

Finally, HDOS uses the clock interrupts, so you should not overlay its interrupt vector. Programs desiring clock service should use all means possible to make do with the interrupt counter (PAM-8s .TICCNT). If you absolutely must have clock interrupts, save the address in the clock vector, install your own vector, and have your service routine exit the interrupt by jumping to the HDOS vector address. HDOS uses the clock interrupts for H17 timings; disturbing it might cause your motors to keep spinning, prematurely wearing the motors. Or worse, you might defeat the H17 driver's head-settle delays and cause a bad sector to be written.

CPU PRECAUTIONS

This precaution should be familiar to all assembly language programmers: don't let the CPU execute undefined memory locations. Should such a thing occur, it is unlikely that your disks will be damaged, due to some safeguards built into the system. However, you should immediately shift-reset and reboot, and not try to warm-start the system, since the CPU may have damaged tables in memory. Remember, the HDOS system uses a sophisticated linked-allocation scheme to handle disk files. Damaging that table or damaging the directory or allocation areas on the disk can cause all files on that disk to become lost; not just one or two!

If you are debugging a program which consistently vectors into undefined memory locations, then it is best to use write-protect labels on the disks. Then if you crash, you can quickly restart by using PAM-8 to start at the HDOS cold start address, 040100A. Entering at this address should return you to HDOS command mode. Do this only if you have your disks write-protected. Otherwise, it is too risky.

PART 4 - PRECAUTIONS (Cont)

CPU PRECAUTIONS (Cont)

Usually, when your program runs wild, the CPU ends up at some high memory location where you don't have any memory. The computer hardware generates 0 for nonexistent memory, so you will quickly run through a long string of NOPs until you wrap from 377377A to 000000A, which is the master clear restart address for PAM-8. If you display the PC and find it set to your high memory address, then you probably took this "circumpolar" route into PAM-8.

Debugging Hints

The best way to debug programs is to ORG them above DBUG, and test them using DBUG. After entering DBUG, use the LOAD command to load in the program under test. You can then break-point and single-step through your program. Do not single-step through an HDOS SCALL, or you may damage the disk.

After the program seems to be working, ORG it back down to 042200A, or wherever, and reassemble.

****

PART 5 - RESIDENT SCALLS

INTRODUCTION

Within HDOS 3.0 all system calls are "Resident Scalls," since there are no overlays. This segment covers those HDOS system calls, often referred to as "SCALLS," which are permanently resident in memory.

In general for the H8, a SCALL (System CALL) consists of a

```
        RST    7
```

instruction followed by a byte containing the request number. Most SCALLs require that some registers be set up before the call. Likewise, most may alter the registers, so a program should save any registers which it wants to preserve.

The ASM (assembler) has a special opcode for SCALLs: 'SCALL code' - where "code" is the number of the request. This statement generates the equivalent of:

```
                    DB    377Q,code
```

We recommend that you use the HOSDEF.ACM file to include all these definitions. In general, it is advisable to use the recommended symbol definitions for all references to HDOS, and include them in one or more XTEXT decks. This will make programming easier for you, and guarantee compatibility with future HDOS releases. Although we will make every effort to keep binary compatibility, we may need to revert to "assembly language compatibility," in which case you may have to change some HDOS symbol values and reassemble.

The SCALLs for HDOS Versions 3.0/3.02 are listed in numerical order:

.EXIT - Exit User Program (Octal 0Q)

```
***     EXIT - EXIT USER PROGRAM
*
*       EXIT IS CALLED TO RETURN CONTROL TO THE SYSTEM COMMAND
*       PROGRAM.
*
*       FOR A NORMAL EXIT, THE CONTROL CHARACTERS ARE
*       CLEARED, AND SYSCMD IS ENTERED.
*
*       MVI       A,FLAG          (see below)
*       SCALL     .EXIT
*
*       FOR EITHER EXIT, THE CONTROL CHARACTER VECTORS
*       (SET BY .CTLC) ARE CLEARED.
```

PART 5 - RESIDENT SCALLS

.EXIT - Exit User Program (Octal 0Q) (Cont)

```
*
*       IN ADDITION, THE ABORT EXIT RESETS THE DISK AND
*       CONSOLE I/O DRIVERS.
*
*       ENTRY     (A)       = FLAG ( 0 = NORMAL, 1 = ABORT )
*       EXIT      -IF-      [ SYSTEM DISK IS STILL MOUNTED ]
*                                 - or -
*                           [ STAND-ALONE IS SET ]
*                 -THEN-    EXIT TO "SYSCMD.SYS"
*                 -ELSE-    EXIT TO REBOOT CODE
```

Notes:

The .EXIT SCALL is the proper way for a program to return control to HDOS. In any mode, .EXIT will close all open I/O channels. This action is equivalent to that of the .CLEAR SCALL. It is best for a program to close or clear its own channels before incurring .EXIT, as future releases may differ in this action.

It should not be necessary for a program to use the abort exit unless some process was being used which affected the state of the console or disk I/O ports. The use of such processes is not recommended.

If SY0: has been dismounted, and the STAND-ALONE flag is not set, HDOS exits to reboot. If the STAND-ALONE flag has been set, and no disk is mounted on SY0:, or SYSCMD.SYS is not found on the disk mounted on SY0:, HDOS exits to reboot. Thus, the only way for a program to return to the command level once SY0: has been dismounted and remounted is for the STAND-ALONE flag to have been previously set via the SET command, and for the disk mounted on SY0: to have a copy of SYSCMD.SYS on it.

```
**      EXAMPLES:
ALDONE  MVI    A,0          FLAG NORMAL EXIT
        SCALL  .EXIT
ABTXIT  MVI    A,1          FLAG ABORT EXIT
        SCALL  .EXIT
```

PART 5 - RESIDENT SCALLS

.EXIT - Exit User Program (OCTAL 0Q)(Cont)

NOTE

You should always close your open channels before exiting. Do not rely upon .EXIT to do it for you. Future HDOS releases may not be so accommodating. The one exception is for temporary scratch files, for which you may use .CLEAR. This will cause the file to be dissolved and its space returned to the free pool.

.SCIN - System Console Input (Octal 1Q)

```
***     SCIN - SYSTEM CONSOLE INPUT.
*
*       .SCIN TAKES A SINGLE CHARACTER FROM THE CONSOLE INPUT
*       BUFFER, IF ANY ARE AVAILABLE.
*
*  L1   SCALL   .SCIN
*       JC      L1          CHARACTER NOT READY
*
*       ENTRY   NONE
*       EXIT    'C' SET IF NO CHARACTER
*               'C' CLEAR IF CHARACTER
*               (A) =CHARACTER
*       USES    A,F
```

Notes:

This command is relatively obvious. Detailed examples of .SCIN are shown in the HEATH HDOS Software Reference Manual, Chapter 11, Appendix 11-B, pages 11-70 thru 11-76. A simple application example follows:

```
**      EXAMPLE:

RDCHAR  SCALL   .SCIN       TRY TO READ CHARACTER
        JC      RDCHAR      NONE READY YET
        RET                  EXIT, (A) = CHARACTER
```

NOTE

The .CONSL SCALL may be used to set console mode bits.

PART 5 - RESIDENT SCALLS

.SCOUT - System Console Output (Octal 2Q)

```
***     SCOUT - SYSTEM CONSOLE OUTPUT
*
*       SCOUT OUTPUTS A SINGLE CHARACTER TO THE CONSOLE.  CURSOR
*       POSITIONING IS KEPT TRACK OF.  A 'NL' CHARACTER
*       INDICATES A NEW LINE.  'CR' AND 'LF' CHARACTERS
*       SHOULD NOT BE USED.
*
*       MVI     A,CHAR
*       SCALL   .SCOUT
*
*       ENTRY   (A) = CHARACTER
*       EXIT    (A) = CHARACTER
*       USES    NONE
```

Notes:

This command is relatively obvious. Detailed examples of .SCOUT are shown in the HEATH HDOS Software Reference Manual, Chapter 11, Appendix 11-B, pages 11-71 through 11-76. A simple application example follows:

```
**      EXAMPLE:

        MVI     A,'*'
        SCALL   .SCOUT      TYPE AN ASTERISK ON THE CONSOLE
```

.....

.PRINT - Print Line On System Console (Octal 3Q)

```
***     .PRINT - PRINT CONSOLE LINE.
*
*       PRINT CAUSES A CODED LINE TO BE PRINTED AT THE CONSOLE.
*
*       LXI     H,LINEADDR
*       SCALL   .PRINT
*
*       THE LAST CHARACTER IN THE LINE SHOULD HAVE THE
*       200Q BIT SET.
*
```

PART 5 - RESIDENT SCALLS

.PRINT - Print Line On System Console (Octal 3Q) (Cont)

```
*
*       ENTRY   (HL) = LINE ADDRESS
*       EXIT    (HL) = LWA OF MESSAGE +1
*       USES    A,F,H,L
```

Notes:

.PRINT is an efficient and convenient way to print lines on the system console. Another good way is to use the subroutine "$TYPTX," as shown in Part 8, Appendix 13-A, pages 13-73 through 13-76 of this manual. Note that the parity bit (200Q) is set over the last character to be printed to notify the end-of-line to HDOS. Remember, use the NL character (012Q, same as LF) for a CRLF sequence. HDOS will automatically insert the required number of PAD characters for the console. If you prefer, you can include the "NULL 00" character in a print line. It is ignored, does not cause a delay in console output, and thus cannot be used as a PAD character.

```
***     EXAMPLE:
        .
        LXI     H,MSGA      TYPE OUT STARTUP MESSAGE
        SCALL   .PRINT
        .
        .
PROMPT  LXI     H,MSGB      TYPE OUT PROMPT MESSAGE
        SCALL   .PRINT
REACHA  SCALL   .SCIN       READ REPLY....
        .
        .
MSGA    DB      12Q,'SET OPTIONS:'
        DB      12Q
        DB      12Q, 'HELP - TYPE THIS LIST'
        DB      12Q, 'CRASH - DESTROY DISK SURFACE'
        DB      12Q+200Q                    NEW LINE, END OF PRINT

MSGB    DB      12Q,'YOUR COMMAND?',' '200Q
```

PART 5 - RESIDENT SCALLS

.READ - Read From File (Octal 4Q)

Notes:

Use the .READ SCALL to read data from an open channel. The channel must already have been opened via a .OPENR or .OPENU SCALL (except for channel -1, as noted previously).

Currently, all device I/O under HDOS (with the exception of the console, via the .SCIN and .SCOUT SCALLs) is "block mode." This means that you must read or write to the device in multiples of 256 bytes. If you cannot fill in the last block, you should pad it with zero bytes. The last block in all HDOS source files is padded out to 256 characters with 00 bytes.

The quoted C in the following example indicates the Carry Flag. This SCALL, as in all others in HDOS, returns with the carry flag set if an error or abnormal condition has occurred. The most common "error" for the .READ command is "end-of-file." The convention used above and throughout this manual is that exit conditions which are predicated on the setting of a flag are discussed directly under that flag, indented one space. Thus, the (BC) register pair contains the unused byte count if, and only if, the "C" flag is set. If "C" is clear, then all of the bytes were read, and (BC) contains garbage. Thus, the (BC) and (DE) registers contain meaningful information only when an error condition occurred, which is normally an "end-of-file." The error codes returned by HDOS are defined in Part 7, pages 13-67 and 13-68 of this manual. This is simply a condensation of the error messages discussed in the HDOS Software Reference Manual. (For details concerning HDOS system error messages, refer to Chapter 11, pages 11-61. Note that you can use the .ERROR SCALL to look up an explanatory message.

```
***     .READ - BLOCKS OF DATA
*
*       READ PROCESSES READ SCALLS.  IF A SERIAL DEVICE, PASS TO
*       DRIVER.  IF A STORAGE DEVICE, HANDLE STORAGE MAPPING.
*
*       MVI     A,CHAN
*       LXI     B,COUNT     MUST BE MULTIPLE OF 256
*       LXI     D,ADDR
*       SCALL   .READ       READ DATA FROM FILE
*
*       ENTRY   (A) = I/O CHANNEL/NUMBER
*               (B) = COUNT OF 256-BYTE BLOCKS TO TRANSFER
*               (C) = 0
*               (DE) = DATA ADDRESS
*
```

PART 5 - RESIDENT SCALLS

.READ - Blocks Of Data (Octal 4Q)(Cont)

```
*
*       EXIT    'C' CLEAR IF ALL OK
*               'C' SET IF ERROR
*               (A) = ERROR CODE
*               (BC) = UNUSED TRANSFER COUNT
*               (DE) = NEXT UNUSED ADDRESS
*       USES    ALL
```

NOTE

All read operations must be for integer multiples of 256 bytes. Thus, the last sector in a file may have to be padded with 00 bytes. All ASCII (coded) files in HDOS are zero-byte filled in the last sector (if they need it). A 00 byte is considered to be a NULL character and should always be ignored when encountered in an ASCII file.

```
***     EXAMPLES:

        .
        .
READ    MVI     A,1         READ FROM ALREADY OPEN CHANNEL 1
        LXI     B,256       READ ONE SECTOR
        LXI     D,BUFFER
        SCALL   .READ       READ IT
        JC      READ1       ERROR
        LXI     B,256       READ 256 BYTES
        JMP     READ2

*       HAVE ERROR.  SEE IF EOF, OR SOMETHING WORSE
READ1   CPI     EC,EOF      SEE IF JUST EOF
        JNE     ERROR       HAVE SERIOUS ERROR
        STA     EOFFLG      FLAG HAVE SEEN EOF
        LXI     H,256       (HL) = ORIGINAL STARTING COUNT
        MOV     A,L
        SUB     C
        MOV     C,A
        MOV     A,H
        SBB     B
        MOV     B,A         (BC) = 256-REMCNT = AMOUNT READ

*       READ COMPLETE.  (BC) = BYTES AVAILABLE
READ2   ....
        .
        .
BUFFER  DS      256         SECTOR BUFFER
```

PART 5 - RESIDENT SCALLS

.WRITE - Write to Open File (Octal 5Q)

```
***     .WRITE - PROCESS WRITE SCALL.
*
*       MVI     A,CHAN
*       LXI     B,COUNT     MUST BE MULTIPLE OF 256
*       LXI     D,ADDR
*       SCALL   .WRITE      WRITE DATA TO CHANNEL
*
*       ENTRY   (A) = CHANNEL #
*               (BC) = DATA COUNT
*               (DE) = DATA ADDRESS
*       EXIT    'C' CLEAR IF ALL OK
*               'C' SET IF ERROR
*                (BC) = UNUSED TRANSFER COUNT
*                (DE) = NEXT UNUSED ADDRESS
*                (A) = ERROR CODE
*       USES     ALL
```

Notes:

The .WRITE SCALL is very similar to the .READ SCALL, except that it writes the data to the file. Once again, the count in (BC) must be an integral multiple of 256. The most typical error returned by .WRITE is "NO ROOM ON MEDIA."

NOTE

All write operations must be for integer multiples of 256 bytes. Thus, the last sector in a file may have to be filled out to 256 bytes. All ASCII (coded) files in HDOS are zero-byte filled in the last sector (if they need it). A 00 byte is considered a NULL character and should always be ignored when encountered in an ASCII file.

```
**      EXAMPLE:

WRIDAT  MVI     A,1         CHANNEL 1 ALREADY OPEN
        LXI     B,512       WRITE 512 BYTES
        LXI     D,BUFFER
        SCALL   .WRITE      WRITE IT
        JC      ERROR       SERIOUS ERROR
        .
        .
BUFFER  DS      512         BUFFER AREA FOR WRITE
```

PART 5 - RESIDENT SCALLS

.CONSL - Set Console Mode Bits (Octal 6Q)

```
***     .CONSL - SET AND CLEAR CONSOLE FLAGS.
*
*       CONSL IS CALLED TO SET, CLEAR, OR READ BITS IN THE
*       VARIOUS CONSOLE FLAGS.
*
*       THE CALLER PASSES AN INDEX INTO THE PROPER FLAG.  A
*       MASK TO INDICATE THE AFFECTED BITS, AND A SET OF NEW
*       VALUES FOR THOSE BITS.
*
*       INDEX =
*
*       0       I.CSLMD
*       1       I.CONTY
*       2       I.CUSOR
*       3       I.CONWI
*       4       I.CONFL
*
*       ENTRY   (A) = INDEX
*               (B) = NEW VALUES
*               (C) = MASK ('1' BIT FOR EVERY BIT TO CHANGE)
*       EXIT    'C' CLEAR IF NO ERROR
*                (A) = NEW VALUE
*               'C' SET IF ERROR
*                (A) = ERROR CODE
*       USES    ALL
```

Notes:

The .CONSL SCALL is used to read and write the console control bits and bytes. These bytes are available directly in memory, but we recommend that you access them via the .CONSL command to guarantee synchronization and upward compatibility with future releases of HDOS.

The caller supplies HDOS with three values: the index of the byte to be read and/or written, the bits to be altered, and the new bit values. The technique of supplying a "bits-affected" mask and a "new value" pattern allows you to alter just one bit in a byte without having to know the values of the other bits in the byte. Since the console is an interrupt-responsive device, this also avoids synchronization problems. There are five bytes which may be read and/or written via the .CONSL function.

PART 5 - RESIDENT SCALLS

.CONSL - Set Console Mode Bits (Octal 6Q)(Cont)

I.CSLMD - Console Mode

```
I.CSLMD EQU     0           I.CSLMD IS FIRST BYTE

CSL.ECH EQU     10000000B   SUPPRESS ECHO
CSL.WRP EQU     00000010B   WRAP LINES AT WIDTH
CSL.CHR EQU     00000001B   UPDATE IN CHARACTER MODE
```

These three bits are used to affect the mode in which HDOS handles characters typed at the console. They are documented in greater detail in the "HDOS Software Reference Manual," Chapter 11, pages 11-70 through 11-76.

I.CONTY - Console Type

```
I.CONTY EQU     1           I.CONTY IS 2ND BYTE

CTP.BKS EQU     10000000B   TERMINAL PROCESSES BACKSPACES
CTP.MLI EQU     00100000B   MAP LOWER CASE TO UPPER ON INPUT
CTP.MLO EQU     00010000B   MAP LOWER CASE TO UPPER ON OUTPUT
CTP.2SB EQU     00001000B   TERMINAL NEEDS TWO STOP BITS
CTP.BKM EQU     00000010B   MAP BKSP (UPON INPUT) TO RUBOUT
CTP.TAB EQU     00000001B   TERMINAL SUPPORTS TAB CHARACTERS
```

The bits in the I.CONTY byte are used to describe the console's hardware characteristics. These bits are all discussed under the SET command section in the "HDOS Software Reference Manual." See Chapter 3, page 3-19, for details.

I.CUSOR - Console Cursor Position

```
I.CUSOR EQU     2           I.CUSOR IS 3RD BYTE
```

The I.CUSOR byte contains the current cursor position of the console terminal cursor. Immediately after a NEW LINE, this byte contains 001.

I.CONWI - Console Width

```
I.CONWI EQU     3           I.CONWI IS 4TH BYTE
```

The I.CONWI byte contains the current console width. This value is documented under the SET command in the HDOS Software Reference Manual. See Chapter 3, page 3-19, for details. In brief, when the cursor reaches this value, HDOS automatically generates a NEW LINE. You can effectively disable this option by setting the width to 255.

PART 5 - RESIDENT SCALLS

.CONSL - Set Console Mode Bits (Octal 6Q)(Cont)

I.CONFL - Console Flags

```
I.CONFL EQU     4           I.CONFL IS 5TH BYTE

CO.FLG  EQU     00000001B   CTL-O FLAG
CO.FLG  EQU     10000000B   CTL-S FLAG
```

The I.CONFL byte contains the current setting of the console CTL-O and CTL-S bytes. A user program may find it useful to note that the user has typed CTL-O or CTL-S. In addition, your program may want to clear the CTL-O flag immediately before an input prompt is typed, so that the typing of the prompt is guaranteed.

NOTE

If the CTL-S flag is set, and your program issues a character to the console via .SCOUT or .PRINT, then your program will hang up in HDOS, waiting for the CTL-S flag to clear. There is no way to do a "conditional" character type out. Programmers who do not want their programs to hang up must check the status of the CTL-S flag after every .SCOUT or .PRINT, and trust to luck that the user doesn't type the CTL-S between the .CONSL and the .SCOUT.

** EXAMPLES:

```
*       SET CHARACTER MODE, NO ECHO

        MVI     A,I.CSLMD   (A) = BYTE INDEX
        MVI     B,CSL.ECH+CSL.CHR   SET BOTH BITS
        MVI     C.CSL.ECH+CSL.CHR   AFFECT BOTH BITS
        SCALL   .CONSL

*       SET MAP LOWER CASE TO UPPER, CLEAR BACKSPACE ON 'RUBOUT' KEY

        MVI     A,I.CONTY   (A) = BYTE INDEX
        MVI     B,CTP.MLI+CTP.MLO   SET MAP LOWER CASE BITS
        MVI     C,CTP.MLI+CTP.MLO+CTP.BKS   SET MAP, CLEAR BKS
        SCALL   .CONSL

*       READ CONSOLE CURSOR POSITION
        MVI     A,I.CUSOR
        MVI     C,0         AFFECT NO BITS, (B) MEANINGLESS
        SCALL   .CONSL      AFFECT NOTHING, JUST GET NEW
                            (SAME AS OLD) VALUE
        CPI     1           SEE IF CURSOR OVER COLUMN 1
```

PART 5 - RESIDENT SCALLS

.CONSL - Set Console Mode Bits (Octal 6Q)(Cont)

```
*       SET CONSOLE WIDTH

        MVI     A,I.CONWI
        MVI     B,80        SET 80 COLUMNS
        MVI     C,377Q      AFFECT FULL BYTE
        SCALL   .CONSL      SET WIDTH
```

.CLRCO - Clear Console Buffer (Octal 7Q)

```
***     .CLRCO - CLEARS THE CONSOLE BUFFERS.
*
*       CLRCO CLEARS THE CONSOLE TYPE-AHEAD BUFFER.
*       CTL-O AND CTL-S FLAGS ARE ALSO CLEARED.
*
*       EMTRY   NONE
*       EXIT    NONE
*       USES    ALL
```

Notes:

The .CLRCO SCALL is used to clear the console buffer and the console CTL-O and CTL-S flags. HDOS contains a "type-ahead" buffer, so the user may type commands before a program asks to read them from the console. All typed text is stored in this "type-ahead" buffer; the .SCIN SCALL reads the characters from the buffer. The special control characters, CTL-A, CTL-B, and CTL-C, are not stored in the type-ahead buffer, but instead cause an interrupt to a user service routine (if you set one up via the .CTLC SCALL). Often a user has typed a partial line before he has typed the CTL-C (or CTL-A, or CTL-B). You can use the .CLRCO function to clear out any unwanted type-ahead characters.

NOTE

Issuing the .CLRCO function does not cause a NEW LINE to be sent to the console. The user is given no indication that the characters he may have typed in have been discarded. Your program should issue a new prompt immediately after the .CLRCO function to make things clear to the user.

PART 5 - RESIDENT SCALLS

.CLRCO - Clear Console Buffer (Octal 7Q)(Cont)

```
***     EXAMPLE: CLEANUP AFTER CTL-C

*       ASSUME CONTROL PASSES HERE AT CTL-C

CCHIT   LXI     H,CCHITA    TYPE CTL-C
        SCALL   .PRINT      ACKNOWLEDGE CTL-C, SETUP NEW LINE
        SCALL   .CLRCO      CLEAR TYPE-AHEAD
        .
        .
CCHITA  DB      '^C',212Q   ^C WITH NEW-LINE
```

NOTE

PART 6 discusses intercepting CTL-Cs.

PART 5 - RESIDENT SCALLS (Cont)

.VERS - HDOS Version Number (Octal 11Q)

```
***     VERS - RETURN HDOS VERSION NUMBER
*
*       VERS RETURNS THE HDOS VERSION NUMBER AS A ONE-BYTE
*       BCD NUMBER.  A DECIMAL IS ASSUMED BETWEEN THE HIGH
*       AND LOW ORDER NYBBLES.
*
*       ENTRY   NONE
*       EXIT    (PSW)       = 'C' CLEAR IF NO ERROR
*                               (A) = VERSION NUMBER
*                             'C' SET IF ERROR ( VERS < 1.5 )
*                               (A) = ERROR CODE ( EC.ILC )
*
*       USES    A,F
```

Notes:

The .VERS system call returns the current version number of HDOS. The primary use of this system call is to ascertain under which version of HDOS the program is running. If the program determines that the version does not support these new SCALLs, it may exit gracefully with an error message.

The version number is returned as one BCD byte. That is, version 1.5 will return 21, 25Q, or 015H. Refer to the HDOS Common Deck listing for an example of the definition format.

```
***     EXAMPLE:

        SCALL   .VERS
        JC      BADVER      No version system call
        CPI     VERS
        JNZ     BADVER      Invalid version
        .
        .
BADVER  LXI     B,MESSAG
        SCALL   .PRINT
        .
        .
MESSAG  DB      12Q,'This Version of HDOS Does Not Support'
        DB      'the Required System Calls.',12Q+200Q
```

PART 5 - RESIDENT SCALLS (Cont)

.GDA - Get Device Driver Address (Octal 12Q)

```
***     GDA - Get Device Driver Address.
*
*       Entry:  (DE) = Device Name
*       Exit:   'C' Clear
*                (HL) = Driver address
*                (BC) = Table address for this device
*               'C' Set if error
*                (A) = Error Code
*                (HL) = (BC) = 0
*       Uses:   A,F,H,L,B,C
```

.CRC16 - CRC-16 Is A Block Of Memory (Octal 13Q)

```
***     CRC16 - CRC-16 A Block of memory.
*
*       Entry:  (HL) = BUFFER ADDRESS
*               (DE) = INITIAL CRC-16
*               (BC) = LENGTH OF DATA
*       Exit:   (HL) = ADVANCED PAST BUFFER
*               (DE) = UPDATED CRC-16
*               (BC) = 0
*       Uses:   ALL
```

.LINK - Process Link Scall (Octal 40Q)

```
***     .LINK - PROCESS LINK SCALL
*
*       LINK LOADS IN AND RUNS ANOTHER PROGRAM.  THE OPEN FILES
*       SYSTEM TABLES, AND STACK ARE NOT DISTURBED.
*
*       ENTRY   (HL) = ADDRESS OF PROGRAM FILE DESCRIPTOR
*       EXIT    TO LINKED PROGRAM, IF OK
*                (A) UNCHANGED
*                (SP) = VALUE AT 'LINK' SCALL
*               TO CALLER IF ERROR
*                'C' SET
*                (A) = ERROR CODE
*       USES    ALL
```

PART 5 - RESIDENT SCALLS (Cont)

.LINK - Process Link Scall (Octal 40)(Cont)

Notes:

The .LINK SCALL is used to pass control to another program.

```
**      EXAMPLE: TRANSFER CONTROL TO PROGRAM 'CLEANUP.ABS'

XFER    MVI     A,-1        CHANNEL -1 OPEN ON LOADED FILE

*       GET DEVICE WERE LOADED FROM, SO THAT WE CAN
*       RUN 'CLEANUP.ABS' FROM THAT SAME DISK

        LXI     D,DEVCODE   AREA FOR DEVCODE
        LXI     H,BUFFER    PUT NAME INTO SCRATCH AREA
        SCALL   .NAME

*       BUILD NAME TO LINK TO ...

        LXI     B,XFERAL    (BC) = NUMBER OF BYTES TO MOVE
        LXI     D,XFERA      FROM XFERA
        LXI     H,DEVCODE+3  PUT AFTER DEVICE SPECIFICATION
        CALL    $MOVE        PUT NAME AFTER DEVICE (ROUTINE
                             IN H17 ROM)

*       CALL PROGRAM

        LXI     H,DEVCODE
        SCALL   .LINK       TRY TO EXECUTE IT
        JC      ERROR       FAILED

XFERA   DB      ':CLEANUP.ABS',0     NAME
XFERAL  EQU     *-XFERA     AMOUNT TO MOVE

DEVCODE DS      3+XFERAL    ROOM FOR ENTIRE FILE SPECIFICATION
```

PART 5 - RESIDENT SCALLS (Cont)

.CTLC - Set Up Handlers for Control Characters (Octal 41Q)

```
***     .CTLC - SET CONTROL CHARACTER ADDRESS
*
*       THE .CTLC SCALL IS USED TO SET UP HANDLING FOR
*       THE CONTROL CHARACTERS CTL-A, CTL-B, AND CTL-C.
*
*       A SEPARATE ADDRESS IS SPECIFIABLE FOR EACH CHARACTER.
*       IF AN ADDRESS OF 0 IS SPECIFIED, PROCESSING OF THAT
*       CHARACTER IS SUSPENDED.
*
*       THE PROCESS ADDRESS MUST BE >255A.
*
*       ENTRY   (A) = CONTROL CHARACTER WHOSE PROCESS ADDRESS IS
*                     TO CHANGE (CTL-A, CTL-B, OR CTL-C)
*               (HL) = NEW ADDRESS (=0 TO CLEAR PROCESSING)
*       EXIT    'C' CLEAR IF OK
*               'C' SET IF ERROR
*                (A) = ERROR CODE
*       USES    A,F,H,L
```

Notes:

The .CTLC SCALL allows you to set up interrupt service routines for the handling of CTL-A, CTL-B, and CTL-C. You may set up a separate service routine for each character.

When a service routine has been set up, and the specified character has been struck, your routine will be entered at interrupt-time, with interrupts enabled.

Upon entry to your routine, the registers B, C, D, E, and L have whatever contents were in them at the time of the control character interrupt. The stack contains:

```
((SP)+0) = Return Address into HDOS
((SP)+2) = Interrupted PSW
((SP)+4) = Interrupted PC
```

Your routine can do some interrupt-time work (having saved the registers first, of course) and then do a RET to HDOS, in which case HDOS will take care of the rest. Or, if you wish, you may ignore the HDOS return address and jump back into your program's command loop or whatever.

PART 5 - RESIDENT SCALLS (Cont)

.CTLC - Set Up Handlers For Control Characters (Octal 41Q)(Cont)

```
**      EXAMPLE 1: SETTING AN 'INTERRUPT OCCURRED' FLAG

        LXI     H,CCINT     SET UP CTL-C INTERRUPT PROCESSOR
        MVI     A,003       (A) = CTLC
        SCALL   .CTLC       SET UP CTL-C
        .
        .
LOOP    SCALL   .SCIN
        JNC     GOTONE      GOT A CHARACTER
        LDA     CCHIT
        ANA     A
        JZ      LOOP        NO CTL-C HIT
        JMP     PROCC       PROCESS CTL-C
        .
        .
*       CTL-C CAUSES THIS ROUTINE TO BE ACTIVATED

CCINT   MVI     A,1         PSW IS ALREADY SAVED
        STA     CCHIT       SET CC HIT
        RET                 RETURN TO INTERRUPTED CODE VIA
HDOS
        .
        .
CCHIT   DB      0           SET =1 WHEN CTL-C TYPED


**      EXAMPLE 2: RETURNING CONTROL TO MAIN COMMAND LOOP.

        .

        .
        .
        LXI     H,CBHIT
        MVI     A,002       (A) = CTLB
        SCALL   .CTLB
        .
        .
START   LXI     SP,STACK    CLEANUP STACK
LOOP    .                   DO WHATEVER WE DO...
        .
        .
*       ENTERED HERE IF CTL-B HIT

CBHIT   JMP     START       RESTART COMMAND LOOP
```

PART 5 - RESIDENT SCALLS (Cont)

.OPENR - Open File for Read (Octal 42Q)

```
***     .OPENR - OPENR SCALL PROCESSOR.
*
*       OPENR IS CALLED TO OPEN A CHANNEL FOR READ.
*
*       THE CALLER SUPPLIES A FILE NAME.  A DEFAULT BLOCK
*       FOR THE DEVICE AND EXTENSION, AND A CHANNEL NUMBER.
*
*       DEFAULT BLOCK FORMAT:
*
*       DB      'DDD'       DEFAULT DEVICE
*       DB      'XXX'       DEFAULT EXTENSION
*
*       ENTRY   (DE) = DEFAULT BLOCK ADDRESS
*               (HL) = NAME ADDRESS
*               (A) = CHANNEL NUMBER
*       EXIT    'C' CLEAR IF OK
*               (HL) = ADVANCED PAST FILE NAME
*               'C' SET IF ERROR
*                (A) = ERROR CODE
*       USES    ALL
```

Notes:

Use the .OPENR SCALL to open files for read access. This means that you may then read the file, but HDOS will not allow any write requests to it. You may open an individual file for read access on as many channels as you wish.

The channel number supplied must be a legal one (i.e., -1 through 5), and must not already have a file open on it.

HDOS will not allow any one file to be open for both read and write at the same time, nor may any one file be open for write to more than one channel. Attempting to do this will cause a "usage conflict" error. This means that you may not open a file via .OPENR if it is already open for write, or update, on another channel.

**   EXAMPLE:

```
*       OPEN PRE-DETERMINED FILE NAME ON CHANNEL 1

        MVI     A,1         CHANNEL 1
        LXI     D,DEFALT    POINT TO DEFAULT BLOCK
        LXI     H,FNAME     POINT TO FILE DESCRIPTOR
        SCALL   .OPENR      OPEN FOR READ
        JC      ERROR       SOME ERROR
```

PART 5 - RESIDENT SCALLS (Cont)

.OPENR - Open File For Read (Octal 42Q)(Cont)

```
        .
        .

*       READ FILE NAME FROM USER, OPEN ON CHANNEL 2

        LXI     H,MSGA
        SCALL   .PRINT      PROMPT HIM
        LXI     H,BUFFER
REA1    SCALL   .SCIN
        JC      REA1        NO CHARACTER
        MOV     M,A         STORE IN MEMORY
        INX     H
        CPI     012Q        SEE IF NEW LINE (USER HIT CR KEY)
        JNE     REA1        NOT YET
        DCX     H
        MVI     M,0         TERMINATE LINE WITH 00, INSTEAD
                            OF 012Q

        LXI     H,BUFFER
        LXI     D,DEFALT    POINT TO DEFAULT BLOCK
        MVI     A,2         CHANNEL 2
        SCALL   .OPENR      OPEN FILE
        JC      ERROR
        .
        .

MSGA    DB      12Q,'FILE NAME?',''+200Q
DEFALT  DB      'SY0TMP'    DEFAULT DEVICE AND EXTENSION
BUFFER  DS      20          FILE NAME BUFFER
FNAME   DS      'SY1:MYFILE.NEW',0   FILE NAME FOR CHANNEL 1
```

.OPENW - Open File for Write (Octal 43Q)

```
***     .OPENW - OPEN FILE FOR WRITE
*
*       OPENW IS CALLED TO OPEN A CHANNEL FOR WRITE.
*
*       THE FILE IS ENTERED IN THE CHANNEL TABLE, BUT NOT ON THE
*       DISK.  IT WILL BE ENTERED IN THE DIRECTORY AT CLOSE TIME.
*
*       THE CALLER SUPPLIES A FILE NAME, A DEFAULT BLOCK FOR THE
*       DEVICE AND EXTENSION, AND A CHANNEL NUMBER.
*
*       DEFAULT BLOCK FORMAT:
*
```

PART 5 - RESIDENT SCALLS (Cont)

.OPENW - Open File For Write (Octal 43Q)(Cont)

```
*
*       DB      'DDD'       DEFAULT DEVICE
*       DB      'XXX'       DEFAULT EXTENSION
*
*       ENTRY   (DE) = DEFAULT BLOCK ADDRESS
*               (HL) = NAME ADDRESS

*               (A) = CHANNEL NUMBER
*       EXIT    'C' CLEAR IF OK
*               (HL) = ADVANCED PAST FILE NAME
*               'C' SET IF ERROR
*                (A) = ERROR CODE
*       USES    ALL
```

Notes:

Use the .OPENW SCALL to open a file for writing. When HDOS processes the .OPENW SCALL, the file is opened with a "temporary" name, which does not appear in the disk directory. When the channel is closed, HDOS will then enter the name in the directory. If any previous file by that name existed, it will be automatically deleted at that time. This procedure has three implications:

1. You cannot modify an existing file by means of the .OPENW SCALL! .OPENW is intended for creating new files or replacing old ones.

2. If you are replacing an existing file, there must be enough free space to hold both the new version and the old one, as the old one will not be deleted until the new one is closed. You might want to manually delete the old file first via .DELETE.

3. If you do not properly close the channel, the new file will be lost. This is intended as a safety factor. A previously existing file by that name will not be destroyed until the new one has been successfully completed. If you should start to write a file by some name, then realize that you already have a useful file by that name, you can CTRL-Z out and still retain the old file.

HDOS will not allow any one file to be open for both read and write at the same time, nor may any one file be open for write on more than one channel. If you attempt to do this, you will cause a "usage conflict" error. This means that you cannot open a file with .OPENW if it is already open for write or update, or if it is open for read.

PART 5 - RESIDENT SCALLS (Cont)

.OPENW - Open File For Write (Octal 43Q)(Cont)

The examples shown above for .READ are applicable to .WRITE as well. The following example illustrates opening a file on a non-disk device, "AT:." Note that exactly the same procedure is followed. In fact, in the above example where the user types in a file name, he may just as well have typed in "TT:" or "AT:" for a device specification.

```
**      EXAMPLE:

        .

        .
        MVI     A,3         OPEN ON CHANNEL 3
        LXI     D,DEFALT    POINT TO DEFAULT BLOCK
        LXI     H,FNAME
        SCALL   .OPENW
        JC      ERROR       ERROR
        .
        .
        .
DEFALT  DB      'SY0',0,0,0 UNUSED, BUT REQUIRED
FNAME   DB      'AT:',0     NAME AND EXTENSION MEANINGLESS
```

.OPENU - Open File For Update (Octal 44Q)

```
***     .OPENU - OPEN FILE FOR UPDATE
*
*       OPENU IS CALLED TO OPEN A CHANNEL FOR UPDATE.
*
*       UPDATE IS JUST LIKE READ, BUT THE FILE MAY BE WRITTEN
*       ALSO.
*
*       THE CALLER SUPPLIES A FILE NAME.  A DEFAULT BLOCK FOR THE
*       DEVICE AND EXTENSION, AND A CHANNEL NUMBER.
*
*       DEFAULT BLOCK FORMAT:
*
*       DB      'DDD'       DEFAULT DEVICE
*       DB      'XXX'       DEFAULT EXTENSION
*
*       ENTRY   (DE) = DEFAULT BLOCK ADDRESS
*               (HL) = NAME ADDRESS
*               (A) = CHANNEL NUMBER
*
```

PART 5 - RESIDENT SCALLS (Cont)

.OPENU - Open File For Update (Octal 44Q)(Cont)

```
*
*       EXIT    'C' CLEAR IF OK
*               (HL) = ADVANCED PAST FILE NAME
*               'C' SET IF ERROR
*                (A) = ERROR CODE
*       USES    ALL
```

Notes:

Use .OPENU to open a file for update. This means that a previously existing disk file is opened for both read and write. When opened, the file is positioned at sector zero.

If the channel is positioned over an existing sector and you issue a .WRITE, then that sector will be rewritten. If the channel is positioned at the end of the file, the file will be extended. You can use the .POSIT SCALL to position the channel at the end of the file. Thus, the .OPENU and .POSIT SCALL combination allows you to append information onto an existing file.

NOTE

Always close a file that you opened for update. Failure to do so causes undefined results. Failing to close the channel properly can also cause "orphaned" sectors, which are not being used by a file, nor are they in the free list. HDOS will automatically recover these orphans when the disk is next mounted or booted and return them to the free list.

The examples used for .OPENR on page 13-31 also apply to .OPENU. Of course, there are some differences.

1. The file opened must already exist.

2. The file must reside on a mass storage device which can be both read and written (i.e., not write-protected).

PART 5 - RESIDENT SCALLS (Cont)

.OPENC - Open Contiguous File for Write (Octal 45Q)

```
***     .OPENC - OPEN CONTIGUOUS FILE FOR WRITE
*
*       OPENC IS CALLED TO OPEN A CHANNEL FOR WRITE.
*
*       THE FILE IS ENTERED IN THE CHANNEL TABLE, BUT NOT ON THE
*       DISK.  IT WILL BE ENTERED IN THE DIRECTORY AT CLOSE TIME.
*
*       THE CALLER SUPPLIES A FILE NAME, A DEFAULT BLOCK FOR THE
*       DEVICE AND EXTENSION, AND A CHANNEL NUMBER.
*
*       DEFAULT BLOCK FORMAT:
*
*       DB      'DDD'           DEFAULT DEVICE
*       DB      'XXX'           DEFAULT EXTENSION
*
*       ENTRY   BC   = SECTOR COUNT
*               (DE) = DEFAULT BLOCK ADDRESS
*               (HL) = NAME ADDRESS
*               A    = CHANNEL NUMBER
*       EXIT    'C' CLEAR IF OK
*               (HL) = ADVANCED PAST FILE NAME
*               'C' SET IF ERROR
*               (A)  = ERROR CODE
*       USES    ALL
```

Notes:

The .OPENC SCALL is used in a manner similar to the .OPENW SCALL, in that it is used to open a file for output. The difference is that HDOS will insure that the file is written with all disk sectors in contiguous order. If there are not enough contiguous sectors available on the diskette (due to fragmentation from prior creation and deletion of files), the .OPENC SCALL will fail. Naturally, HDOS will need to know in advance exactly how big the file will be, so that it can determine if a large enough contiguous area exists on the diskette. The file size (in sectors) is passed to HDOS in register pair BC for the .OPENC SCALL, whereas the .OPENW SCALL does not care what value is present in register pair BC when it is called. This is one of the very useful undocumented SCALLs. HDOS primarily uses it for the creation of the operating system files during SYSGEN, but you can use it anytime you want to insure that the files you create occupy contiguous sectors, as long as you accept the restrictions that this imposes -- (1) you need to know the file size in advance, and (2) you may fail to open the file, even if there is room on the diskette, if there is not enough CONTIGUOUS space.

.....

PART 5 - RESIDENT SCALLS (Cont)

.CLOSE - Close Channel (Octal 46Q)

```
***     .CLOSE - PROCESS CLOSE SCALL.
*
*       CLOSE PROCESSING DEPENDS UPON THE FILE AND DEVICE TYPE.
*
*       FOR A WRITE/DIRECTORY TYPE, THE DIRECTORY IS SEARCHED
*       FOR A PREVIOUS ENTRY.  IF FOUND, IT IS DELETED.  THE NEW
*       ENTRY IS THEN INSERTED.
*
*       FOR A UPDATE/DIRECTORY TYPE, THE PREVIOUS ENTRY IS UPDATED.
*
*       FOR ALL FILES, THE DRIVER IS CALLED WITH THE DC.CLO
*       FUNCTION.  THE CHANNEL IS RELEASED.
*
*       ENTRY   (A) = CHANNEL #
*       EXIT    'C' CLEAR IF OK
*               'C' SET IF ERROR
*                (A) = CODE
*       USES    ALL
```

Notes:

Use the .CLOSE SCALL to close a channel when you are done with it. Always close all channels your program has opened, with two exceptions:

1. HDOS enters your program with channel -1 open on your program load file. If you do not use this channel, you need not close it -- HDOS will perform the close on it automatically.

2. Scratch files which were created via .OPENW, which are no longer needed, need not be closed. See ".CLEAR," on page 13-49.

**   EXAMPLE:

```
        .
        .
        MVI     A,1
        SCALL   .CLOSE      CLOSE CHANNEL 1
        JC      ERROR
        MVI     A,2
        SCALL   .CLOSE      CLOSE CHANNEL 2
        JC      ERROR       IF ERROR
```

PART 5 - RESIDENT SCALLS (Cont)

.POSIT - Position Disk File (Octal 47Q)

```
***     .POSIT - POSITION FILE.
*
*       LXI     B,POSITION
*       MVI     A,CHANNEL NUMBER
*       SCALL   .POSIT
*
*       ENTRY   (A) = CHANNEL NUMBER
*               (BC) = SECTOR NUMBER TO POSITION BEFORE
*       EXIT    'C' CLEAR IF OK
*               'C' SET IF ERROR
*                (A) = ERROR CODE
*                (A) = EC.EOF IF OFF END
*                 (BC) = SECTORS UNSKIPPED (REMAINDER OF COUNT)
*                 FILE POSITIONED AT EOF
*       USES    ALL
```

Notes:

Use the .POSIT SCALL to position the "channel cursor." Since each read or write on a file, via a channel, must transfer in sector (or multi-sector) lots, the channel's current position in the file is simply the logical sector number next to be read or written. This sector number has no relation to actual physical sector numbers. The first sector in a file is sector zero, the next is sector 1, the last sector in an n sector file is n-1.

> NOTE
>
> The .POSIT SCALL positions the channel (file) before the specified sector. Thus, a .POSIT to 0 positions the channel before sector 0, so that a one-sector read will return sector 0. To position the channel at the end of a file, .POSIT to n, where n is the number of sectors in the file. If you do not know how long the file is, .POSIT to 65535 (377377A), verify that an EC.EOF error was flagged, and then compute the file size as SIZE = 65535-(BC).

Thus, when a file is first opened via .OPENR, .OPENW, or .OPENU, it is positioned at sector 0. The first read or write of m sectors will read or write sectors 0 through m-1. This is a normal sequential access. For example, when reading, each one-sector read will return the next sector in the file.

PART 5 - RESIDENT SCALLS (Cont)

.POSIT - Position Disk File (Octal 47Q)

You can use the .POSIT SCALL to set this "sector cursor" at any spot in the file. Positioning a file at sector 0 is the equivalent of rewinding it. A file may be positioned at its end, so a read will return end-of-file, and a write will extend the file. It may not be positioned after the last sector +1 in an attempt to extend the file. Files may be extended only via .WRITE SCALLs.

The .POSIT SCALL strengthens the similarity between .OPENW and .OPENU. If you have opened a file via .OPENW, you may use .POSIT to position the channel cursor to allow you to rewrite any sector in the file, at any time. If you then wish to add some more sectors to the end, you can position to the end of the file, and .WRITE some more. Also note that you can change the value of any byte or bytes in a file open for write or update by positioning before the proper sector, reading the sector, modifying it, repositioning it over again, and writing the sector back.

```
**      EXAMPLE 1: REWINDING A FILE AFTER READING IT

        <OPEN AND READ A FILE ON CHANNEL 1>

        .
        .
        .
        LXI     B,0         BEFORE SECTOR 0
        MVI     A,1         CHANNEL 1
        SCALL   .POSIT      POSITION
        JC      ERROR

        <READ THE FILE OVER AGAIN>

**      EXAMPLE 2: REPLACING A SECTOR IN A FILE BEING WRITTEN

        <OPEN THE FILE VIA .OPENW >
        MVI     A,2         CHANNEL 2
        LXI     B,256*10    WRITE 10 SECTORS
        LXI     D,BUFFER    FROM BUFFER
        SCALL   .WRITE
        JC      ERROR
        LXI     B,1         PREPARE TO RE-WRITE 2ND SECTOR IN FILE
```

PART 5 - RESIDENT SCALLS (Cont)

.POSIT - Position Disk File (Octal 47Q)(Cont)

```
        MVI     A,2
        SCALL   .POSIT
        JC      ERROR
        MVI     A,2         CHANNEL 2
        LXI     B,256
        LXI     D,BUFFER2
        SCALL   .WRITE      WRITE DIFFERENT DATA
        MVI     A,2         CHANNEL 2
        LXI     B,-1        POSITION AT END OF FILE
        SCALL   .POSIT      WILL RETURN EOF ERROR
        CPI     EC.EOF
        JNE     ERROR       OTHER ERROR

        < FURTHER WRITES WILL APPEND TO END OF FILE >

**      EXAMPLE 3: INCREMENTING BYTE 7423 IN FILE "DATA.RAW"

        MVI     A,0         OPEN ON CHANNEL 0
        LXI     D,DEFALT
        LXI     H,FNAME
        SCALL   .OPENU      OPEN FOR UPDATE
        JC      ERROR

*       POSITION FOR READ

        LXI     H,7423      (H) = SECTOR NUMBER
                            (L) = BYTE INDEX
        MOV     C,H
        MVI     B,0         (BC) = SECTOR NUMBER
        PUSH    H           SAVE (HL)
        MVI     A,0
        SCALL   .POSIT      POSITION
        JC      ERROR


*       READ SECTOR INTO WORK AREA

        MVI     A,0         (A) = CHANNEL
        LXI     B,256
        LXI     D,BUFFER
        SCALL   .READ
        JC      ERROR
```

PART 5 - RESIDENT SCALLS (Cont)

.POSIT - Position Disk File (Octal 47Q)(Cont)

```
*       INCREMENT BYTE

        POP     B           (B) = SECTOR, (C) = BYTE INDEX
        LXI     H,BUFFER
        MOV     A,C         (A) = BYTE INDEX
        CALL    $DADA       ADD (A) INTO (HL) (ROUTINE IN
                            H17 ROM)
        INR     M           INCREMENT BYTE IN BUFFER

*       POSITION FOR RE-WRITE

        MOV     C,B
        MVI     B,0         (BC) = SECTOR NUMBER
        MVI     A,0
        SCALL   .POSIT
        JC      ERROR

*       WRITE BACK OUT

        MVI     A,0         (A) = CHANNEL
        LXI     B,256
        LXI     D,BUFFER
        SCALL   .WRITE
        JC      ERROR

*       CLOSE FILE

        MVI     A,0
        SCALL   .CLOSE
        JC      ERROR
        .
        .
        .
DEFALT  DB      'SY0',0,0,0
FNAME   DB      'SY0:DATA.RAW',0
BUFFER  DS      256
```

PART 5 - RESIDENT SCALLS (Cont)

.DELETE - Delete Disk File (Octal 50Q)

```
**      DELETE - PROCESS DELETE COMMAND.
*
*       ENTRY   (HL) = NAME STRING
*               (DE) = DEFAULT BLOCK
*       EXIT    'C' CLEAR IF OK
*               'C' SET IF ERROR
*                (A) = CODE
*       USES    ALL
```

Notes:

Use the .DELETE SCALL to delete a disk file. The format of the call is similar to that of .OPENW, except that no channel number is specified. Note that deleting a file is considered to be a form of writing, so the file must not be open on any channel for reading or writing, as that would cause a "file usage conflict."

```
**      EXAMPLE:

        DELETE FILE "SY0:TEMP.TMP"

DELTEMP LXI     H,NAME
        LXI     D,DEFALT
        SCALL   .DELETE
        JC      ERROR
        .
        .
NAME    DB      'TEMP.TMP',0     FILE NAME
DEFALT  DB      'SY0XXX'         DEFAULT DEVICE, DEFAULT EXTENSION
```

.RENAME - Rename A File (Octal 51Q)

```
***     .RENAME - PROCESS RENAME FUNCTION.
*
*       RENAME RENAMES A FILE ON A DIRECTORY DEVICE.
*
*       * NOTE * RENAME DOES NOT CHECK TO SEE IF THE NEW NAME
*       ALREADY EXISTS--THIS IS THE RESPONSIBILITY OF THE CALLER!
*
*       ENTRY   (HL) = NAME STRING
*               (DE) = DEFAULT BLOCK
*               (BE) = NEW NAME STRING
*
```

PART 5 - RESIDENT SCALLS (Cont)

.RENAME - Rename A File (Octal 51Q)(Cont)

```
*
*       EXIT    'C' CLEAR IF OK
*               'C' SET IF ERROR
*                (A) = CODE
*       USES    ALL
```

Notes:

Use the .RENAME SCALL to change the name of a file on disk. A renaming is considered a form of writing on a file, so the same "usage conflict" restrictions apply: the file to be renamed must not be open on another channel. Two other restrictions exist:

1. A file with the "new name" must not already exist on that device. .RENAME, unfortunately, does not check this for you, so you must check yourself by trying to .OPENR the file before doing the .RENAME. Also, .RENAME will allow you to designate two files on a disk with the same name. The results of this will be disasterous.

2. The "name string" and the "new name string" must both specify the same device (SY0:, SY1:, SY2:, DK0:, DK1:, or DK2:). Alternatively, both files may use the default device, which may be any valid HDOS disk drive name.

> NOTE
> The default block device and extension apply only to the old file name, not to the new name. The new file name must be fully specified, including device, file name, and extension, if there is to be one.

```
**      EXAMPLE:

*       RENAME 'SY1:SORT.ASM' TO 'SY1:SORT.BAK'

        LXI     B,NEWNAM
        LXI     D,DEFALT
        LXI     H,OLDNAM
        SCALL   .RENAME
        JC      ERROR
        .
        .
NEWNAM  DB      'SY1:SORT.BAK',0     NO DEFAULTS ALLOWED
OLDNAM  DB      'SORT',0             USE DEFAULT DEVICE AND EXTENSION
DEFALT  DB      'SY1ASM',0           DEFAULT DEVICE AND EXTENSION
```

PART 5 - RESIDENT SCALLS (Cont)

.SETTOP - Set Top of User Memory (Octal 52Q)

```
***     .SETTOP - SET TOP OF USER MEMORY
*
*       SETTOP IS CALLED TO NOTIFY THE SYSTEM OF A NEW
*       MEMORY LIMIT ADDRESS.  IF NECESSARY, THE OVERLAYS
*       WILL BE UNLOADED.
*
*       ENTRY   (HL) = NEW ADDRESS
*       EXIT    (PSW) = 'C' CLEAR IF OK
*                       'C' SET IF TOO HIGH
*                        (A) = ERROR CODE
*                        (HL) = MAXIMUM ADDRESS
*       USES    ALL
```

Notes:

Use the .SETTOP SCALL to set the top of the user memory area. Since HDOS sets the top of memory to the last address in your program, most programs do not need to use .SETTOP. Programs which need large buffer areas should not declare them with DS statements, since the generated binary file will be excessively large. Instead, they should define the areas via EQU statements, and use the .SETTOP SCALL to request the needed space from HDOS.

Note that by requesting the impossible (65535 bytes), you can determine the actual maximum memory available from the error return. The following information regarding overlays pertains to HDOS 2.0 and below. [If you want to request maximum memory, but avoid swapping the overlays, the approved method is to first load both overlays (see .LOADO for details) and then make the memory request.]

```
**      EXAMPLE 1: GETTING MAXIMUM MEMORY WITHOUT SWAPPING

        MVI     A,OVL0      LOAD OVERLAY 0
        SCALL   .LOADO
        JC      ERROR
*
        MVI     A,OVL1      LOAD OVERLAY 1
        SCALL   .LOADO
        JC      ERROR
*
        LXI     H, -1       CAUSE DELIBERATE ERROR
        SCALL   .SETTOP     .. TO GET MAX IN (HL)
        LXI     D,-10       SUBTRACT 'SLOP' FACTOR
        DAD     D
        SHLD    MAXMEM      SAVE MAX MEMORY
        SCALL   .SETTOP     NOW ASK FOR THE MAX ALLOWABLE
        JC      ERROR       SHOULD NOT HAPPEN
```

PART 5 - RESIDENT SCALLS (Cont)

.SETTOP - Set Top Of User Memory (Octal 52Q)(Cont)

```
        .
        .
MAXMEM  DS      2           MEMORY LIMIT


**      EXAMPLE 2: GETTING ABSOLUTE MAXIMUM MEMORY
*                  ( ENTER HERE WITHOUT LOADING OVERLAYS )

        LXI     H,-1        IMPOSSIBLE AMOUNT
        SCALL   .SETTOP     WILL FAIL ...
        SHLD    MAXMEM      SAVE RESULT
        SCALL   .SETTOP     ASK FOR MAX
        JC      ERROR       SHOULD NOT HAPPEN
        .
        .
MAXMEM  DS      2           MEMORY LIMIT

*       REMEMBER IF THE .SETTOP IS SUCCESSFUL,
*       THE CONTENTS OF (HL) ARE MEANINGLESS.
```

.DECODE - Decode File Name (Octal 53Q)

```
***     .DECODE - PROCESS DECODE SCALL.
*
*       DECODE DECODES THE SUPPLIED FILE NAME
*       INTO A BLOCK IN THE FORM:
*
*       DS      1           RESERVED
*       DS      2           DEVICE NAME
*       DS      1           DEVICE UNIT
*       DS      8           FILE NAME
*       DS      3           FILE EXTENSION
*       DS      4           RESERVED
*
*       ENTRY   (BC) = AREA FOR TABLE TO BE WRITTEN
*               (DE) = DEFAULT LIST
*               (HL) = NAME ADDRESS
*       EXIT    'C' CLEAR IF OK
*               'C' SET IF ERROR
*                (A) = ERROR CODE
*       USES    ALL
```

PART 5 - RESIDENT SCALLS (Cont)

.DECODE - Decode File Name (Octal 53Q)(Cont)

Notes:

Use the .DECODE SCALL to decode an ASCII file descriptor into a formatted block. The fields in the block contain the device, unit, name, and extension values from the file descriptor. The fields are 0 filled. This function is useful for programs which wish to in some way examine the file name, extension, or device specification without going to the work of manually cracking the file descriptor. For example, if your program reads a file descriptor from the console, then wants to know if the extension is "ABC," it might use the .DECODE SCALL to crack out the extension field.

```
**      EXAMPLE: SEE IF USER TYPED DEVICE CODE 'TT:'

        < READ LINE FROM CONSOLE INTO *LINE* >

        LXI     B,BUFFER
        LXI     D,DEFALT
        LXI     H,LINE
        SCALL   .DECODE     DECODE SUPPLIED FILE NAME
        JC      ERROR       ILLEGAL NAME
        LXI     B,3         COMPARE 3 BYTES
        LXI     D,BUFFER+1  (DE) = SUPPLIED DEVICE NAME
        LXI     H,TTSTR
        CALL    #COMP       COMPARE STRINGS (ROUTINE
                            IN H17 ROM)
        JNE     NOTTT       FILE NOT ON TT:
        JMP     GOTTT       NAME DID SPECIFY TT:

BUFFER  DS      19          ROOM FOR REPLY DATA
LINE    DS      80          USER SUPPLIED FILE NAME
TTSTR   DB      'TT',0      NAME AND UNIT IF DEVICE WAS 'TT:'
```

PART 5 - RESIDENT SCALLS (Cont)

.NAME - Get File Name from Channel (Octal 54Q)

```
***     .NAME - PROCESS NAME SCALL.
*
*        THE NAME SCALL RETURNS THE DEVICE, FILE NAME, AND
*        FILE EXTENSION OF AN OPEN CHANNEL.
*
*        THE INFORMATION IS OBTAINED FROM THE CHANNEL TABLE.
*        WHICH WAS SET UP UPON FILE OPEN.
*
*        ENTRY  (A) = CHANNEL NUMBER
*               (DE) = ADDRESS FOR DEVICE AND EXTENSION
*                      (DEFAULT BLOCK FORMAT)
*               (HL) = ADDRESS FOR NAME (8 CHARACTERS
*                      FOLLOWED BY 00 BYTE)
*       EXIT    'C' CLEAR IF OK
*               'C' SET IF ERROR
*                (A) = ERROR CODE
*       USES    ALL
```

Notes:

Use the .NAME SCALL to recall the name which was supplied to HDOS when the channel was opened. This is mainly used when an error message is prepared after HDOS has flagged an error on a channel operation.

```
**      EXAMPLE: ERROR PRINTING PROGRAM.
*
*       THIS ROUTINE PRINTS AN ERROR MESSAGE FOR A
*       FILE OPERATION GONE WRONG.
*
*       ENTRY   (A) = ERROR NUMBER
*               (CURCHAN) = CHANNEL NUMBER USED IN
*                           FAILED OPERATION
*       EXIT    ...

ERROR   PUSH    PSW         SAVE ERROR CODE
        LXI     H,ERRORA
        SCALL   .PRINT      PRINT 'ERROR - '
        POP     PSW         (A) = CODE
        MVI     H,07Q       BELL AFTER ERROR CODE
        SCALL   .ERROR      PRINT ERROR
        LXI     H,ERRORB
        SCALL   .PRINT      PRINT ' ON FILE '
        LDA     CURCHAN     (A) = CHANNEL NUMBER
        LXI     D,ERRDFB    (DE) = ADDRESS FOR DEVICE AND EXTENSION

        LXI     H,ERRNAM    (HL) = ADDRESS FOR NAME
        SCALL   .NAME       GET FILE NAME
```

PART 5 - RESIDENT SCALLS (Cont)

.NAME - Get File Name From Channel (Octal 54Q)(Cont)

```
*       MANIPULATE DEVICE, NAME, AND EXTENSION INTO
*       PRESENTABLE FORMAT, AND PRINT ON CONSOLE

        .
        .
        .
ERRORA  DB      D12Q,'ERROR -',' '+200Q
ERRORB  DB      ' ON FILE ',' '200Q
ERRDFB  DS      6               DEVICE AND EXTENSION FOR BAD FILE
ERRNAM  DS      9               NAME FOR BAD FILE
```

PART 5 - RESIDENT SCALLS (Cont)

.CLEAR - Clear I/O Channel (Octal 55Q)

```
***     CLEAR - CLEAR I/O CHANNEL.
*
*       CLEAR IS CALLED TO CLEAR AN I/O CHANNEL.  IF THE
*       CHANNEL IS CLOSED, NO ACTION IS PERFORMED.  IF THE
*       CHANNEL IS OPEN, IT IS FLAGGED CLOSED.  THE RESULTS
*       OF THIS OPERATION DEPEND UPON THE TYPE OF FILE:
*
*       OPEN FOR                ACTION
*
*       READ                SAME AS .CLOSE
*
*       WRITE               FILE IS FORGOTTEN.  ANY WRITTEN
*                           DISK BLOCKS ARE RESTORED TO THE
*                           FREE POOL.
*
*       UPDATE              REPLACED SECTORS REMAIN REPLACED.
*                           APPENDED SECTORS ARE LOST UNTIL
*                           NEXT BOOT.  FILE STARY AT PREVIOUS
*                           LENGTH.
*
*       THE DEVICE DRIVER IS NOT INFORMED OF THE CLOSING.
*
*       SCALL   .CLEAR
*
*       ENTRY   (A) = CHANNEL NUMBER
*       EXIT    'C' CLEAR IS OK
*               'C' SET IF ERROR
*                (A) = ERROR CODE
*       USES    ALL
```

Notes:

Use the .CLEAR SCALL to free up a channel without closing it. The actions discussed above merely document the current results of the .CLEAR SCALL; they may not stay the same for future releases of HDOS. There is only one supported use of the .CLEAR SCALL, which is to delete temp files. A temp work file is created by means of an .OPENW SCALL. You need not worry about name conflicts, as any preexisting file with the same file name will not be disturbed by the .OPENW. However, when you are done, you do not want to .CLOSE then .DELETE the file, since this would destroy any preexisting file by that name. In that case use .CLEAR on the channel to free up the channel and release the used disk sectors.

PART 5 - RESIDENT SCALLS (Cont)

.CLEAR - Clear I/O Channel (Octal 55Q)(Cont)

```
**      EXAMPLE: CREATING, USING, AND DESTROYING A SCRATCH FILE

        MVI     A,0         USE CHANNEL 0
        LXI     D,DEFALT
        LXI     H,SCRNAME
        SCALL   .OPENW      OPEN TO SCRATCH FILE
        JC      ERROR
        .
        .
        < WRITE DATA ON SCRATCH FILE >
        .
        .
        MVI     A,0
        LXI     B,0
        SCALL   .POSIT      REWIND SCRATCH FILE
        .
        .
        < READ DATA FROM SCRATCH FILE >
        .
        .
        MVI     A,0
        SCALL   .CLEAR      DESTROY SCRATCH FILE
        .
        .
SCRNAME DB      'SY0:TEMP.TMP',0     ANY PRE-EXISTING TEMP.TMP
                                     NOT AFFECTED
```

.CLEARA - Clear All Channels Except -1 (Octal 56Q)

```
***     .CLEARA - CLEAR ALL CHANNELS.
*
*       CLEARA PERFORMS THE .CLEAR ACTION FOR ALL EXISTING CHANNELS,
*       EXCEPT CHANNEL 377Q, THE LOAD IMAGE CHANNEL.
*
*       ENTRY   NONE
*       EXIT    NONE
*       USES    ALL
```

PART 5 - RESIDENT SCALLS (Cont)

.CLEARA - Clear All Channels Except -1 (Octal 56Q)(Cont)

Notes:

The .CLEARA SCALL is equivalent to calling the .CLEAR SCALL once for every valid HDOS channel except for channel 377Q. If you had multiple temporary files open simultaneously and wanted to scratch all of them at once, this SCALL could be used as a convenience. Refer to the section on .CLEAR for details on the actions taken by HDOS when a channel is .CLEARed as opposed to being .CLOSEed. There is no reason to avoid the use of this SCALL, since it is fully functional. It appears to this author that the programmers working on HDOS 2.0 had some plans to change the action of the .CLEAR SCALL in a future release of HDOS, and they chose not to document the .CLEARA SCALL because it might also change. Obviously, .CLEARA is only a 'convenience' for rare situations when you have multiple temporary output files open (or multiple files open for reading).

.....

.ERROR - Print Error Message (Octal 57Q)

```
*       .ERROR - PRINT ERROR MESSAGE.
*
*       ERROR IS CALLED TO PRINT AN ERROR MESSAGE.
*
*       THE HDOS SYSTEM RETURNS ERROR CODE NUMBERS WHEN
*       IT DETECTS AN ERROR.  THE ERROR FUNCTION MAY BE
*       USED TO TYPE AN ALPHABETICAL EXPLANATION OF THE ERROR.
*
*       THE ERRORS ARE STORED IN THE FILE 'ERRORMSG.SYS'
*       ON THE SYSTEM DISK, ONE MESSAGE PER LINE.
*       THE LINES LOOK LIKE:
*
*       NNNTEXT
*
*       FOR EXAMPLE:
*
*       002END OF MEDIA
*
*       IF THE ERROR MESSAGE FILE CANNOT BE READ, OR THE
*       MESSAGE DOES NOT APPEAR, THE ERROR IS TYPED AS
*
*       'SYSTEM ERROR # NNN'
*
```

PART 5 - RESIDENT SCALLS (Cont)

.ERROR - Print Error Message (Octal 57Q)(Cont)

```
*
*       ENTRY   (A) = ERROR CODE
*               (H) = TRAILING CHARACTER (TYPED AFTER MESSAGE)
*       EXIT    NONE
*       USES    ALL
```

Notes:

Use the .ERROR SCALL to look up an error message in the system error message file "ERRORMSG.SYS." Since HDOS returns all error messages as numbers, this function allows you to easily inform the user in English, and tell him just what went wrong. Also note that if you have a program which needs to generate a large number of messages, you can add them to "ERRORMSG.SYS." Of course, this is not a supported use of the .ERROR SCALL and may not work with future HDOS releases.

An example of the use of the .ERROR SCALL is shown in the example of the .NAME SCALL.

.CHFLG - Change File Flag(s) (Octal 60Q)

```
***     .CHFLG - CHANGE FILE FLAGS.
*
*       CHFLG IS CALLED TO CHANGE THE FILE DESCRIPTION FLAGS
*       FOR A MASS STORAGE FILE.  ONLY CERTAIN FLAGS MAY BE
*       CHANGED:
*
*       FLAG    BIT         MEANING
*
*       DIF.SYS 200Q        IS SYSTEM FILE
*       DIF.LOC 100Q        LOCKED FOR CHANGE (SETABLE ONLY)
*       DIF.WP  040Q        IS WRITE PROTECTED
*
*       CHFLG WILL REFUSE THE OPERATION IF THE DIF.LOC BIT IS SET.
*
*       ENTRY   (B) = NEW BIT VALUES
*               (C) = CHANGE MASK (BIT SET FOR EVERY BIT
*                     TO REPLACE FROM (B))
*               (DE) = DEFAULT BLOCK ADDRESS
*               (HL) = FILE NAME

*
```

PART 5 - RESIDENT SCALLS (Cont)

.CHFLG - Change File Flag(s) (Octal 60Q)(Cont)

```
*
*       EXIT    'C' CLEAR, CHANGE DONE
*               'C' SET, ERROR
*                (A) = ERROR CODE
*       USES    ALL
```

Notes:

Use the .CHFLG SCALL to change the attribute flags on a file. These flags are discussed in detail in the HEATH HDOS Software Reference Manual, Chapter 4, SYSCMD/Plus, page 4-11. The arguments are similar to the .OPEN SCALLs. Note that a two-byte "bits to effect" and "new bit values" scheme is used, just as described earlier for the .CONSL SCALL.

NOTE

You can use the .CHFLG SCALL to set the DIF.LOC (LOCKed) flag on a file, but you cannot use it to clear the flag. Once the DIF.LOC flag is set, no other flag changes may be made, including the clearing of the DIF.LOC flag. If the file is not write-protected (DIF.WP not set), you can copy it to a temp file, delete the old LOCKed version, and rename the temp file back. If the file is both LOCKed and write-protected, then it is there "forever" or until the disk is reinitialized.

```
**      EXAMPLE:
*
*       WRITE PROTECT 'OUTPUT.DAT'

WRIPRO  MVI     B,DIF.WP    EFFECT WRITE PROTECT
        MVI     C,DIF.WP    SET WRITE PROTECT
        LXI     D,DEFALT
        LXI     H,NAME
        SCALL   .CHFLG
        JC      ERROR       ERROR
        .
        .
        .
NAME    DB      'SY1:OUTPUT.DAT',0     FILE NAME
```

PART 5 - RESIDENT SCALLS (Cont)

.DISMT - Flag System Disk As Dismounted (61Q)

```
***     .DISMT - FLAG SYSTEM DISK DISMOUNTED
*
*       THE DISMT FUNCTION IS USED WHEN THE SYSTEM DISK IS
*       ABOUT TO BE DISMOUNTED.  ANY HDOS FUNCTIONS WHICH
*       REQUIRE SYSTEM FILES WILL BE TREATED AS FATAL
*       SYSTEM ERRORS.
*
*       ENTRY   None
*       EXIT    S.SYSM = LWA OF FREE SPACE FOR USER
*               (HL) = (S.SYSM)
*       USES    ALL
```

.LOADD - Load Device Driver (Octal 62Q)

```
***     .LOADD - LOAD DEVICE DRIVER
*
*       LOADD LOADS THE SPECIFIED DEVICE DRIVER.
*
*       ENTRY   (HL)    = DEVICE DRIVER DESCRIPTOR STRING.
*       EXIT    (PSW)   = 'C' CLEAR IF OK
*                         'C' SET IF ERROR
*
*       USES    ALL
```

Notes:

Use the .LOADD system call to load a specified device driver in memory without opening a file on the device. Like the .LOADO system call, this system call is not to be used when SY0: is to be dismounted. If a device driver is not in memory at the time SY0: is dismounted (because it was not loaded and no channel is currently open on the device), subsequent references to the device will generate unknown device errors.

Examples of this SCALL are found in Part 8, and in the example below:

```
        LXI     H,DEVICE
        SCALL   .LOADD
        JC      ERROR
        .
        .
DEVICE  DB      'LP:',0
```

PART 5 - RESIDENT SCALLS (Cont)

.TASK - Communicates with TMG (Octal 101Q)

```
***     TASK - EXTERNAL SCALL FOR THE TASK MANAGER.
*
*       TASK IS THE WAY THE USER COMMUNICATES WITH TMG,
*       THE TASK MANAGER.
*
*       ENTRY:  See TASKDEF.ACM
*       EXIT:   See TASKDEF.ACM
*       USES:   See TASKDEF.ACM

.TASK   EQU     101Q    41H     65
```

.TDU - External SCALL for Terminal DeBugging Utility (Octal 122Q)

```
***     TDU - EXTERNAL SCALL FOR TERMINAL DEBUGGING UTILITY.
*
*       TDU is a basic debugging task which allows the user to examine
*       and alter the state of his CPU.
*
*       THE USER MUST FIRST START THE TDU TASK WITH THE COMMAND:
*
*               ST TDU
*
*       ENTRY INTO TDU IS THEN ACHIEVED BY THE FOLLOWING:
*
*               .
*               user code
*               .
*               SCALL   .TDU            ; This is a breakpoint
*               .
*               user code
*               .
*
*       ENTRY:  NONE
*       EXIT:   NONE
*       USES:   Any registers the user alters

.TDU    EQU     122Q    52H     82
```

PART 5 - RESIDENT SCALLS (Cont)

.LOG - External SCALL Used With the ECHO Task (177Q)

```
***     LOG - EXTERNAL SCALL USED WITH THE ECHO TASK.
*
*       LOG IS THE SOFTWARE METHOD TO TOGGLE THIS TASK ON AND OFF.
*
*       ENTRY:  A = 0   Turn the task off
*                 = 1   Turn the task on
*                 > 1   Will case error
*       EXIT:   PSW     'C' clear if NO error
*                       'C' set   if    error
*                         A = EC.ILC if ECHO is NOT resident
*                         A = -1     if illegal value was sent in A
*       USES:   A,F,H,L

.LOG    EQU     177Q    7FH     127
```

.MOUNT - Mount A Disk (Octal 200Q)

```
***     .MOUNT - MOUNT A DISK
*
*       MOUNT DISK ON SPECIFIED UNIT OF SELECTED DEVICE.
*
*       ENTRY   (HL)    = ADDRESS OF DEVICE SPECIFICATION
*       EXIT    (PSW)   = 'C' SET IF ERROR
*                         'C' CLEAR IF NO ERROR
*                          (A) = ERROR CODE
*                           'Z' CLEAR IF AN ABORT
*       USES    ALL
```

Notes:

Use the .MOUNT system call to mount additional devices. The device specified must not have a volume already mounted on it. If it does, a successful dismount must be issued before a .MOUNT may be processed. The devices currently supported are SY0:, SY1:, SY2:, DK0:, DK1:, and DK2:. This system call also prints a message informing the user that a volume has been mounted. This message is the same one used with the HDOS "MOUNT" command. This call will also verify that the disk structure is not corrupt. If the disk is corrupt, it will not be mounted, and an error message will be returned. If you do not want the message, you may issue the .MONMS system call.

For a more detailed example of .MOUNT, refer to Part 8.

PART 5 - RESIDENT SCALLS (Cont)

.DMOUN - Dismount A Disk (Octal 201Q)

```
***     .DMOUN - DISMOUNT A DISK
*
*       DISMOUNT DISK ON SELECTED DRIVE.
*
*       ENTRY   (HL)    = ADDRESS OF DEVICE SPECIFICATION
*       EXIT    (PSW)   = 'C' SET IF ERROR
*                           (A) = ERROR CODE

*
*       USES    ALL
```

Notes:

Use the .DEMOUN system call to dismount diskettes. After the volume has been successfully dismounted, it will also print a message verifying that the volume has been dismounted. The device to be dismounted must have a volume currently mounted. If it does not, .DMOUN returns an error.

If the volume to be dismounted is the system volume, you must observe several precautions. Device drivers not currently in memory at the time of the dismount will be considered nonexistent. Subsequent references to drivers so marked will generate unknown device errors. You may load a device driver by opening a channel on the device or by ".LOADD"ing it. Even if the current program will not use the device, the device must be loaded before you dismount the system volume if any subsequent programs are to use it.

Before you dismount a disk, you must clear all of the I/O channels open to that disk. Remember that the program itself is left open on channel -1 (377Q), and this channel must be closed before you dismount the system disk (SY0:).

.MONMS - Mount A Disk With No Message (Octal 202Q)

```
***     MONMS - MOUNT/NO MESSAGE
*
*       MOUNT SPECIFIED UNIT OF SELECTED DEVICE WITHOUT ISSUING
*       A MOUNT MESSAGE.
*
*       ENTRY   (HL)    = ADDRESS OF DEVICE SPECIFICATION
*       EXIT    (PSW)   = 'C' SET IF ERROR
*                           (A) =ERROR CODE
*                         'C' CLEAR IF NO ERROR
*                          'Z' CLEAR IF AN ABORT
*       USES    ALL
```

PART 5 - RESIDENT SCALLS (Cont)

.MONMS - Mount A Disk With No Message (Octal 202Q)(Cont)

Notes:

In versions of HDOS later than Version 1.5 the .MONMS system call is identical to the .MOUNT system call, except that .MONMS prints no error message. In the future, this may not be the case. In all likelihood, this will be changed to a "quick" mount which neither prints the message nor verifies the disk structure.

.DMNMS - Dismount A Disk With No Message (Octal 203Q)

```
***     .DMNMS - DISMOUNT A DISK WITH NO MESSAGE
*
*       DISMOUNT SELECTED UNIT OF SPECIFIED DEVICE WITHOUT
*       ISSUING MESSAGE.
*
*       ENTRY   (HL)    = ADDRESS OF DEVICE SPECIFICATION
*       EXIT    (PSW)   = 'C' SET IF ERROR
*                            (A)  = ERROR CODE
*       USES    ALL
```

Notes:

The .DMNMS system call is virtually identical to the .DMOUN call, except for the printing of the dismount message. In future releases, this will probably be changed to some form of "quick" dismount.

.RESET - Reset A Disk (Octal 204Q)

```
***     RESET - RESET A DISK
*
*       RESET THE SPECIFIED UNIT OF THE SELECTED DEVICE
*       BY ISSUING A DISMOUNT FOLLOWED BY A MOUNT.
*       THE DEVICE NAME SHOULD BE IN THE SAME FORMAT AS
*       THAT EXPECTED BY A MOUNT AND DISMOUNT.
*
```

PART 5 - RESIDENT SCALLS (Cont)

.RESET - Reset A Disk (Octal 204Q)(Cont)

```
*       ENTRY:  (HL) = ADDRESS OF DEVICE SPECIFICATION
*       EXIT:   (PSW) = 'C' CLEAR IF NO ERROR
*               'C' SET IF ERROR
*                     (A) = Error Code
*
*       Uses:   ALL
```

.RESMNS - Reset A Disk With No Message (OCTAL 205Q)

```
***     RESMNS - RESET A DISK WITH NO MESSAGE.
*
*       ENTRY:  (HL) = ADDRESS OF DEVICE SPECIFICATION
*               (BC) = ADDRESS OF USER MESSAGE
*                 0 = NO MESSAGE
*                -1 = USE STANDARD MESSAGE
*       EXIT:   'C' SET IF ERROR
*                (A) = ERROR CODE
*
*       Uses:   ALL
```

.DAD - Dismount all Disks (Octal 206Q)

```
***     DAD     - DISMOUNT ALL DISKS
*
*       DAD DISMOUNTS ALL DISKS, THAT IS, ALL MOUNTED VOLUMES OF
*       ALL CURRENTLY MOUNTED DISKS.  THIS IS USUALLY USED IN
*       PREPARATION FOR DISMOUNTING THE SYSTEM DISK.
*
*       THIS IS INCLUDED AS A SYSTEM CALL SINCE THE .EXIT CALL
*       NEEDS THIS CODE.  EVERYONE ELSE MIGHT AS WELL HAVE ACCESS
*       TO IT.
*
*       ENTRY   NONE
*       EXIT    'NC' IF NO ERROR
*               'C' IF ERROR
*               A  = ERROR CODE
*       USES    ALL
```

PART 5 - RESIDENT SCALLS (Cont)

.DAD - Dismount All Disks (Octal 206Q)(Cont)

Notes:

This SCALL is used to dismount ALL units of ALL directory devices known to HDOS. Note that this includes the system disk drive SY0: as well as all the other disks that have been mounted. Therefore, if this SCALL is executed and the program issues an .EXIT SCALL afterwards, it will require the user to reboot HDOS. Since HDOS 3.0 is already set to stand-alone operation, reboot will not be necessary.

SUMMARY OF HDOS 3.0 SYSTEM CALLS

Name		OCT	HEX	DEC	Description
.EXIT	EQU	0Q	0H	0	Exit to SYSCMD.SYS
.SCIN	EQU	1Q	1H	1	System Console Input
.SCOUT	EQU	2Q	2H	2	System Console Output
.PRINT	EQU	3Q	3H	3	Print Text
.READ	EQU	4Q	4H	4	Read Disk Sector(s)
.WRITE	EQU	5Q	5H	5	Write Disk Sector(s)
.CONSL	EQU	6Q	6H	6	Console Attributes
.CLRCO	EQU	7Q	7H	7	Clear Console Buffer
.LOADO	EQU	10Q	8H	8	Load Overlay (OBSOLETE)
.VERS	EQU	11Q	9H	9	Version
.GDA	EQU	12Q	AH	10	Get Driver Address
.CRC16	EQU	13Q	BH	11	CRC Checksum of Block
.LINK	EQU	40Q	20H	32	Link to a Program
.CTLC	EQU	41Q	21H	33	Control A, B & C
.OPENR	EQU	42Q	22H	34	Open for Read
.OPENW	EQU	43Q	23H	35	Open for Write

SUMMARY OF HDOS 3.0 SYSTEM CALLS

Name	OCT	HEX	DEC	Description
.OPENU	EQU	44Q	24H	36 Open for Update
.OPENC	EQU	45Q	25H	37 Open Contiguous
.CLOSE	EQU	46Q	26H	38 Close a channel
.POSIT	EQU	47Q	27H	39 Position within a File
.DELETE	EQU	50Q	28H	40 Delete a File
.RENAME	EQU	51Q	29H	41 Rename a File
.SETTOP	EQU	52Q	2AH	42 Set Top of User Memory
.DECODE	EQU	53Q	2BH	43 Decode File Name
.NAME	EQU	54Q	2CH	44 Get Name from Channel
.CLEAR	EQU	55Q	2DH	45 Clear a Channel
.CLEARA	EQU	56Q	2EH	46 Clear All Channels except -1
.ERROR	EQU	57Q	2FH	47 Print Error Message
.CHFLG	EQU	60Q	30H	48 Change File Flag(s)
.DISMT	EQU	61Q	31H	49 Flag System Disk as Dismounted
.LOADD	EQU	62Q	32H	50 Load Device Driver
.TASK	EQU	101Q	41H	65 Used with Task Manager
.TDU	EQU	122Q	52H	82 Used with Terminal Debugger
.LOG	EQU	177Q	7FH	127 Used with Echo Task
.MOUNT	EQU	200Q	80H	128 Mount a Disk
.MONMS	EQU	202Q	82H	130 Mount a Disk - No Messages
.DMNMS	EQU	203Q	83H	131 Dismount a Disk - No Messages
.RESET	EQU	204Q	84H	132 Reset a Drive
.RESNMS	EQU	205Q	85H	133 Reset a Drive - No Messages
.DAD	EQU	206Q	86H	134 Dismount All Devices

PART 6 - HDOS SYMBOL DEFINITIONS

As we stressed in earlier sections of this manual, there are numerous advantages to using symbolic definitions when you are interfacing to the operating system. This section lists suggested common decks which contain the appropriate symbolic definitions.

To obtain access to these definitions, simply insert the pseudo ops:

```
    XTEXT    HDOSDEF              XTEXT    ASCII
    XTEXT    HOSEQU               XTEXT    ECDEF
```

into the initial statements of your program. This will cause the assembler to process, as required, the statements in the file HDOS.ACM, thus defining those symbols for that assembly.

Note that the assembler will not normally list the contents of any file read by XTEXT. However, by using the

```
    LON    C
```

pseudo op, or the

```
    /LON:C
```

switch when you are using the assembler, you can cause a listing of all files read by XTEXT to be written to the listing file.

.....

The following data is extracted from selected source code .ACM files. Refer to the on-disk library of source code files for further information.

HDOS 3.0 COMMON DECK CONTENTS

HOSDEF.ACM Contents

```
HOSDEF  SPACE   3,10
**      HOSDEF - DEFINE HOS PARAMETER.
*

VERS    EQU     3*16+0      CURRENT VERSION = 3.0
SYSCALL EQU     377Q        SYSCALL INSTRUCTION
        ORG     0

*       RESIDENT FUNCTIONS

.EXIT   DS      1       0Q  EXIT (MUST BE FIRST)
.SCIN   DS      1       1Q  SCIN
.SCOUT  DS      1       2Q  SCOUT
.PRINT  DS      1       3Q  PRINT
.READ   DS      1       4Q  READ
.WRITE  DS      1       5Q  WRITE
.CONSL  DS      1       6Q  SET CLEAR CONSOLE OPTIONS
.CLRCO  DS      1       7Q  CLEAR CONSOLE BUFFER
.LOADO  DS      1      10Q  LOAD AN OVERLAY (OBSOLETE)   /30a/
.VERS   DS      1      11Q  RETURN HDOS VERSION NUMBER
.GDA    DS      1      12Q  GET DEVICE DRIVER ADDRESS    /30a/
.CRC16  DS      1      13Q  CRC A BLOCK OF MEMORY        /30a/
.SYSRES DS      1           PRECEDING FUNCTIONS ARE RESIDENT

*       HDOSOVL0.SYS FUNCTIONS (Note: HDOS 3.0 Has No Overlays)

.LINK   DS      1      40Q  LINK
.CTLC   DS      1      41Q  CTL-C
.OPENR  DS      1      42Q  OPENR
.OPENW  DS      1      43Q  OPENW
.OPENU  DS      1      44Q  OPENU
.OPENC  DS      1      45Q  OPENC
.CLOSE  DS      1      46Q  CLOSE
.POSIT  DS      1      47Q  POSITION
.DELET  DS      1      50Q  DELETE
.DELETE       EQU     .DELET
.RENAM  DS      1      51Q  RENAME
.RENAME       EQU     .RENAM
.SETTP  DS      1      52Q  SETTOP
.SETTOP       EQU     .SETTP
.DECODE DS      1      53Q  DECODE
.NAME   DS      1      54Q  NAME
.CLEAR  DS      1      55Q  CLEAR
.CLEARA DS      1      56Q  RESERVED
.ERROR  DS      1      57Q  LOOKUP ERROR

.CHFLG  DS      1      60Q  CHANGE FLAGS
.DISMT  DS      1      61Q  FLAG SYSTEM DISK DISMOUNTED
.LOADD  DS      1      62Q  LOAD DEVICE DRIVER
```

HDOS 3.0 COMMON DECK CONTENTS (Cont)

HOSDEF.ACM Contents (Cont)

```
*      HDOSOVL1.SYS FUNCTIONS (NOTE: HDOS 3.0 Has No Overlays)

               ORG     200Q
.MOUNT         DS      1     200Q  MOUNT (MUST BE FIRST)
.DMOUN         DS      1     201Q  DISMOUNT
.DMOUNT        EQU     .DMOUN
.MONMS         DS      1     202Q  MOUNT/NO MESSAGE
.DMNMS         DS      1     203Q  DISMOUNT/NO MESSAGE
.RESET         DS      1     204Q  RESET = DISMOUNT/MOUNT UNIT
.RESNMS        DS      1     205Q  RESET/OPTIONAL MESSAGE
.DAD           DS      1     206Q  DISMOUNT ALL DISKS
```

HOSEQU.ACM Contents

```
         SPACE   4,10
**      HDOS SYSTEM EQUIVALENCES.                              /3.0a/
*

        ORG     040100A

S.EXITA DS      8               ; Jump to System Exit
D.CON   DS      16              ; Disk Constants
SYDD    EQU     *               ; System Disk Entry Point
D.VEC   DS      24*3            ; H17 Disk Vectors
D.RAM   DS      31              ; H17 Disk Work Area
S.VAL   DS      36              ; SYSTEM VALUES
S.INT   DS      147             ; SYSTEM INTERNAL WORK AREAS
S.SOVR  DS      2               ; STACK OVERFLOW WARNING
        DS      042200A-*       ; SYSTEM STACK
STACKL  EQU     *-S.SOVR        ; STACK SIZE
STACK   EQU     *               ; LWA+1 SYSTEM STACK
USERFWA EQU     *               ; USER FWA

*       Ensure Compatibility

        ERRNZ   040130A-SYDD
        ERRNZ   040277A-S.VAL
        ERRNZ   040343A-S.INT
        ERRNZ   042200A-USERFWA
```

```
**      THE FOLLOWING SYMBOLS ARE USED BY THE .CONSL SCALL

CSL.ECH EQU     10000000B   SUPPRESS ECHO
CSL.RAW EQU     00000100B   RAW MODE I/O
CSL.WRP EQU     00000010B   WRAP LINES AT WIDTH
CSL.CHR EQU     00000001B   OPERATE IN CHARACTER MODE
```

HDOS 3.0 COMMON DECK CONTENTS (Cont)

```
**      THE FOLLOWING SYMBOLS ARE USED BY THE .CONSL SCALL (Cont)

I.CSLMD EQU     0           CONSOLE MODE

CTP.BKS EQU     10000000B   TERMINAL PROCESSES BACKSPACES
CTP.FF  EQU     01000000B   TERMINAL PROCESSES FORMFEED
CTP.MLI EQU     00100000B   MAP LOWER CASE TO UPPER ON INPUT
CTP.MLO EQU     00010000B   MAP LOWER CASE TO UPPER ON OUTPUT
CTP.2SB EQU     00001000B   TERMINAL NEEDS TWO STOP BITS
CTP.HHS EQU     00000100B   TERMINAL USES HARDWARE HANDSHAKE
CTP.BKM EQU     00000010B   MAP BKSP (UPON INPUT) TO RUBOUT
CTP.TAB EQU     00000001B   TERMINAL SUPPORTS TAB CHARACTERS

I.CONTY EQU     1           S.CONTY IS 2ND BYTE
I.CUSOR EQU     2           S.CUSOR IS 3RD BYTE
I.CONWI EQU     3           S.CONWI IS 4TH BYTE

CO.FLG  EQU     00000001B   CTL-O FLAG
CO.FLG  EQU     10000000B   CTL-S FLAG
```

.....

ASCII.ACM Contents

```
        SPACE   3,10
**      ASCII CHARACTER EQUIVALENCES.

NUL     EQU     000Q            ; null
BELL    EQU     007Q            ; bell
BKSP    EQU     010Q            ; backspace
BS      EQU     BKSP
TAB     EQU     011Q            ; horizontal tab
LF      EQU     012Q            ; line feed
NL      EQU     012Q            ; new line (HDOS)
FF      EQU     014Q            ; form feed
CR      EQU     015Q            ; carriage return
ESC     EQU     033Q            ; escape
DEL     EQU     177Q            ; delete

*       Specials

EOL     EQU     200Q            ; end of line flag
NULL    EQU     200Q            ; pad character
NUL2    EQU     0               ; ditto
RUBOUT  EQU     DEL             ; rubout/delete
C.SYN   EQU     026Q            ; SYNC
C.STX   EQU     002Q            ; STX
QUOTE   EQU     047Q            ; quote character (")
ENL     EQU     NL+EOL          ; NL + end-of-line flag
```

HDOS 3.0 COMMON DECK CONTENTS (Cont)

ASCII.ACM Contents (Cont)

```
*       Control keys

CTLA    EQU     'A'-'@'          ; CTRL/A
CTLB    EQU     'B'-'@'          ; CTRL/B
CTLC    EQU     'C'-'@'          ; CTRL/C
CTLD    EQU     'D'-'@'          ; CTRL/D
CTLE    EQU     'E'-'@'          ; CTRL/E
CTLO    EQU     'O'-'@'          ; CTRL/O
CTLP    EQU     'P'-'@'          ; CTRL/P
CTLQ    EQU     'Q'-'@'          ; CTRL/Q
CTLR    EQU     'R'-'@'          ; CTRL/R
CTLS    EQU     'S'-'@'          ; CTRL/S
CTLX    EQU     'X'-'@'          ; CTRL/X
CTLZ    EQU     'Z'-'@'          ; CTRL/Z
```

TYPTX.ACM Contents

```
$TYPTX  SPACE   4,10
**      $TYPTX - TYPE TEXT.
*
*       $TYPTX IS CALLED TO TYPE A BLOCK OF TEXT TO THE SYSTEM
*       CONSOLE, WHERE THE BLOCK OF TEXT TO BE TYPED IMMEDIATELY
*       FOLLOWS THE SUBROUTINE CALL INSTRUCTION.
*
*       $TYPTX. PERFORMS THE SAME FUNCTION AS $TYPTX, EXCEPT THAT
*       USES REGISTER H,L AS A POINTER TO THE TEXT WHICH IS TO
*       BE TYPED.
*
*       IMBEDDED ZERO BYTES INDICATES A CARRIAGE RETURN LINE FEED.
*       A BYTE WITH THE 200Q BIT SET IS THE LAST BYTE IN THE MESSAGE.
*
*       EMTRY   (RET) = TEXT
*       EXIT    TO (RET+LENGTH)
*       USES    A,F

$TYPTX  EQU     031136A                 ; IN H17 ROM
                TAKES TEXT IMMEDIATELY

$TYPTX. EQU     031144A                 ; IN H17 ROM
                TAKES TEXT FROM HL REGISTER

EXAMPLE:

$TYPTX          TAB IN ONE TAB STOP, OR CALL TAB OVER.
                TAB CALL TAB $TYPTX<RTN>
                TAB DB TAB 'HELLO, WORLD',
                212Q<RTN>
```

HDOS 3.0 COMMON DECK CONTENTS (Cont)

TYPTX.ACM Contents (Cont)

```
$TYPTX.         TAB LXI TAB H,MSG<RTN>
                TAB CALL TAB $TYPTX.
```

MOVE.ACM Contents

```
MOVE    SPACE   4,10
**      $MOVE - MOVE DATA
*
*       $MOVE MOVES A BLOCK OF BYTES TO A NEW MEMORY ADDRESS.
*       IF THE MOVE IS TO A LOWER ADDRESS, THE BYTES ARE MOVED FROM
*       FIRST TO LAST.
*
*       IF THE MOVE IS TO A HIGHER ADDRESS, THE BYTES ARE MOVED FROM
*       LAST TO FIRST.
*
*       THIS IS DONE SO THAT AN OVERLAPED MOVE WILL NOT 'RIPPLE'.
*
*       ENTRY   (BC) = COUNT
*               (DE) = FROM
*               (HL) = TO
*       EXIT    MOVED
*               (DE) = ADDRESS OF NEXT FROM BYTE
*               (HL) = ADDRESS OF NEXT *TO* BYTE
*               'C' CLEAR
*       USES    ALL

$MOVE   EQU     030252A                 ; IN H17 ROM
```

ECDEF.ACM Contents

```
        SPACE   4,10
**      ERROR CODE DEFINITIONS.

        ORG     0

        DS      1           ; NO ERROR #0
EC.EOF  DS      1           ; END OF FILE
EC.EOM  DS      1           ; END OF MEDIA
EC.ILC  DS      1           ; ILLEGAL SCALL CODE
EC.CNA  DS      1           ; CHANNEL NOT AVAILABLE
EC.DNS  DS      1           ; DEVICE NOT SUITABLE
EC.IDN  DS      1           ; ILLEGAL DEVICE NAME
EC.IFN  DS      1           ; ILLEGAL FILE NAME
EC.NRD  DS      1           ; NO ROOM FOR DEVICE DRIVER
```

HDOS 3.0 COMMON DECK CONTENTS (Cont)

ECDEF.ACM Contents (Cont)

```
EC.FNO  DS      1           ; CHANNEL NOT OPEN
EC.ILR  DS      1           ; ILLEGAL REQUEST
EC.FUC  DS      1           ; FILE USAGE CONFLICT
EC.FNF  DS      1           ; FILE NAME NOT FOUND
EC.UND  DS      1           ; UNKNOWN DEVICE
EC.ICN  DS      1           ; ILLEGAL CHANNEL NUMBER
EC.DIF  DS      1           ; DIRECTORY FULL
EC.IFC  DS      1           ; ILLEGAL FILE CONTENTS
EC.NEM  DS      1           ; NOT ENOUGH MEMORY
EC.RF   DS      1           ; READ FAILURE
EC.WF   DS      1           ; WRITE FAILURE
EC.WPV  DS      1           ; WRITE PROTECTION VIOLATION
EC.WP   DS      1           ; DISK WRITE PROTECTED
EC.FAP  DS      1           ; FILE ALREADY PRESENT
EC.DDA  DS      1           ; DEVICE DRIVER ABORT
EC.FL   DS      1           ; FILE LOCKED
EC.FAO  DS      1           ; FILE ALREADY OPEN
EC.IS   DS      1           ; ILLEGAL SWITCH
EC.UUN  DS      1           ; UNKNOWN UNIT NUMBER
EC.FNR  DS      1           ; FILE NAME REQUIRED
EC.DIW  DS      1           ; DEVICE IS NOT WRITABLE (OR WRITE-LOCKED)
EC.UNA  DS      1           ; UNIT NOT AVAILABLE
EC.ILV  DS      1           ; ILLEGAL VALUE
EC.ILO  DS      1           ; ILLEGAL OPTION
EC.VPM  DS      1           ; VOLUME PRESENTLY MOUNTED ON DEVICE
EC.NVM  DS      1           ; NO VOLUME PRESENTLY MOUNTED
EC.FOD  DS      1           ; FILE OPEN ON DEVICE
EC.NPM  DS      1           ; NO PROVISIONS MADE FOR REMOUNTING
                              MORE DISKS
EC.DNI  DS      1           ; DISK NOT INITIALIZED
EC.DNR  DS      1           ; DISK IS NOT READABLE
EC.DSC  DS      1           ; DISK STRUCTURE IS CORRUPT
EC.NCV  DS      1           ; NOT CORRECT VERSION OF HDOS
EC.NOS  DS      1           ; NO OPERATING SYSTEM MOUNTED
EC.IOI  DS      1           ; ILLEGAL OVERLAY INDEX
EC.OTL  DS      1           ; OVERLAY TOO LARGE
EC.LAD  DS      1           ; File is locked against delete      /3.0a/
EC.FIX  DS      1           ; Device media is fixed              /3.0a/
EC.ILA  DS      1           ; Illegal Load Address               /3.0a/
EC.DNL  DS      1           ; Device Not Loaded                  /3.0a/
EC.DNP  DS      1           ; Device Not Locked in Memory        /3.0a/
EC.DFM  DS      1           ; Device is Fixed in Memory          /3.0a/
EC.IDF  DS      1           ; Illegal Date Format                /3.0a/
EC.ITS  DS      1           ; Illegal Time Format                /3.0a/
EC.CNR  DS      1           ; System Clock Not Resident          /3.0a/
EC.SDR  DS      1           ; System Disk is Reset               /3.0a/
EC.LDO  DS      1           ; Line Buffer Overflow               /3.0a/
EC.CUI  DS      1           ; Can't Unlink from Interrupt Vector /3.0a/
EC.PNG  DS      1           ; Permission Not Given               /3.0a/
.....................................................................
```

HDOS SYMBOL VALUES

This section contains a list of split-octal values for the HDOS symbols discussed in this document. These values are presented as a double check, so you can compare them to the values generated when you assemble the common decks. Once again, it is important that you use the common decks and use symbolic values, rather than using the octal values directly.

ECDEF Symbol Definitions

```
 EC.CNA = 000004A          EC.FNO = 000011A          EC.NCV = 000050A
*EC.CNR = 000064A          EC.FNR = 000034A          EC.NEW = 000021A
*EC.CUI = 000067A          EC.FOD = 000043A          EC.NOS = 000051A
 EC.DDA = 000027A          EC.FUC = 000013A          EC.NPM = 000044A
*EC.DFM = 000061A          EC.HIN = 000000A          EC.NRD = 000010A
 EC.DIF = 000017A          EC.1CN = 000016A          EC.NVM = 000042A
 EC.DIW = 000035A         *EC.IDF = 000062A          EC.OTL = 000053A
 EC.DNI = 000045A          EC.IDN = 000006A          EC.RF  = 000022A
*EC.DNL = 000057A          EC.IFC = 000020A         *EC.SDR = 000065A
*EC.DNP = 000060A          EC.IFN = 000007A          EC.UNA = 000036A
 EC.DNR = 000046A         *EC.ILA = 000056A          EC.UND = 000015A
 EC.DNS = 000005A          EC.ILC = 000003A          EC.UUN = 000033A
 EC.DSC = 000047A          EC.ILO = 000040A          EC.VPM = 000041A
 EC.EOF = 000001A          EC.ILR = 000012A          EC.WF  = 000023A
 EC.EOM = 000002A          EC.ILV - 000037A          EC.WP  = 000025A
 EC.FAO = 000031A          EC.IOI = 000052A          EC.WFV = 000024A
 EC.FAP = 000026A          EC.IS  = 000032A
*EC.FIX = 000055A         *EC.ITS = 000063A
 EC.FL  = 000030A         *EC.LAD = 000054A
 EC.FNF = 000014A         *EC.LBO = 000066A
```

* New for HDOS 3.0

HOSDEF Symbol Definitions

```
 .CHFLG  =  000060A        .EXIT  =  000000A        .PRINT  =  000003A
 .CLEAR  =  000055A       *.GDA   =  000013A        .READ   =  000004A
 .CLEARA =  000045A        .LINK  =  000040A        .RENAME =  000051A
 .CLOSE  =  000046A       *.LOADD =  000062A        .RESET  =  000204A
 .CLRCO  =  000007A        .LOADO =  000010A       *.RESMNS =  000250A
 .CONSL  =  000006A       *.LOG   =  000177A        .SCIN   =  000001A
*.CRC16  =  000013A        .MONMS =  000202A        .SCOUT  =  000002A
 .CTLC   =  000041A        .MOUNT =  000200A        .RESMNS =  000250A
 .DAD    =  000207A        .NAME  =  000054A        .SETTOP =  000052A
 .DECODE =  000053A        .OPENC =  000045A       *.TASK   =  000101A
 .DELETE =  000050A        .OPENR =  000042A       *.TDU    =  000122A
 .DMNMS  =  000203A        .OPENU =  000044A        .VERS   =  000011A
 .DMOUN  =  000201A        .OPENW =  000043A        .WRITE  =  000005A
 .ERROR  =  000057A        .POSIT =  000047A

  CO.FLG  =  000001A        CS.FLG  = 000200A        CSL.CHR =  000001A
```

HDOS SYMBOL VALUES

HOSDEF Symbol Definitions (Cont)

```
  CSL.ECH =  000200A        CSL.WRP = 000002A        CTP.2SB =  000010A
  CTP.BKM =  000002A        CTP.BKS = 000200A        CTP.ML1 =  000040A
  EC.DDA  =  000027A        EC.DIF  = 000017A        EC.DIW  =  000035A

  EC.CNA  =  000004A        EC.FOD  = 000043A        EC.NRD  =  000010A
  EC.CNR  =  000064A        EC.FUC  = 000013A        EC.NTF  =  000311A
  EC.CUI  =  000067A        EC.ICN  = 000016A        EC.NTM  =  000003A
  EC.DDA  =  000027A        EC.IDF  = 000062A        EC.NVM  =  000042A
  EC.DFM  =  000061A        EC.IDN  = 000017A        EC.OTL  =  000053A
  EC.DIF  =  000017A        EC.IFC  = 000020A        EC.PNG  =  000070A
  EC.DIW  =  000035A        EC.IFN  = 000007A        EC.RF   =  000022A
  BC.DNI  =  000045A        EC.ILA  = 000056A        EC.SDR  =  000065A
  EC.DNL  =  000057A        EC.ILC  = 000003A        EC.TAA  =  000301A
  EC.DNP  =  000060A        EC.ILO  = 000040A        EC.TCA  =  000305A
  EC.DNR  =  000046A        EC.ILR  = 000012A        EC.TIF  =  000304A
  EC.DNS  =  000005A        EC.ILV  = 000037A        EC.TNA  =  000302A
  EC.DSC  =  000047A        EC.IOI  = 000052A        EC.TNF  =  000307A
  EC.EOF  =  000001A        EC.IS   = 000032A        EC.TSN  =  000036A
  EC.EOM  =  000002A        EC.ITF  = 000300A        EC.TUN  =  000303A
  EC.FAO  =  000031A        EC.ITS  = 000063A        EC.UNA  =  000036A
  EC.FAP  =  000026A        EC.LAD  = 000054A        EC.UND  =  000015A
  EC.FIX  =  000055A        EC.LB0  = 000066A        EC.UUN  =  000033A
  EC.FL   =  000030A        EC.NCV  = 000050A        EC.VPM  =  000041A
  EC.FNF  =  000014A        EC.NEM  = 000021A        EC.WF   =  000023A
  EC.FNO  =  000011A        EC.NOS  = 000051A        EC.WP   =  000025A
  EC.FNR  =  000034A        EC.NPM  = 000044A        EC.WPV  =  000024A

                            I.CONFL = 000004A        I.CONTY =  000001A
  I.CONWI =  000003A        I.CSLMD = 000000A        I.CUSOR =  000002A
  S.DATC  =  040310A        S.DATE  = 040277A        S.HIMEM =  040316A
  S.OMAX  =  040324A        S.SYSM  = 040320A        S.USRM  =  040322A
```

* New for HDOS 3.02

HOSEQU Symbol Definitions

```
USERFWA = 42200A
STACK   = 42200A

CO.FLG  = 000001A         CTP.MLO = 000020A         I.CUSOR = 000002A

CS.FLG  = 000200A         CTP.TAB = 000001A         S.DATC  = 040310A
CSL.CHR = 000001A         CTP.2SB = 000010A         S.DATE  = 040277A
CSL.ECH = 000200A         I.CONFL = 000004A         S.HIMEM = 040316A
CSL.WRP = 000002A         I.CONTY = 000001A         S.OMAX  = 040324A
CTP.BKM = 000002A         I.CONWI = 000003A         S.SYSM  = 040320A
CTP.BKS = 000200A         I.CSLMD = 000000A         S.USRM  = 040322A
CTP.MLI = 000040A
```

PART 7 - PROLOGUE.SYS

A PROLOGUE.SYS is a special program generated in Assembly Language.

Appendix 13-A contains an example of the HDOS "PROLOGUE.SYS" feature incorporated into assembly language source code.

When HDOS is started at the BOOT command, it will attempt to transfer control to SY0:PROLOGUE.SYS. If such a file cannot be found, control is transferred to "SYSCMD.SYS." The use of a PROLOGUE.SYS sets up special automatic tasking parameters at bootup time, such as loading device drivers, or starting menu programs in MBASIC or Assembly Language. If the processing of your PROLOGUE.SYS program is not desired at boot time, just simply type IGNORE when HDOS states: <BOOT><B>? instead of the usual <RTN>.

****

PART 8 - PROGRAMMING EXAMPLE

Refer to pages 13-72 thru 13-76 for typical programming examples.

** CAUTION **

The following files, being typical source code files, extend beyond the normal 80 column line! They cannot be either viewed by the standard text editor, nor can they be printed at the normal 10 or 12 CPI. Therefore, you must take the measures indicated below.

There are two options: (1) This material may be monitored on the screen using an HDOS editor such as EDIT19 which permits viewing of the file contents extending beyond 80 columns. (2) You can print out the following files on a line printer, but you must use a 17 CPI instead of the customary 10 or 12 CPI.

(1) Screen Option:

To scroll the screen so that it goes off the left side, while in the SCREEN mode of EDIT19, Version 3.10, hit the ENTER key and type LEFT 60<RTN>. This will cause your screen to move 60 columns to the left, and you will be able to see the hidden part of the listing. To return the screen to normal, once more hit the ENTER key and type LEFT 0<RTN>. If your version of EDIT19 is earlier than 3.10, substitute the command "DISCARD" for "LEFT" as indicated above. The same technique is used.

(2) Print Option:

Set your printer according to the following settings and print as usual. You will receive a smaller printout, but the entire listing will be visible on the page. CPI=17, LPI=8, FORM=88, PAGE=80, LMARG 15.

****

APPENDIX 13-A: PROGRAMMING EXAMPLES

Menu Prologue for MBASIC - Listing 1:

```
                    00002   ***     MENU Prologue
                    00003   *
                    00004   *       SOURCE: The HEATH Company
                    00005   *
                    00006   *       This Prologue:
                    00007   *               Loads Device Drivers (If Present)
                    00008   *                       LP:
                    00009   *                       LT:
                    00010   *                       LD:
                    00011   *                       AT:
                    00012   *               Runs MBASIC establishing 5 file buffers
                    00013   *               Runs the MBASIC program "MENU.BAS"
                    00014   *
                    00015   *       Note:   The command line may be easily modified to
                    00016   *               accomodate other files, etc., by changing
                    00017   *               the line pushed on the stack at "PROB."
                    00018   *
                    00019
042.200             00020           XTEXT   ASCII
042.200             00049           XTEXT   HOSDEF
000.205             00117           XTEXT   HOSEQU
                    00123
030.252             00124   $MOVE   EQU     30252A    These are routines in the H17 ROM
031.136             00125   $TYPTX  EQU     31136A
                    00126
                    00127
042.200             00128           ORG     USERFWA
                    00129
042.200             00130   START   EQU     *
                    00131
                    00132                *       Load the device drivers
                    00133
042.200  041 027 043 00134   LOAD1   LXI     H,PROAA
042.203  377 062     00135           SCALL   .LOADD                Load the device driver
042.205  332 226 042 00136           JC      LOAD2                 No load, skip message
042.210  315 136 031 00137           CALL    $TYPTX
042.213  114 120 072 00138           DB      'LP: Loaded',ENL
                    00139
042.226  041 033 043 00140   LOAD2   LXI     H,PROAB
042.231  377 062     00141           SCALL   .LOADD
042.233  332 254 042 00142           JC      LOAD3
042.236  315 136 031 00143           CALL    $TYPTX
042.241  114 104 072 00144           DB      'LD: Loaded',ENL
                    00145
042.254  041 037 043 00146   LOAD3   LXI     H,PROAC
042.257  377 062     00147           SCALL   .LOADD
042.261  332 302 042 00148           JC      LOAD4
042.264  315 136 031 00149           CALL    $TYPTX
042.267  114 124 072 00150           DB      'LT: Loaded',ENL
                    00151
042.302  041 043 043 00152   LOAD4   LXI     H,PROAD
042.305  377 062     00153           SCALL   .LOADD
042.307  332 330 042 00154           JC      PSTACK
042.312  315 136 031 00155           CALL    $TYPTX
042.315  101 124 072 00156           DB      'AT: Loaded',ENL
042.330  041 000 000 00157
                    00158   * Push pseudo command line on user stack for MBASIC to find
                    00159
042.330  041 000 000 00160   PSTACK  LXI     H,0
042.333  071         00161           DAD     SP            *    HL = current stack value
000.012             00162           SET     PROBE-PROB+1
042.334  021 366 377 00163           LXI     D,-,     * DE = - (Number of bytes to push)
042.337  031         00164           DAD     D
042.340  371         00165           SPHL                       Reserve the stack space
```

APPENDIX 13-A: - PROGRAMMING EXAMPLES (Cont)

Menu Prologue for MBASIC - Sample 1 (Cont)

```
                    00166
042.341  001 012 000 00167           LXI     B,PROBE-PROB+1
042.021  021 047 043 00168           LXI     D,PROB
042.347  315 252 030 00169           CALL    $MOVE                   Move the stuff onto the stack
                    00170
                    00171   *       Link to MBASIC
                    00172
042.352  041 061 043 00173           LXI     H,PROC
042.355  377 040     00174           SCALL   .LINK                   Try to run MBASIC
                    00175
042.357  315 136 031 00176           CALL    $TYPTX
042.362  007 105 122 00177           DB      BELL,'ERROR - Unable to execute MBASIC',ENL
                    00178
043.024  257         00179   EXIT    XRA     A
043.025  377 000     00180           SCALL   .EXIT                   Normal EXIT
                    00181
043.027  114 120 072 00182   PROAA   DB      'LP:',0
043.033  114 104 072 00183   PROAB   DB      'LD:',0
043.037  114 124 072 00184   PROAC   DB      'LT:',0
043.043  101 124 072 00185   PROAD   DB      'AT:',0
                    00186
043.047  040 115 105 00187   PROB    DB      'MENU/F:5',0
043.060             00188   PROBE   EQU     *-1
                    00189
043.061  123 131 060 00190   PROC    DB      'SY0:MBASIC.ABS',0
                    00191
043.100  000         00192           END     START

00192 Statements Assembled
32007 Bytes Free
No Errors Detected
```

Example 1: Menu Prologue for MBASIC [B]

```
                    00002   ***     MENU Prologue
                    00003   *
                    00004   *       SOURCE: The HEATH Company
                    00005   *
                    00006   *       This Prologue:
                    00007   *               Loads Device Drivers (If Present)
                    00008   *                       LP:
                    00009   *                       LT:
                    00010   *                       LD:
                    00011   *                       AT:
                    00012   *               Runs MBASIC establishing 5 file buffers
                    00013   *               Runs the MBASIC program "MENU.BAS"
                    00014   *
                    00015   *       Note:   The command line may be easily modified to
                    00016   *               accomodate other files, etc., by changing
                    00017   *               the line pushed on the stack at "PROB."
                    00018   *
                    00019
042.200             00020           XTEXT   ASCII
042.200             00049           XTEXT   HOSDEF
000.205             00117           XTEXT   HOSEQU
                    00123
                    00124
042.200             00125           ORG     USERFWA
                    00126
042.200             00127   START   EQU     *
```

```
                    00128
                    00129   *       Load the device drivers
                    00130
042.200  041 027 043 00131   LOAD1   LXI     H,PROAA
042.203  377 062     00132           SCALL   .LOADD                  LOAD DEVICE DRIVER
042.205  332 226 042 00133           JC      LOAD2                   ERROR LOADING LP:
042.210  315 136 031 00134           CALL    $TYPTX
042.213  114 120 072 00135           DB      'LP: LOADED',ENL
                    00136
042.226  041 033 043 00137   LOAD2   LXI     H,PROAB
042.231  377 062     00138           SCALL   .LOADD
042.233  332 254 042 00139           JC      LOAD3
042.236  315 136 031 00140           CALL    $TYPTX
042.241  114 104 072 00141           DB      'LD: LOADED',ENL
                    00142
042.254  041 037 043 00143   LOAD3   LXI     H,PROAC
042.257  377 062     00144           SCALL   .LOAD
042.261  332 302 042 00145           JC      LOAD4
042.264  315 136 031 00146           CALL    $TYPTX
042.267  114 124 072 00147           DB      'LT: LOADED',ENL
                    00148
042.342  041 043 043 00149   LOAD4   LXI     H,PROAD
042.305  377 062     00150           SCALL   .LOAD
042.307  332 330 042 00151           JC      PSTACK
042.312  315 136 031 00152           CALL    $TYPTX
042.315  101 124 072 00153           DB      'AT:' LOADED',ENL
                    00154
                    00155   *       Push pseudo command line on the user stack for MBASIC to find
                    00156
042.330  041 000 000 00157   PSTACK  LXI     H,0
042.333  071         00158           DAD     SP                      HL = current stack value
000.012               00159   *       SET     PROBE-PROB+1
042.334  021 366 377 00160           LXI     D,_.                    DE = - ( Number of bytes to
push )
042.337  031         00161           DAD     D
042.340  371         00162           SPHL                            Reserve the stack space
                    00163
042.341  001 012 000 00164           LXI     B,PROBE-PROB+1
042.344  021 047 043 00165           LXI     D,PROB
042.347  315 252 030 00166           CALL    $MOVE           Move the stuff into the stack space
                    00167
                    00168   *       Link to MBASIC
                    00169
042.352  041 061 043 00170           LXI     H,PROC
042.355  377 040     00171           SCALL   .LINK                   Try to run MBASIC
                    00172
042.357  335 136 031 00173           CALL    $TYPTX
042.362  007 105 122 00174           DB      BELL,'ERROR - Unable to Execute MBASIC',ENL
                    00175
043.024  257         00176   EXIT    XRA     A
043.025  377 000     00177           SCALL   .EXIT                   Normal EXIT
                    00178
043.027  114 120 072 00179   PROAA   DB      'LP:',0
043.033  114 104 072 00180   PROAB   DB      'LD:',0
043.037  114 124 072 00181   PROAC   DB      'LT:',0
043.043  101 124 072 00182   PROAD   DB      'AT:',0
                    00183
043.047  040 115 105 00184   PROB    DB      ' MENU/F:5',0
043.060               00185   PROBE   DB      *-1
                    00186
043.061  123 131 060 00187   PROC    DB      'SY0:MBASIC.ABS',0
                    00188
043.100               00189           XTEXT   $TYPTX
043.100               00207           XTEXT   $MOVE
043.100  000         00232           END     START

00232 Statements Assembled
18320 Bytes Free
No Errors Detected
```

APPENDIX 13-A: PROGRAMMING EXAMPLES (Cont)

Example 2: Prologue for a One-Drive System

```
                    00002   ***     PROLOGUE.H89 - PROLOGUE FOR A ONE DRIVE HDOS SYSTEM
                    00003   *
                    00004   *       SOURCE: The HEATH Company
                    00005   *
                    00006   *       THIS PROLOGUE AUTOMATICALLY RESETS DRIVE SY0: SO THAT THE
                    00007   *       USER MAY INSERT ANOTHER DISK.
                    00008   *
                    00009
042.200             00010           XTEXT   ASCII
042.200             00039           XTEXT   HOSDEF
000.205             00107           XTEXT   HOSFQU
                    00113
042.200             00114           ORG     USERFWA
                    00115
042.200  041 243 042 00116   START   LXI     H,CCINT
042.203  076 003     00117           MVI     A,CTLC
042.205  377 041     00118           SCALL   .CTLC                   SET UP CONTROL-C PROCESSING
                    00119
042.207  076 377     00120           MVI     A,377Q
042.211  377 055     00121           SCALL   .CLEAR                  CLOSE THE CHANNEL THAT WE CAME
IN ON
042.213  076 000     00122
042.215  377 010     00123
042.217  332 271 042 00124
042.222  076 001     00125
042.224  337 010     00126
042.226  332 271 042 00127
042.231  041 261 042 00128           LXI     H,PROB
042.234  377 062     00129           SCALL   .LOADD                  LOAD LP.DVD
042.236  041 265 042 00130           LXI     H,PROC
042.241  377 062     00131           SCALL   .LOAD                   LOAD AT.DVD
                    00132
042.243  041 254 042 00133   CCINT   LXI     H,PROA
042.246  377 204     00134           SCALL   .RESET
                    00135
042.250             00136   EXIT    EQU     *
042.250  076 001     00137           MVI     A,1
042.252  377 000     00138           SCALL   .EXIT                   EXIT AND ABORT
                    00139
042.254  123 131 060 00140   PROA    DB      'SY0:',0
042.261  114 120 072 00141   PROB    DB      'LP:',0
042.265  101 124 072 00142   PROC    DB      'AT:',0
                    00143
042.271  046 012     00144   ERROR   MVI     H,NL
042.273  377 057     00145           SCALL   .ERROR
042.275  303 250 042 00146           JMP     EXIT
                    00147
042.300             00148           XTEXT   $TYPTX
                    00166
o42.300  000        00167           END     START

00167 Statements Assembled
18390 Bytes Free
No Errors Detected
```

****

APPENDIX 13-A: PROGRAMMING EXAMPLES (Cont)

Example 3: Prologue for a 3-drive System:

PROLOGUE.H17 - PROLOGUE FOR A THREE-DRIVE SYSTEM

```
                    00002   ***     PROLOGUE.H17 - PROLOGUE FOR A THREE DRIVE HDOS SYSTEM
                    00003   *
                    00004   *       SOURCE: 1980, HEATH Company
                    00005   *
                    00006   *       THIS PROLOGUE IS TO BE RUN UPON BOOTUP OF A THREE DRIVE
SYSTEM.
                    00007   *       IT AUTOMATICALLY MOUNTS THE DISK IN SY1: AND SY2:.
                    00008   *
                    00009
042.200             00010           XTEXT   ASCII
042.200             00039           XTEXT   HOSDEF
000.205             00107           XTEXT   HOSEQU
                    00113
042.200             00114           ORG     USERFWA
                    00115
042.200             00116   START   EQU     *
                    00117
                    00118   *       Initialize Control-C processing
                    00119
042.200  041 221 042 00120           LXI     H,CCINT
042.203  076 003     00121           MVI     A,CTLC
042.205  377 041     00122           SCALL   .CTLC                   Set up Control-C Processing
                    00123
                    00124   *       Mount the Disks
                    00125
042.207  041 225 042 00126           LXI     H,PROA
042.212  377 200     00127           SCALL   .MOUNT
                    00128
042.214  041 232 042 00129           LXI     H,PROB                  Mount SY2:
042.217  377 200     00130           SCALL   .MOUNT                  (Delete for a 2 drive system.)
                    00131
                    00132   *       EXIT
                    00133
042.221  076 001     00134   CCINT   MVI     A,1
042.223  377 000     00135           SCALL   .EXIT                   EXIT and ABORT
                    00136
042.225  123 131 061 00137   PROA    DB      'SY1:',0
042.232  123 131 062 00138   PROB    DB      'SY2:',0
                    00139
042.237  000         00140           END     START

00140 Statements Assembled
18431 Bytes Free
No Errors Detected
```

****

APPENDIX 13-B:
HDOS Device Drivers Programmers' Guide

by

Al Dallas (70250,637),
Dale Lamm (70555,302), and
Tom Jorgenson (70120,153)
Richard Musgrave

With Permission from the Heath Company
Originally published in REMark #20.
Modified for HDOS 3.0

VERSION 1.2

Introduction

What is a device driver? Under HDOS, a device driver is a relocatable program (usually less than 3K) which the operating system loads in order to communicate with external devices. When programmers speak of HDOS as a "high-level" or sophisticated operating system, one of the things they have in mind is this device-independence, which makes HDOS adaptable to just about any peripheral equipment.

Originally, device drivers were a method of providing software support for the Heath line of printers. The console and disk device drivers were built-in to HDOS, because 1) they constitute a minimum system, 2) their functions are more sophisticated than those of printers or punches, and 3) it was assumed that end users would not need access to them. Users, it seems, have surprised quite a few with their knowledge, programming ability, and above all, their desire for access to everything about their computer. Heath's introduction of the H47 8" disk drives demonstrated a need for greater flexibility for mass-storage devices as well as printers. The result was version 2.0 of HDOS, which allows device drivers for "mass storage devices", such as disk drives.

Obviously, a device driver is necessary in order to use HDOS and system utilities with a peripheral device. A functioning device driver incorporates the entire system; i.e., SYSCMD's COPY and CAT commands will work with the new device, as will PIP and even MBASIC (provided the driver is loaded first). This is very powerful, as it amounts to "patching" the entire operating system and high-level languages to suit potentially unique external equipment. It also opens the door to psuedo-device driver development, such as a software clock.

We assume the reader has a good knowledge of assembly language programming techniques. The relocatable aspect of the driver code coupled with the numerous system communication parameters, flags and "magic" addresses make device drivers challenging to the uninitiated. Registered owners of HDOS 2.0 have several Software Tools at their disposal -- Heath supplies several device drivers in source code form along with a plethora of .ACM files describing system equates and other useful data.

APPENDIX 13-B:
HDOS Device Drivers Programmers' Guide (Cont)
++

Environment
~~~~~~~~~~~

The basic HDOS Memory Map includes HDOS tables and data resident in high memory, device driver space below that, and user code below that. Device drivers must be LOADed at the command level (by SYSCMD) in order to allocate space permanently. Loading from within a program (by using the .LOADD system call) does not 'lock' the driver in memory. The routine to do that is explained later.

At boot-time, HDOS builds a Device Table in memory by scanning the Directory for two-letter files with the .DVD extension. It reads in each file's header and checks that it is flagged as a device driver, in an attempt to keep prying hands from creating illicit device drivers. HDOS must be re-booted to re-build this table, so remember to re-boot after assembling, debugging, or renaming a device driver. Curiously, the device table includes the physical sector of the device driver and therefore deleting a .DVD file on SY0: and then referencing it can confuse things badly. HDOS has verified that a valid .DVD file exists at a particular sector which the GRT now flags as available for new files -- prudent programmers will avoid this situation by simply re-booting after all creation, deletion or change of device drivers. Note also that device drivers on disks other than SY0: are not "known" to the system because they are not included in the table.

User programs access any new devices just the same as typical Heath devices. Performing an .OPENW with HL pointing to 'CK:' as the name is just as valid as 'AT:' as the name, though what the driver chooses to do with the data it is sent may be totally different. To better lace the ties with the operating system, HDOS informs the driver as it performs each step. For instance, a MOUNT CK: command causes HDOS to first load the device driver, then check device ready status, and finally mount the device. At each step, as HDOS finishes what it has to do, the driver code is entered and the appropriate function is requested, so that the driver code is informed of the action. The SY: driver, for example, doesn't care about the OPEN command -- it returns 'no error' to HDOS, where the real work is done (setting up tables, flags, etc.). However, an LP: device cannot be mounted, so if this function is requested, the driver must flag an Illegal Function error upon return. MBASIC is even simpler. Any device can be used as a data sink, provided its driver defines it as capable of WRITE, and the driver was first LOADed from the command mode. Just 'OPEN "O",1,"DV:", and start PRINTing data to #1.
~~~~~~~~~~~

APPENDIX 13-B:
HDOS Device Drivers Programmers' Guide (Cont)

PIC Header

PIC stands for Position Independent Code, probably the biggest hurdle to overcome for the potential device driver programmer. Device Drivers are, for all intents and purposes ORG'd at address 6. Why 6 and not 0 in a minute. When HDOS loads the driver into memory, its exact location will vary based on a number of factors including the amount of RAM in the computer. Therefore, HDOS must relocate (change all the addresses) in the driver before running (entering) it, so that the various JMPs and CALLs know where they're going. To keep track of which bytes need relocation, the assembler treats PIC code differently and generates a relocation table of bytes at the end of the actual program. Routines in HDOS process this table and change the necessary addresses.

Binary files in HDOS are stored with a header, or descriptor, to flag the file type and note the starting and entry addresses and the length of code. For this purpose, Heath has supplied the PICDEF.ACM file, which (starting at 0) generates a six-byte header, and creates the offset mentioned above. The header format is:

Byte	Value	Description
0	377Q	Binary File Flag
1	1	File Type = PIC
2,3		Length of Code + Rel Table
4,5		Address of Start of Rel Table

These are the actual bytes -- assembly language (as opposed to machine language) programmers do not refer to them directly. It is important to refer to actual values and absolute addresses by their symbolic names because the code is much more easily adapted to future releases (it also helps reduce a common source of errors). The only way to learn these techniques is by observing examples -- and the Heath device drivers and XTEXTs are especially good. Insert the PICDEF.ACM as an XTEXT last thing before the first actual program instructions, followed by CODE PIC in the opcode and operand fields. The CODE PIC pseudo must come before the start of the program (so that the entire program will be relocatable), but after all the external definitions (such as H17 ROM addresses) which are not to be relocated.

The INIT program supplied with HDOS is capable of initializing just about any mass-storage device when passed certain parameters. MAKMSD.ABS is a program to concatenate your xx.DVD file with a xxINIT.SYS file containing these parameters and device-specific initialization routines. One format for the xxINIT.SYS file is:

512-byte Read-Only Boot Driver (org'd at 42200A)

Sub-Functions
Media Initialization

APPENDIX 13-B:
HDOS Device Drivers Programmers' Guide (Cont)

Pic Header (Cont)

Volume Parameters
Cluster Sizes
Directory Offsets

Studying the SYINIT and DDINIT examples distributed with HDOS 2.0 will better explain how these files must be configured, but there is considerable latitude.

The first 17 bytes of the file constitute the device header, defined as follows:

Byte	Value	Description
0	307Q	Device Driver Flag
1		Device Capabilities Byte
2		Mounted Units Mask
3		Maximum Number of Units
4-11		Unit Capabilities for 0 - 7
12	307Q	Device Driver Flag (if device will take Set options)
13-14		Pointer to INIT code (set by MAKMSD)
15	307Q	Device Driver Flag (if driver is specific to HDOS 3.0)
16		Set Preamble Size

Bear in mind that these are the actual bytes, not the symbolic values. It is dangerous to ignore the Heath XTEXTs, because future compatibility requires their use. The actual bytes are shown here as an aid to understanding what the XTEXTs accomplish. The Device Driver flag is apparently just an arbitrary 11000111 pattern used to validate the driver code. The Device Capability Byte is defined:

Bit: 7 6 5 4 3 2 1 0

0	Directory-Type Device
1	Capable of Read
2	Capable of Write
3	Capable of Random-Access
4	Capable of Character Mode

These capability codes are defined in the DEVDEF.ACM file supplied by Heath. The Mounted Units Mask is typically defined as 0 for variable numbers of units, or 1 for devices with only one unit. The Maximum Number of Units is 8 (0..7) for any device, but your driver can set this to any value between 1 and 8. For each valid unit, the unit capability is flagged as explained above, and each invalid unit is flagged with a zero. These 13 bytes are used directly in building the Device Table entry in HDOS. Another 20 bytes are reserved by Heath prior to the Set Entry Point, by use of the symbolic SET Entry Point, DVD.STE.

APPENDIX 13-B:
HDOS Device Drivers Programmers' Guide (Cont)

Set Entry

The SET.ABS program is used to patch device driver files, among several other things. The address 53Q is considered the SET entry point, and the next 469 bytes are reserved for SET processing. To assure that these important addresses are maintained, Heath device drivers make use of the error-checking abilities of ASM to flag an error if the correct address is not produced. DVD.STE is defined as 53Q, and this code follows the header bytes:

```
        .       SET     025Q     (SET is used as an ASM psuedo here)
                ERRNZ   *-.
                DS      DVD.STE-.

        SETN    EQU     *
                ERRNZ   *-DVD.STE
```

A 'P' error is generated while assembling the program if this critical address (SETN) is not 53Q, but the technique allows the value of DVD.STE to be changed at any time, reassembling, and preserving the intent of the code. An undocumented feature of the 2.0 Assembler is the ability to turn relocation on and off, and therefore legally SET a value to the Origin pointer (*), thus:

```
                CODE    -REL     Turn Relocation Off
        .       SET     *
                CODE    +REL     Turn Relocation On
                DS      DVD.ENT-.
```

These procedures are not mandatory, they simply represent good programming practice, make the code easier to update, and easier for others to work on.

The SET program loads the device driver into a convenient location in memory and relocates the first 512 bytes (the Header and the SET processor). It then enters the driver's SET portion, passing a unit number parameter in A, and a pointer to the rest of the command line in DE. From here, your program can do anything you want, except that memory values thus updated must be in the part of the code that was not relocated. Only this portion is read back to the disk by SET when done, to save having to un-relocate the set code. Rather than free-wheeling, however, due to the limited space for set functions, most Heath drivers make use of routines in the SET.ABS program itself to process the various options. These useful routines are documented in Appendix C.

The primary routine, $SOP, is the Set Option Processor. $SOP is called with BC pointing to the command line, DE pointing to a processor table, and HL pointing to an option table. $SOP matches the command line to an option in the table, uses the index found in this table to fetch the

APPENDIX 13-B:
HDOS Device Drivers Programmers' Guide (Cont)

SET Entry (Cont)

processor (sub-routine) address, and then jumps to that processor with HL pointing to any additional data in the option table and BC pointing to the rest of the command line. The option table is defined as follows:

```
        DW      End of Table Address
        DB      Number of Data Bytes following option
        DB      'SEARCH STRIN','G'+200Q
        DB      Index into Processor Table (8 Bit)
        DB      Additional Data Bytes (N - 1)
        DB      'NEXT STRIN','G'+200Q

        DB      0       (End of Table)

        The processor table lists routines:

        DW      HELP
        DW      FLAG... etc.
```

Refer to the Heath Device Drivers for the assembly language methods used to implement these tables and provide for modification ease, documentation, and compatibility with future releases.

SET commands should include a HELP command which prints a list of valid commands. Typically, SET commands will either set a flag bit in a variable somewhere, or change a value. To assist these operations, the SET.ABS program includes the $PBF and $PBV routines. Both routines are compatible with $SOP, so that the FLAG or VAL processors listed in the processor table need only jump to $PBF or $PBV. The difference is evident in the option table data structure. $PBF expects at least 5 data bytes following the option string and $PBV expects 6.

```
        $PBV Data Bytes:
           DB   Default Radix (2,8,10)
           DB   Minimum Value
           DB   Maximum Value
           DW   Address of Variable

        $PBF Data Bytes:
           DB   Mask (Bits to Alter)
           DB   Bit Pattern to Set
           DW   Address of Variable
           DB   0 (if 6 data bytes are used)
```

$PBV is quite sophisicated. The Default Radix is used unless the user specifies B, D, Q, etc. after the value. $PBF uses a fail-safe mask just as the .CONSL SCALL does (see the System Programmer's Guide to HDOS). The remaining SET routines are described in the new SETCAL.ACM included as Appendix C.

APPENDIX 13-B:
HDOS Device Drivers Programmers' Guide (Cont)

Driver Entry

Starting at DVD.ENT (2000A, typically), the remainder of the device driver code is just that -- device driver. HDOS enters at this address with (A) equal to a Device Communication code as defined in DDDEF.ACM. If (A) exceeds DC.MAX, the driver must flag an Illegal Request error. Functions which are logical for the device in question (Write for a printer, for example) must be directed to appropriate processors. However, inappropriate functions must return errors to HDOS. In a gray area between are functions which are not erroneous, but at the same time require no processing by the driver. These functions simply return no error to HDOS.

READ enters with a byte count in BC (typically a multiple of 256), a buffer address in DE (to which the data must be read), and a block number in HL. A block number is equal to a logical sector number (i.e., 320 as opposed to Track 32, Sector 0). A serial device simply ignores any value in HL.

WRITE is the same as READ, except that DE points to the data to be written out to the device.

READ REGARDLESS is anachronistic. It involves reading the label sector on the disk, disregarding volume number protection. Chances are good your driver can either map it to READ, return no error, or return a Device Not Suitable error without processing it at all.

OPENR opens a file for read. Disk drivers typically ignore all OPENs, but a tape device driver might use OPENR as a signal to rewind a data tape.

OPENW opens a file for write. The LP: device driver, for instance, uses this routine to initialize and prepare the device, a function that would probably be handled by READY if the driver had been first written under 2.0.

OPENU opens a file for update (random read/write). You most likely will not have to deal with this, in that the really tricky part is handled by HDOS.

CLOSE presents a good opportunity to dump a buffer out to a printer, but disk drivers typically ignore it.

ABORT cancels the current operation. The SY: driver resets the device, seeks track zero, and exits with no error flagged. LP: flags a Device Driver Abort error before leaving, however.

MOUNT is used by the SY: driver to set up volume protection and to seek track zero. (Register L = the volume number at entry). LP: ignores this routine.

APPENDIX 13-B:
HDOS Device Drivers Programmers' Guide (Cont)

Driver Entry (Cont)

LOAD is used by SY: to initialize constants in system RAM, re-vector obsolete ROM code, etc. This is a new function for 2.0.

READY is another function added for 2.0. Your code should perform some test to verify that your device is ready and return no error to HDOS. 'C' set indicates that the device is not ready, and HDOS will provide the loop -- this way, HDOS remains cognizant of interrupt requests as opposed to hanging up in your routine.

The following functions added in HDOS 3.0. Refer to the supplied device driver source code for descriptions and examples.

Update Set Parameters
Unload Device Driver
Interrupt Service
Device Specific Function

Often, a device driver may want, under certain circumstances, to load itself permanently in memory. The following code from SYSCMD.SYS explains the simple process.

```
        LHLD    S.SYSM          Update System FWA
        SHLD    S.RFWA
        LHLD    AIO.DTA         Get Device Table Address
        LXI     D,DEV.REX       Offset to Residency Flag
        DAD     D
        MOV     A,M
        ORI     DR.PR           Set Flag = Perm. Resident
        MOV     M,A
```

These symbols are defined in the DEVDEF.ACM, ESINT.ACM and ESVAL.ACM files. This works because HDOS has variable pointers which are addressing our device driver as it is entered. If the code is to be locked, this routine must be called before any other device I/O is attempted. In fact, it is usually not a good idea to perform system calls from within the device driver, because it was entered using the system call process which is only partially re-entrant.

You may want to include a LON G pseudo in your code just before the end to direct the assembler to list the relocation table. Heath drivers typically include a patch area here as well. With PIC code, entry begins at PIC.COD, so there is no need for an operand with the END statement.

Summary

Writing or modifying a device driver should not be beyond the capabilities of any assembly language programmer. Looking at naked

APPENDIX 13-B:
HDOS Device Drivers Programmers' Guide (Cont)

Summary (Cont)

Heath driver code can be confusing due to the large number of symbolic values assigned in .ACM files elsewhere, but studying these examples is the best way to learn about device drivers. Remember to define all external (non-relocating) addresses before using the CODE PIC psuedo, and to confine the variable accessed by SET to addresses higher than DVD.ENT.

This guide represents Al Dallas' study of device drivers and the HDOS Version 1.6 source code listings, along with many suggestions and helpful guidance of two HDOS wizards, Dale Lamm and Tom Jorgenson. There are no warranties, express or implied and Heath Company takes no responsibility for the data herein.

APPENDIX A
Minimum XTEXTs

DDDEF	.ACM	Device Driver Communication Flags
DEVDEF	.ACM	Capability Flags, etc.
DVDDEF	.ACM	Driver Header Equates
ECDEF	.ACM	Error Code Definitions
ESINT	.ACM	For Direct HDOS operations only
ESVAL	.ACM	Direct operations only
PICDEF	.ACM	PIC Format
SETCAL	.ACM	Routines in SET.ABS

APPENDIX B
Typical Driver Layout

```
0 - PIC.COD     PIC Header
PIC.COD - 20    Driver Header
21 - 42         Reserved
DVD.STE -       SET Code
DVD.ENT-1               Entry Processor
                        Processor Routines
                        Processor Table
                        Option Table
DVD.ENT - ?     Driver Code
                        Entry Processor
                        Processor Routines
                        Sub-Routines
                        Data Area
```

APPENDIX 13-B:
HDOS Device Drivers Programmers' Guide (Cont)

APPENDIX C
SETCAL.ACM

```
SETCAL  SPACE   4,10
**      SETCAL - ROUTINES IN SET.ABS
*

$SNA    SPACE   3,10
**      SNA - SCAN TO NEXT ARGUMENT
*       SNA IS CALLED TO SKIP OVER BLANKS
*
```

APPENDIX C
SETCAL.ACM (Cont)

```
*       ENTRY:  (BC) = LINE POINTER
*       EXIT:   (BC) UPDATED
*               'Z' SET IF AT END OF LINE
*       USES:   A,F,B,C
$SNA    EQU     42201A

$DCS    SPACE   3,10
**      DCS - DELIMIT CHARACTER STRING
*
*       ENTRY:  (BC) = LINE POINTER
*       EXIT:   (BC) UPDATED
*               (DE) = ADDR FIRST STRING CHAR
*               (HL) = ADDR LAST STRING CHAR
*               (A) = STRING LENGTH
*               'Z' SET IF STRING EMPTY
```

APPENDIX C
SETCAL.ACM (Cont)

```
*       USES:   ALL
$DCS    EQU     42204A

$CNA    SPACE   3,10
*       CNA - CONVERT NUMERIC ARGUMENT
*       CNA CONVERTS ARGUMENT IN COMMAND LINE TO
*       A BINARY VALUE
*
*       ENTRY:  (BC) = LINE POINTER
*               (A) = DEFAULT RADIX
*       EXIT:   (BC) = UPDATED
*               (HL) = VALUE
*               'C' SET IF ERROR
```

APPENDIX 13-B:
HDOS Device Drivers Programmers' Guide (Cont)
++

APPENDIX C
SETCAL.ACM (Cont)

```
*       USES:   ALL
$CNA    EQU     42207A

$FST    SPACE   3,10
**      FST - FIND IN SERIAL TABLE
*       FST SEARCHES A SERIAL TABLE FOR A
*       SPECIFIC KEY
*
*       ENTRY:  (HL) = ADDR OF TABLE
*               (DE) = ADDR OF SEARCH KEY
*       EXIT:   (DE) = UNCHANGED
*               'Z' SET IF MATCH FOUND
*       USES:   A,F,H,L
$FST    EQU     42212A
$TBLS   SPACE   3,10
**      TBLS - TABLE SEARCH
*       TABLE FORMAT:
*       DB      KEY1,VAL1
*       *       *
*       *       *
*       DB      KEYN,VALN
*       DB      0

*
*       ENTRY:  (A) = PATTERN
*               (HL) = ADDR OF TABLE
*       EXIT:   (A) = PATTERN IF FOUND
*               'Z' SET IF FOUND
*       USES:   A,F,H,L
$TBLS   EQU     42215A

$WTBLS  SPACE   3,10
**      WTBLS - WORD TABLE SEARCH
*       LOOK-UP WORD VALUE USING 1-BYTE KEY
*       TABLE FORMAT:
*       DB      KEY1
*       DW      VAL1
*       *       *
*       *       *
*       DB      KEYN
*       DW      VALN
*       DB      0
*
*       ENTRY:  (A) = PATTERN
*               (HL) = ADDR OF TABLE
*       EXIT:   (A) = PATTERN IF FOUND
*               'Z' SET IF FOUND
*       USES:   A,F,H,L
```

APPENDIX 13-B:
HDOS Device Drivers Programmers' Guide (Cont)

APPENDIX C
SETCAL.ACM (Cont)

```
$WTBLS  EQU     42220A

$LBD    SPACE   3,10
**      LBD - LOOK UP BAUD RATE DIVISOR
*
*       ENTRY:  (DE) = BINARY BAUD RATE
*       EXIT:   'Z' SET IF VALID BAUD RATE
*               (HL) = DIVISOR
*       USES:   A,F,D,E,H,L
$LBD    EQU     42223A

$SOP    SPACE   3,10
**      SOP - SET OPTION PROCESSOR
*
*       ENTRY:  (BC) = LINE POINTER
*               (DE) = PROCESSOR TABLE ADDRESS
*               (HL) = OPTION TABLE ADDRESS
*       EXIT:   (RET) TO PROCESSOR IF NO ERROR
*               (BC) = UPDATED
*               (HL) = NEXT AVAILABLE DATA BYTE
*       USES:   ALL
$SOP    EQU     42226A

$PBF    SPACE   3,10
**      PBF - PROCESS BYTE FLAG
*
*       ENTRY:  (HL) = ADDR OF TABLE VECTOR
*       EXIT:   'C' SET IF ERROR
*       USES:   ALL
$PBF    EQU     42231A

$PBV    SPACE   3,10
**      PBV - PROCESS BYTE VALUE
*
*       ENTRY:  (BC) = NEXT CHAR ADDRESS
*               (HL) = TABLE VECTOR INDEX
*       EXIT:   (BC) UPDATED
*               'C' SET IF ERROR
*       USES:   ALL
$PBV    EQU     42234A

*****************************************************************************
```

APPENDIX 13-B:
HDOS Device Drivers' Programming Guide (Cont)

```
*       APPENDIX D
*       CK.DVD By Dale Lamm
*
*       TITLE   'Super-Simple Super-Small CK.DVD'
*       STL     'Version 2.1  11-Jun-81  D. Lamm'
*       This  begins  an  elementary  example of a "device driver", in this
*       case, a real-time clock.  For the sake of simplicity, no XTEXTs are
*       used.  Instead, all required symbol definitions are included in the
*       main body of the source code (this file).
*       SPACE   7

DVDFLV  EQU     0C7H            THIS FLAGS TO HDOS AS A DEVICE DRIVER
DVD.ENT EQU     200H            STANDARD DEVICE DRIVER ENTRY POINT
DC.MAX  EQU     11              ELEVEN DRIVER FUNCTIONS CURRENTLY SUPPORTED
DT.CR   EQU     00000010B       FLAG BIT; CAPABLE OF READS
DT.CW   EQU     00000100B       FLAG BIT; CAPABLE OF WRITES
EC.EOF  EQU     01H             ERROR CODE; END OF FILE
EC.FNO  EQU     09H             ERROR CODE; CHANNEL NOT OPEN
EC.ILR  EQU     0AH             ERROR CODE; ILLEGAL REQUEST
EC.FAO  EQU     19H             ERROR CODE; FILE ALREADY OPEN
UIVEC   EQU     201FH           HDOS UIVEC TABLE FROM MTR-88
$TBRA   EQU     193EH           ROM TABLE BRANCH ROUTINE
NL      EQU     0AH             HDOS NEWLINE CHARACTER
CAL     EQU     -1-500          CLOCK CALIBRATION; 500 TICKS=1 SECOND
        SPACE   3
        CODE    PIC
$       EQU     *+DVD.ENT-6     NEED TO DEFINE DVD.ENT AS A RELOCATABLE SYMBOL
        DB      DVDFLV          STICK IN THE DEVICE DRIVER FLAG
        DB      DT.CR+DT.CW     MAKE IT CAPABLE OF READS AND WRITES
        DB      1               MOUNTED UNIT MASK
        DB      1               MAXIMUM NUMBER OF UNITS
        DB      DT.CR+DT.CW     SUB-CAPABILITY IS SAME AS UNIT CAPABILITY
        DS      7               DON'T CARE ABOUT UNITS 2-8 SUB-CAPABILITY
        DB      0               NO SET OPTIONS AVAILABLE
        DS      $-*             RESERVE SPACE UP TO DRIVER'S ENTRY POINT
        STL     'DRIVER ENTRY POINT'
        EJECT
***     CK.DVD  PROCESS ENTRY POINT
*
*       ENTRY:  (A)     = PROCESS CODE
*               (BC)    = BYTE COUNT
*               (DE)    = DATA BUFFER ADDRESS
*
*       EXIT:   (PSW)   = CARRY CLEAR IF NO ERROR
*                       = CARRY SET IF ERROR; ERROR CODE IN (A)
*
*       USES:   DEPENDS ON FUNCTION CALLED
*
*
*
```

APPENDIX 13-B:
HDOS Device Drivers' Programming Guide (Cont)

```
START   EQU     *               HDOS COMES HERE EVERY TIME DRIVER IS CALLED
        CPI     DC.MAX          SEE IF REQUESTED FUNCTION IS UNDEFINED
        JNC     ILLEGAL         IF SO, TREAT AS ILLEGAL REQUEST
        CALL    $TBRA           LET THE TABLE BRANCH ROUTINE FIND CORRECT JUMP
        DB      READ-*          FNCTN 0= READ FROM DEVICE
        DB      WRITE-*         FNCTN 1= WRITE TO DEVICE
        DB      IGNORE-*        FNCTN 2= READ REGARDLESS
        DB      OPREAD-*        FNCTN 3= OPEN FOR READS
        DB      OPWRITE-*       FNCTN 4= OPEN FOR WRITES
        DB      ILLEGAL-*       FNCTN 5= OPEN FOR UPDATES
        DB      CLOSE-*         FNCTN 6= CLOSE CHANNEL TO DEVICE
        DB      CLOSE-*         FNCTN 7= ABORT; TREAT AS CLOSE
        DB      ILLEGAL-*       FNCTN 8= MOUNT DEVICE
        DB      LOAD-*          FNCTN 9= LOAD DEVICE
        DB      IGNORE-*        FNCTN 10= EXAMINE DEVICE READY STATUS
        STL     'DOCUMENTATION'
        EJECT

*       When HDOS calls up a device driver, the accumulator contains the
*       function that HDOS wants the driver to perform.  In HDOS version
*       2.0, there are eleven possible functions.  The clock driver ignores
*       some, treats others as illegal, and processes the rest.  The
*       routine at label "Start" uses the supplied function number to
*       branch into the proper handler, via the ROM routine called "Table
*       Branch," or $TBRA for short.  It is up to the individual function
*       processors to handle errors that occur within a processor.
*
*       In general, if a device driver returns with a set carry bit, HDOS
*       assumes that an error occured.  The single case here where an error
*       is when reading from the clock.  An "End of File" error must
*       happen, or HDOS will continue to read until you abort the driver
*       with CTRL-C.
*
*       The "Load" processor is special in that it can only happen once.
*       When you type "Load CK:" from the HDOS command mode, HDOS will
*       check that the driver's address is in the device table, which was
*       built upon boot-up.  If so, it spools the driver code into memory
*       and enters at the label called "Load".  The load function places a
*       detour in HDOS's normal TICCNT interrupt path which causes control
*       to branch to the label called "Clock" every TICCNT (2 ms
*       intervals).  The routine called "Clock" will determine whether 500
*       of these TICCNTs have occured, and if so, will update the
*       time-of-day held in the storage area labeled "Timebuf."  The "Read"
*       and "Write" functions are the only functions that pull data out of
*       the buffer or insert data into the buffer.
*
```

APPENDIX 13-B:
HDOS Device Drivers' Programming Guide (Cont)

After the Load Processor is finished patching internal vectors, it returns control to HDOS. The operating system then flags the device driver as being permanent in memory. Further "Load CK:" commands are ignored by HDOS. If they were not, the system would crash as soon as the "Load" processor trys to patch vectors that have already been patched. No provision is made to "Unload" the driver after it has been made permanent. The CPU overhead associated with maintaining the correct time-of-day is minimal, and you will not notice any degradation in CPU execution speed in a practical sense.

The "Illegal" function is branched to when HDOS issues an impossible or meaningless command to the driver. The "Ignore" function does just that ... ignore. It is called when the command is meaningless, yet otherwise harmless.

If either the "Read" or "Write" function is executed, they first determine whether there is already file I/O in progress determine whether there is already file I/O in progress for the respective function. If that is the case, an error is returned and the second invocation of "Read" or "Write" is ignored. The "Close" function resets both the "Read" and "Write" functions status flags. Channels open on CK.DVD must be closed before new channels can be opened on the clock driver.
EJECT

Another point that bears explanation is the "Xfercnt" storage cell, used during writes to the clock's buffer. Since we have no way of knowing for sure how many bytes the caller wants to stick in the clock buffer, we have to have a means of limiting insertions to a specific number of bytes, in this case, eight.

When Basic writes to a file, it usually sends along enough null characters to make the file writes multiples of 256 bytes. If we simply moved the supplied bytes into the time buffer and beyond, we would likely crash whatever was loaded in memory above the clock driver (usually another driver). During writes to the time buffer, a running total of how many bytes have already been sent to the buffer is kept in "Xfercnt". As soon as eight bytes have been written, the remainder are simply eaten up and not used.

The "Xfercnt" is reset to eight when the clock driver is once again opened for writes.

Another case for possible trouble is when the clock is open for reads and the caller requests but one byte. If we let the caller get the time out of the buffer in this fashion, chances are that the time will change in between single byte transfers. That is why the "Read" function processor checks to make certain that more than eight bytes have been requested. For the sake of Basics "line

APPENDIX 13-B:
HDOS Device Drivers' Programming Guide (Cont)

```
*       input"  function,  we always send a "newline" immediately after the
*       current time-of-day.  If the caller requests more than nine  bytes,
*       it  is  treated  to  a barrage of nulls until its appetite has been
*       satisfied, or until a full 256  bytes  have  been  sent,  whichever
*       comes  first.   Basic  will  ask  for 256 bytes at a time, but user
*       programs or PIP will ask for varied amounts of  bytes.   After  the
*       requested number of bytes, including nulls, have been supplied, the
*       "Read" processor returns with the carry bit set,  and  an  "End  of
*       File" error code in the accumulator.

*       At  the  end  of  this listing are two Benton Harbor Basic routines
*       that allow you to read or write to the clock.  It is possible, from
*       the  HDOS  command  mode,  to do the same.  If you want to see what
*       time it is, type "COPY TT:=CK:", or more simply, "PIP CK:".

*       If you want to set the clock from the command mode, type this:
*       COPY CK:=TT:      [carriage return]
*       12:34:56          [or whatever]
*       CTL-D
*
*       Note  that  anything  beyond  eight  characters  is  ignored by the
*       "Write" processor inside the clock driver.
*
*       Experienced assembly language programmers will have no  problem  in
*       getting the current time from the driver, or updating the time from
*       their special programs.  The driver is small as far as drivers  go,
*       only about 300 bytes of memory are given up to it.
*
*       Note  finally that ANY interrupt driven clock will lose time if the
*       interrupts are turned off for some reason, such  as  SY:  accesses.
*       The  value  of  the  CAL  factor  has been chosen to yield accurate
*       timekeeping only if the SY:'s are not heavily used  and  the  CPU's
*       clock frequency is exact.

            STL     'CK.DVD FUNCTION PROCESSORS'
            EJECT
ILLEGAL EQU     *           COME HERE IF ILLEGAL DRIVER FUNCTION REQUESTED
        MVI     A,EC.ILR    PUT THE "ILLEGAL REQUEST" ERROR CODE IN (A)
        STC                 SO HDOS KNOWS AN ERROR OCCURRED
        RET                 TO WHOMEVER CALLED THE DRIVER
        SPACE   3,9
IGNORE  EQU     *           COME HERE TO IGNORE A REQUESTED FUNCTION
        XRA     A           CLEAR THE CARRY BIT
        RET                 TO WHOMEVER CALLED THE DRIVER
        SPACE   3,9
LOAD    EQU     *           COME HERE WHEN USER TYPES "LOAD CK:"
        LHLD    UIVEC+1     GET THE HDOS "TICCNT" VECTOR
        SHLD    CLKRET+1    INSTALL AT END OF OUR CLOCK ROUTINE
        LXI     H,CLOCK     GET OUR CLOCK'S START ADDRESS
        SHLD    UIVEC+1     REPLACE THE HDOS "TICCNT" VECTOR
```

APPENDIX 13-B:
HDOS Device Drivers' Programming Guide (Cont)

```
        XRA     A           CLEAR THE CARRY BIT
        RET                 RETURN, SHOWING NO ERROR
        SPACE   3,9
OPREAD  EQU     *           FLAG TO DEVICE THAT A CHANNEL IS OPEN ON IT
        LDA     RSTAT       FETCH OUR READ STATUS FLAG
        ORA     A           SEE IF ALREADY OPEN FOR READS
        JZ      OPREAD1     NOT OPEN, SO SKIP NEXT
        MVI     A,EC.FAO    PUT "FILE ALREADY OPEN" ERROR CODE IN (A)
        STC                 TELL HDOS WE HAVE AN ERROR
        RET
OPREAD1 MVI     A,-1        PUT A 0FFH IN READ STATUS FLAG CELL
        STA     RSTAT       THIS FLAGS THE DEVICE NOW OPEN
        XRA     A           CLEAR CARRY BIT
        RET
        SPACE   3,9
OPWRITE EQU     *           FLAG TO DEVICE THAT A CHANNEL IS OPEN ON IT
        LDA     WSTAT       FETCH OUR WRITE STATUS FLAG
        ORA     A           SEE IF ALREADY OPEN FOR WRITES
        JZ      OPWRIT1     NOT OPEN, SO SKIP NEXT
        MVI     A,EC.FAO    PUT "FILE ALREADY OPEN" ERROR CODE IN (A)
        STC                 TELL HDOS WE HAVE AN ERROR
        RET
OPWRIT1 MVI     A,-1        PUT A 0FFH IN WRITE STATUS FLAG CELL
        STA     WSTAT       THIS FLAGS THE DEVICE NOW OPEN
        MVI     A,8         PUT MAXIMUM BYTE XFER COUNT IN HOLD PLACE
        STA     XFERCNT
        LXI     H,TEMPBUF   POINT AT FIRST ADDRESS IN TEMPORARY BUFFER

        SHLD    BUFPTR      SAVE IT IN POINTER HOLDING PLACE
        XRA     A           CLEAR CARRY BIT
        RET
        SPACE   3,9
CLOSE   EQU     *           FLAG DEVICE DISCONNECTED FROM ACTIVE CHANNELS
        XRA     A           ZERO (A)
        STA     RSTAT       RESET "OPEN FOR READ" FLAG, IF SET
        STA     WSTAT       RESET "OPEN FOR WRITE" FLAG, IF SET
        RET                 WITH A CLEAR CARRY BIT
        SPACE   3,9
MOVE    EQU     *           MOVE (BC) BYTES FROM ((HL)) TO ((DE))
        MOV     A,B         SEE IF BYTE COUNTER AT ZERO
        ORA     C
        RZ                  ALL FINISHED; (DE)=NEXT *TO* ADDRESS
        MOV     A,M         FETCH BYTE TO BE MOVED
        STAX    D           WRITE BYTE VIA (DE)
        INX     H           BUMP *FROM* LOCATION
        INX     D           BUMP *TO* POINTER
        DCX     B           DECREMENT BYTE COUNTER
     B  JMP     MOVE        LOOP UNTIL (BC) DECREMENTED TO ZERO
        STL     'CK.DVD WRITE PROCESSOR'
        EJECT
```

APPENDIX 13-B:
HDOS Device Drivers' Programming Guide (Cont)

```
***     CK.DVD  WRITE PROCESSOR
*
*       ENTRY:  (BC)    = BYTE COUNT; MUST BE EQUAL TO OR GREATER
*                         THAN EIGHT.
*               (DE)    = BUFFER ADDRESS; WHERE NEW TIME STRING IS
*                         COMING FROM.
*       EXIT:   (PSW)   = CARRY CLEAR IF NO ERROR
*                       = CARRY SET IF ERROR; EXIT THROUGH "ILLEGAL"
*
*       USES:   ALL
*
*
WRITE   EQU     *               COME HERE WHEN CALLER WANTS TO WRITE TO DVD
        LDA     WSTAT           FIRST, SEE IF WE'RE OPEN FOR WRITES
        ORA     A
        MVI     A,EC.FNO        PREPARE OURSELVES FOR "CHANNEL NOT OPEN" ERROR
        STC
        RZ                      RETURN IF CHANNEL WAS NOT PREVIOUSLY OPENED
WRITE1  MOV     A,B             SEE IF BYTE COUNT AT ZERO
        ORA     C
        RZ                      RETURN WITH CLEAR CARRY; TIME IN "TEMPBUF"
        LDA     XFERCNT         SEE HOW MANY BYTES WE'VE WRITTEN SO FAR
        ORA     A               SEE IF MAXIMUM NUMBER WRITTEN INTO "TEMPBUF"
        JZ      WRITE2          EIGHT BYTES WRITTEN; NOW MOVE TO REAL BUFFER
        DCR     A               ADJUST TRANSFER COUNT
        STA     XFERCNT         UPDATE TRANSFER COUNT
        LHLD    BUFPTR          GET POINTER INTO "TEMPBUF"
        LDAX    D               FETCH CHARACTER TO BE WRITTEN INTO "TEMPBUF"
        MOV     M,A             PLACE CHARACTER IN "TEMPBUF"
        INX     H               POINT TO NEXT POSITION IN BUFFER
        SHLD    BUFPTR          UPDATE POINTER INTO "TEMPBUF"
        INX     D               BUMP POINTER INTO SOURCE FIELD
        DCX     B               DECREMENT BYTE COUNTER
        JMP     WRITE1          TRY AND TRANSFER SOME MORE CHARACTERS
WRITE2  LXI     H,TEMPBUF       SOURCE FOR MOVE
        LXI     D,TIMEBUF       DESTINATION OF MOVE
        LXI     B,8             HOW MANY TO MOVE
        DI                      DON'T WANT "TICCNTS" TO MESS US UP !
        CALL    MOVE            PUT TIME IN THE REAL "TIMEBUF"
        EI                      ENABLE ALL INTERRUPTS
        RET                     CARRY CLEARED; (BC)=ZERO
        SPACE   3

*       This  looks  kludgy, first writing the time into a temporary buffer
*       then putting it into the actual time buffer, but if a user  program
*       for  some  reason  only  transfers  one byte at a time, we risk the
*       chance of a TICCNT updating the actual buffer while we are  writing
*       into it.  The time is moved into the actual buffer only after eight
*       bytes have been placed into the temporary buffer.
*       STL 'CK.DVD READ PROCESSOR'
*       EJECT
```

APPENDIX 13-B:
HDOS Device Drivers' Programming Guide (Cont)

```
***     CK.DVD  READ PROCESSOR
*
*       ENTRY:  (BC)    = BYTES REQUESTED; MUST BE AT LEAST NINE
*               (DE)    = BUFFER ADDRESS; WHERE TIME STRING GETS PLACED
*
*       EXIT:   (PSW)   = CARRY SET; EOF ERROR IF GOOD READ
*                         ELSE; EXIT THROUGH "ILLEGAL"
*               (BC)    = UNUSED BYTE COUNT (NORMALLY ZERO)
*               (DE)    = ADDRESS OF NEXT BYTE TO BE READ (IF ANY)
*
*       USES:   ALL
*
*
*
READ    EQU     *           COME HERE WHEN CALLER WANTS TO READ TIME OF DAY

        LDA     RSTAT       FIRST, SEE IF WE'RE OPEN FOR READS
        ORA     A
        MVI     A,EC.FNO    PREPARE OURSELVES FOR "CHANNEL NOT OPEN" ERROR
        STC
        RZ                  RETURN IF CHANNEL NOT PREVIOUSLY OPENED
        MOV     A,C         CHECK FOR REQUEST OF AT LEAST NINE BYTES
        CPI     9
        JNC     READ1       IF (C) GREATER THAN OR EQUAL TO NINE
        MOV     A,B         SEE IF A MULTIPLE OF 256 BYTES REQUESTED
        ORA     A
        JZ      ILLEGAL     CAN'T SUPPLY LESS THAN NINE BYTES !
READ1   PUSH    B           SAVE BYTE COUNTER
        LXI     B,9         FORCE DEFAULT MOVE OF NINE BYTES
        LXI     H,TIMEBUF   WHERE TIME STRING IS COMING FROM
        DI                  IN CASE A "TICCNT" BOTCHES THINGS UP !
        CALL    MOVE        GIVE THE STRING TO THE CALLER OF THE DVD
        EI                  TURN ALL INTERRUPTS BACK ON
        POP     B           RESTORE BYTE COUNTER
        LXI     H,-9        ACCOUNT FOR BYTES ALREADY SENT TO CALLER
        DAD     B           (HL) EQUAL TO NUMBER OF BYTES TO PAD OUT
        MOV     B,H         PUT UNUSED BYTE COUNT IN (BC)
        MOV     C,L
        MVI     L,9         ACCOUNT FOR BYTES ALREADY SENT TO CALLER
READ2   MOV     A,B         SEE IF DONE PADDING OUT THE BUFFER
        ORA     C           (BC)=ZERO IF DONE
        JZ      READ3       FINISHED, SO EXIT
        XRA     A           ZERO (A)
        STAX    D           WRITE A NULL INTO THE BUFFER
        DCX     B           DECREMENT BYTE COUNTER
        INX     D           BUMP BUFFER POINTER
        INR     L           BUMP THE MODULO 256 BYTE COUNTER
        JZ      READ3       EXIT IF 256 BYTES HAVE BEEN SENT TO CALLER
        JMP     READ2       ELSE; EXIT IF (BC) REACHES ZERO FIRST
REBD3   MVI     A,EC.EOF    RETURN WITH "END OF FILE" ERROR
        STC                 TELL HDOS WE HAVE AN "ERROR"
        RET
```

APPENDIX 13-B:
HDOS Device Drivers' Programming Guide (Cont)

```
*       PIP will demand thousands of bytes, but we'll just give it 256 maximum!
        STL     'CK.DVD TICCNT ENTRY POINT'
        EJECT
***     CK.DVD  TICCNT ENTRY POINT
*
*       ENTRY:  EVERY TICCNT (2 MILLISECOND INTERVALS)
*
*       EXIT:   TO NORMAL HDOS TICCNT PROCESSOR
*               TIME IN "TIMEBUF" UPDATED IF A NEW SECOND
*
*       USES:   ALL; HDOS SAVES IT'S OWN REGISTERS
*
*
*
CLOCK   EQU     *           COME HERE EVERY "TICCNT" AND UPDATE THE TIME
        LHLD    TICKS       RETRIEVE OUR OWN PRIVATE "TICKER"
        INX     H           ADD ONE MORE TICK
        SHLD    TICKS       UPDATE OUR PRIVATE "TICKER"
        LXI     B,CAL       GET CALIBRATION FACTOR
        DAD     B           WILL CREATE A (CY) IF "TICKS"=500 DECIMAL
        JNC     CLKRET      NOT TIME FOR A NEW SECOND, SO RETURN
        SHLD    TICKS       (HL) WAS ZERO, RESET PRIVATE "TICKER"
        MVI     C,'0'       PUT AN ASCII ZERO IN REGISTER (C)

INRS    LXI     H,TIMEBUF+7 POINT AT UNITS SECONDS
        INR     M           BUMP IT
        MOV     A,M
        CPI     '9'+1       OVERFLOW ?
        JM      CLKRET      NO, SO RETURN
        MOV     M,C         RESET UNITS SECONDS
INRTS   DCX     H           POINT AT TENS SECONDS
        INR     M           BUMP IT
        MOV     A,M
        CPI     '6'         OVERFLOW ?
        JM      CLKRET      NO, SO RETURN
        MOV     M,C         RESET TENS SECONDS
INRM    DCX     H           POINT AT THE COLON
        DCX     H           POINT AT UNITS MINUTES
        INR     M           BUMP IT
        MOV     A,M
        CPI     '9'+1       OVERFLOW ?
        JM      CLKRET      NO, SO RETURN
        MOV     M,C         RESET UNITS MINUTES
INRTM   DCX     H           POINT AT TENS MINUTES
        INR     M           BUMP IT
        MOV     A,M
        CPI     '6'         OVERFLOW ?
        JM      CLKRET      NO, SO RETURN
        MOV     M,C         RESET TENS MINUTES
INRH    DCX     H           POINT AT THE COLON
        DCX     H           POINT AT UNITS HOURS
```

APPENDIX 13-B:
HDOS Device Drivers' Programming Guide (Cont)

```
        INR     M           BUMP IT
        MOV     A,M
        CPI     '4'         OVERFLOW ?
        JM      CLKRET      NO, SO RETURN
INRH1   DCX     H           POINT AT TENS HOURS
        MOV     A,M
        CPI     '2'         IS IT 24 AND NOT 14 OR 04 ?
        JM      INRH2       STILL MORE TO CHECK
        MOV     M,C         RESET TENS HOURS
        INX     H           POINT AT UNITS HOURS
        MOV     M,C         RESET UNITS HOURS
        JMP     CLKRET      (IT WAS MIDNIGHT)
INRH2   INX     H           POINT AT UNITS HOURS
        MOV     A,M
        CPI     '9'+1       OVERFLOW ?
        JM      CLKRET      NO, SO RETURN
        MOV     M,C         RESET UNITS HOURS
INRTH   DCX     H           POINT AT TENS HOURS
        INR     M           BUMP IT
        SPACE   3,9
CLKRET  EQU     *           NOW CONTINUE ON WITH HDOS CLOCK INT. ROUTINE
        JMP     0           NEW ADDRESS INSTALLED AT "LOAD" TIME
        SPACE   3,9
***     CK.DVD  STORAGE AREAS
*
*
TICKS   DW      0           THIS IS OUR PRIVATE "TICK COUNTER"
TIMEBUF DB      '00:00:00'  CORRECT TIME OF DAY MAINTAINED HERE, IN ASCII
BUFEND  DB      NL          TIME STRING ALWAYS TERMINATED WITH A "NEWLINE"
RSTAT   DB      0           HOLD PLACE FOR "OPEN FOR READ" STATUS FLAG
WSTAT   DB      0           HOLD PLACE FOR "OPEN FOR WRITE" STATUS FLAG
XFERCNT DB      0           HOLD PLACE FOR COUNT OF BYTES TRANSFERED
BUFPTR  DW      0           HOLD PLACE FOR POINTER INTO "TEMPBUF"
TEMPBUF DB      '00:00:00'  TEMPORARY BUFFER DURING "WRITE" PROCESSING
        SPACE   3,9
*       Another note for the astute programmer:
*
*       This  driver,  as  does  all Heath drivers except the disk drivers,
*       will allow you to read or write  any  number  of  bytes  you  want.
*       Contrary  to  what  the  System  Programmer's Guide says,  I/O to
*       non-storage  devices  need  not  be  in  multiples  of  256  bytes.
*       Realizing  this fact makes it easier to get data one character at a
*       time from a device  or  into  a  device.   Programs  will  be  more
*       efficient,  memory wise, if they do not have to maintain a 256 byte
*       buffer just to handle small I/O tasks.  Be aware that this may  not
*       be the case with drivers not coming from Heath Company.
*       STL     'BENTON HARBOR BASIC EXAMPLES'
*       EJECT
```

APPENDIX 13-B:
HDOS Device Drivers' Programming Guide (Cont)

```
*       00010 REM       READ CLOCK FROM BENTON HARBOR BASIC
*       00020 REM       11-JUN-81   DALE LAMM
*       00030 OPEN "CK:" FOR READ AS FILE #1
*       00040 INPUT #1,;T$
*       00050 PRINT T$
*       00060 CLOSE #1
*       00070 GOTO 10
*
*       The  above  merely  opens  a channel on the clock driver for reads,
*       then reads the current time-of-day, and then  closes  the  channel.
*       The program repeats until the control-C key is struck.
        SPACE   6
*       00010 REM       WRITE NEW TIME TO CLOCK FROM BENTON HARBOR BASIC
*       00020 REM       11-JUN-81   DALE LAMM
*       00030 OPEN "CK:" FOR WRITE AS FILE #1
*       00040 INPUT "WHAT TIME IS IT NOW ?  ";T$
*       00050 PRINT #1,T$
*       00060 CLOSE #1
*       00070 OPEN "CK:" FOR READ AS FILE #1
*       00080 INPUT #1,;T$
*       00090 PRINT "VERIFYING... CLOCK NOW READS "T$
*       00100 CLOSE #1
*       00110 END
*
*       The  example  above  demonstrates  how a new time-of-day may be put
*       into the clocks buffer.  The clock remains running, and keeps  time
*       using the just installed time string as the base.
*
*       Note  that  no  error  checking is done by the actual clock driver.
*       Whatever characters are inserted into the clocks buffer will be the
*       new base for timekeeping.  Likewise, you may use whatever character
*       suits your fancy to delimit the hours, minutes, and seconds.
*
*       The newly  entered  time-of-day  is  read  back  to  the  user  for
*       verification.   Only  the  first  eight characters in T$ are loaded
*       into the  drivers  buffer.   There  must  be  at  least  eight  new
*       characters  written into the buffer, else, the new time-of-day will
*       be meaningless.  Since Basic pads out writes to a file with  nulls,
*       it is not possible to use Basic to update only the first two digits
*       in the clock drivers buffer.
*       STL     'PIC TABLE'
        EJECT
        LON     G               TURN ON THE PIC TABLE LISTER
        END
```

****

APPENDIX 13-B:
HDOS Device Drivers' Programming Guide (Cont)

```
        TITLE   'Special Null Device Driver'
        STL     'Version 1.0  18-Sep-81'
***     NULL DEVICE DRIVER
*
*       by: Dale Lamm
*
*       The  purpose  of  this  program  is  to  aid  the assembly language
*       programmer  who  is  having  a  hard  time  making   his   programs
*       communicate  with  device drivers, and to provide some clues to the
*       file operations of HDOS.
*
*
*       After  assembly,  put it on a bootable disk and re-boot.  Then, try
*       the following:
*
*       PIP ND:=TT:             <CTL-D to get out>
*       PIP TT:=ND:
*       PIP ND:=*.*/S
*       PIP LP:=ND:
*       PIP ND:=ND:
*       LOAD ND:
*       MOUNT ND:               <will show an error>
*
*
*       It is interesting to watch when PIP requests gobs of bytes from ND:
*       and to watch BH Basic do file I/O on ND:.
*
*       Note that because of the way in which console output  is  performed
*       by  ND.DVD  that you must have either an H-8-4 card or an H-89.  In
*       both cases, the console must be on port 350Q.  Also,  all  register
*       values are displayed in hexadecimal.
*
*       If  you  have  a program which communicates with device drivers and
*       cannot seem to get it to work, the ND.DVD will verify that you  are
*       passing  the  right  numbers back and forth and are using the right
*       SYSCALLS.
*
*       STL     'External Definitions and Header'
*       EJECT
USERFWA EQU     2280H

        XTEXT   DDDEF
        XTEXT   PICDEF
        XTEXT   DEVDEF
        XTEXT   DVDDEF
        XTEXT   SETCAL
        XTEXT   U8250
```

APPENDIX 13-B:
HDOS Device Drivers' Programming Guide (Cont)

```
NL      EQU     0AH             SOME ASCII CHARACTERS...
CR      EQU     0DH
LF      EQU     0AH
BELL    EQU     07H

$TYPTX  EQU     195EH           SOME ROM ROUTINES...
$TBRA   EQU     193EH

EC.EOF  EQU     1               ERROR CODE; END OF FILE
EC.DNS  EQU     5               ERROR CODE; DEVICE NOT SUITABLE

        CODE    PIC

        DB      DVDFLV          DEVICE DRIVER FLAG VALUE
        DB      DT.CW+DT.CR     DEVICE CAPABILITY; READ AND WRITE
        DB      00000001B       MOUNTED UNIT MASK
        DB      1               ONLY 1 UNIT
        DB      DT.CW+DT.CR     UNIT ZERO; CAPABLE OF READ AND WRITE
        DS      7               UNITS 1-7;  IGNORED
        DB      DVDFLV          SET OPTIONS ARE AVAILABLE
        DW      0               NO INIT CODE

        CODE    -REL
        DS      DVD.STE-*
        CODE    +REL
        STL     'SET Entry Point'
        EJECT
***     SET CODE ENTRY POINT
*
*
*       Only SET option is 'HELP'
*
*
SETNTR  EQU     *
        ERRNZ   *-DVD.STE

        CALL    $TYPTX

        DB      NL,NL,NL
        DB      'This is nothing more than a special NULL DEVICE',NL
        DB      'driver for the express purpose of exploring PIP',NL
        DB      'and other programs which communicate with DVDs.',NL,NL
        DB      'To see it work, just try reading or writing to',NL
        DB      'it. Every time any DVD function is encountered',NL
        DB      'by this special driver, it will type on the screen',NL
        DB      'an explanation of what is going on.',NL,NL,NL,NL+80H

        XRA     A               CLEAR CARRY
        RET
```

APPENDIX 13-B:
HDOS Device Drivers' Programming Guide (Cont)

```
        CODE    -REL
        DS      DVD.ENT-*
        CODE    +REL
        STL     'Main-Line Code'
        EJECT
***     NULL DVD ENTRY POINT
*
*       ENTRY   (A) = PROCESS CODE
*               (BC) = BYTE COUNT (USUALLY)
*               (DE) = MEMORY ADDRESS (USUALLY)
*       EXIT    'C' CLEAR IF OK
*               'C' SET IF ERROR
*                (A) = ERROR CODE
*
*
ND.DVD  EQU     *               ENTRY POINT
        ERRNZ   *-DVD.ENT
        SHLD    HLSAV           SAVE (HL) FOR 'REGDIS'
        PUSH    H               SAVE (HL)
        CPI     DC.MAX          FUNCTION LEGAL ?
        JNC     NSUIT           NOPE; TREAT AS 'DEVICE NOT SUITABLE'
        CALL    $TBRA           ENTER PROCESSOR VIA 'TABLE BRANCH'
        DB      NREAD-*         READ
        DB      NWRITE-*        WRITE
        DB      NREADR-*        READR
        DB      NOPNR-*         OPENR
        DB      NOPNW-*         OPENW
        DB      NOPNU-*         OPENU
        DB      NCLOS-*         CLOSE
        DB      NABRT-*         ABORT
        DB      NMOUN-*         MOUNT
        DB      NLOAD-*         LOAD
        DB      NRDY-*          READY
        SPACE   3,9
***     NREAD - READ FROM NULL DEVICE
*
*
NREAD   EQU     *               READ FROM NULL DEVICE
        LXI     H,MESG1         POINT TO TEXT

NREAD1  CALL    TYPMES          TYPE IT
        CALL    REGDIS          DISPLAY ALL REGISTERS
        XRA     A               MAKE A 'NULL'
        MOV     L,A             INITIALIZE 'BYTE COUNTER'

NREAD2  STAX    D               WRITE THE 'NULL' TO WHOMEVER
        INX     D               NEXT BUFFER POSITION
        DCX     B               DECREMENT 'BYTE COUNTER'
        DCR     L               256 NULLS ?
        JNZ     NREAD2          NOPE; LOOP UNTIL DONE
```

APPENDIX 13-B:
HDOS Device Drivers' Programming Guide (Cont)

```
        POP     H               RESTORE (HL)
        MVI     A,EC.EOF        'END OF FILE ERROR'
        STC                     SAY 'ERROR'
        RET
        SPACE   3,9
***     NWRITE - WRITE TO NULL DEVICE
*
*
NWRITE  EQU     *               WRITE TO NULL DEVICE
        LXI     H,MESG2         POINT TO TEXT
        ERRNZ   EXIT-*          FALL THROUGH TO 'EXIT'; RETURN
        SPACE   3,9

***     EXIT - GENERAL EXIT FROM FUNCTION PROCESSORS
*
*
EXIT    EQU     *               GENERAL EXIT FROM PROCESSORS
        CALL    TYPMES          TYPE MESSAGE POINTED BY (HL)
        CALL    REGDIS          DISPLAY ALL REGISTERS
        POP     H               RESTORE (HL)
        XRA     A               CLEAR (CY)
        MOV     B,A             CLEAR 'BYTE COUNT'
        MOV     C,A
        RET
        SPACE   3,9
***     NREADR - READ REGARDLESS FROM NULL DEVICE
*
*
NREADR  EQU     *               READ REGARDLESS FROM NULL DEVICE
        LXI     H,MESG3         POINT TO TEXT
        JMP     NREAD1          LET THE 'READ' PROCESSOR FINISH UP
        SPACE   3,9
***     NOPNR - OPEN FOR READS FROM NULL DEVICE
*
*
NOPNR   EQU     *               OPEN FOR READS FROM NULL DEVICE
        LXI     H,MESG4         POINT TO TEXT
        JMP     EXIT            LET 'EXIT' FINISH UP
        SPACE   3,9
***     NOPNW - OPEN FOR WRITES TO NULL DEVICE
*
*
NOPNW   EQU     *               OPEN FOR WRITES TO NULL DEVICE
        LXI     H,MESG5         POINT TO TEXT
        JMP     EXIT            LET 'EXIT' FINISH UP
        SPACE   3,9
***     NOPNU - OPEN FOR UPDATES TO NULL DEVICE
*
*
```

APPENDIX 13-B:
HDOS Device Drivers' Programming Guide (Cont)

```
NOPNU   EQU     *               OPEN FOR UPDATES TO NULL DEVICE
        LXI     H,MESG6         POINT TO TEXT
        JMP     EXIT            LET 'EXIT' FINISH UP
        SPACE   3,9
***     NCLOS - CLOSE FILES ON NULL DEVICE
*
*
NCLOS   EQU     *               CLOSE FILES ON NULL DEVICE
        LXI     H,MESG7         POINT TO TEXT
        JMP     EXIT            LET 'EXIT' FINISH UP
        SPACE   3,9
***     NABRT - ABORT OPERATIONS ON NULL DEVICE
*
*
NABRT   EQU     *               ABORT OPERATIONS ON NULL DEVICE
        LXI     H,MESG8         POINT TO TEXT
        JMP     EXIT            LET 'EXIT' FINISH UP
        SPACE   3,9
***     NMOUN - MOUNT NULL DEVICE
*
*
NMOUN   EQU     *               MOUNT NULL DEVICE
        LXI     H,MESG9         POINT TO TEXT
        JMP     EXIT            LET 'EXIT' FINISH UP
        SPACE   3,9
***     NLOAD - LOAD NULL DEVICE DRIVER INTO MEMORY
*
*
NLOAD   EQU     *               LOAD NULL DEVICE DRIVER
        LXI     H,MESG10        POINT TO TEXT
        JMP     EXIT            LET 'EXIT' FINISH UP
        SPACE   3,9
***     NRDY - EXAMINE READY STATUS OF NULL DEVICE
*
*
NRDY    EQU     *               EXAMINE READY STATUS OF NULL DEVICE

        LXI     H,MESG11        POINT TO TEXT
        JMP     EXIT            LET 'EXIT' FINISH UP
        SPACE   3,9
***     NSUIT - UNSUITABLE FUNCTION REQUEST RECEIVED
*
*
NSUIT   EQU     *               UNSUITABLE FUNCTION REQUESTED
        STA     PSWSAV          SAVE ILLEGAL FUNCTION CODE
        LXI     H,MESG12        POINT TO TEXT
        CALL    TYPMES          TYPE IT
        LDA     PSWSAV          RESTORE ILLEGAL FUNCTION CODE
        CALL    NUMOUT          PRINT THE ILLEGAL FUNCTION CODE
        LXI     H,MESG13        DO A CR,LF
        CALL    TYPMES
```

APPENDIX 13-B:
HDOS Device Drivers' Programming Guide (Cont)

```
        POP     H               RESTORE (HL)
        MVI     A,EC.DNS        'DEVICE NOT SUITABLE ERROR'
        STC                     SAY 'ERROR'
        RET
        SPACE   3,9
        STL     'Subroutines'
        EJECT
***     TYPMES - TYPE MESSAGE POINTED BY (HL) UNTIL NULL; THEN RETURN
*
*
TYPMES  EQU     *               TYPE MESSAGE POINTED BY (HL)
        MOV     A,M             GET CHARACTER
        ANI     01111111B       STRIP PARITY; RAISE FLAGS
        RZ                      ALL DONE
        CALL    TCH             TYPE CHARACTER
        INX     H               NEXT CHARACTER
        JMP     TYPMES
        SPACE   3,9
***     REGDIS - FORMATTED DUMP OF (BC), (DE), AND (HL) REGISTERS
*
*
REGDIS  EQU     *               DISPLAY PRIMARY REGISTERS
        LXI     H,MESG14        PRINT (BC)
        CALL    TYPMES
        MOV     A,B             GET (B)
        CALL    NUMOUT          PRINT HEX VALUE
        MOV     A,C             GET (C)
        CALL    NUMOUT          PRINT HEX VALUE

        LXI     H,MESG15        PRINT (DE)
        CALL    TYPMES
        MOV     A,D             GET (D)
        CALL    NUMOUT          PRINT HEX VALUE
        MOV     A,E             GET (E)
        CALL    NUMOUT          PRINT HEX VALUE

        LXI     H,MESG16        PRINT (HL)
        CALL    TYPMES
        LHLD    HLSAV           GET ORIGINAL (HL)
        MOV     A,H             GET (H)
        CALL    NUMOUT          PRINT HEX VALUE
        MOV     A,L             GET (L)
        CALL    NUMOUT          PRINT HEX VALUE

        LXI     H,MESG13        DO A CR,LF
        JMP     TYPMES          PRINT; THEN RETURN
        SPACE   3,9
***     NUMOUT - PRINT (A) AS ASCII HEXADECIMAL
*
*
```

APPENDIX 13-B:
HDOS Device Drivers' Programming Guide (Cont)

```
NUMOUT  EQU     *               PRINT (PSW) AS ASCII HEXADECIMAL
        PUSH    PSW             SAVE BYTE VALUE
        RAR                     ROTATE UPPER INTO LOWER NIBBLE
        RAR
        RAR
        RAR
        CALL    NUM1            PRINT THE LOWER NIBBLE
        POP     PSW             RESTORE BYTE VALUE
        ERRNZ   NUM1-*          FALL THROUGH TO 'NUM1'; RETURN

NUM1    ANI     00001111B       MASK ONLY LOWER NIBBLE
        ADI     90H
        DAA
        ACI     40H
        DAA
        ERRNZ   TCH-*           FALL THROUGH TO 'TCH'; RETURN
        SPACE   3,9
***     TCH - TYPE CHARACTER TO H-8-4 OR H-89 CONSOLE UART
*
*
TCH     EQU     *               TYPE CHARACTER TO UART
        PUSH    PSW             SAVE CHARACTER

TCH1    IN      SC.ACE+UR.LSR   LOOK AT LINE STATUS
        ANI     UC.THE          XMTR HOLDING REGISTER EMPTY ?
        JZ      TCH1            NOPE; LOOP UNTIL EMPTY

        POP     PSW             RESTORE CHARACTER
        OUT     SC.ACE+UR.THR   SEND TO UART
        RET
        SPACE   3,9
***     DATA AREAS
*
*
HLSAV   DW      0               SAVE FOR (HL)
PSWSAV  DB      0               SAVE FOR (PSW)

MESG1   DB      CR,LF,BELL,'Reading from ND: device:',CR,LF,0
MESG2   DB      CR,LF,BELL,'Writing to ND: device:',CR,LF,0
MESG3   DB      CR,LF,BELL,'Reading regardless from ND: device',CR,LF,0
MESG4   DB      CR,LF,BELL,'Opening ND: for reads:',CR,LF,0
MESG5   DB      CR,LF,BELL,'Opening ND: for writes:',CR,LF,0
MESG6   DB      CR,LF,BELL,'Opening ND: for updates:',CR,LF,0
MESG7   DB      CR,LF,BELL,'Closing files on ND: device:',CR,LF,0
MESG8   DB      CR,LF,BELL,'Aborting ND: device:',CR,LF,0
MESG9   DB      CR,LF,BELL,'Mounting ND: device:',CR,LF,0
MESG10  DB      CR,LF,BELL,'Loading ND: device driver:',CR,LF,0
MESG11  DB      CR,LF,BELL,'Checking ND: ready status:',CR,LF,0
MESG12  DB      CR,LF,BELL,'Illegal ND: function request: (A)= ',0
```

APPENDIX 13-B:
HDOS Device Drivers' Programming Guide (Cont)

```
MESG13  DB      CR,LF,0
MESG14  DB      '(BC) = ',0
MESG15  DB      '   (DE) = ',0
MESG16  DB      '   (HL) = ',0
        END

*****************************************************************************
```

APPENDIX 13-C: CONVERSION CHART

DEC	OCT	HEX	BINARY	ASCII	8080 OPCODE	
0	0	0	00000000	^@ NUL	NOP	
1	1	1	00000001	^A SOH	LXI	B,ADDRESS
2	2	2	00000010	^B STX	STAX	B
3	3	3	00000011	^C ETX	INX	B
4	4	4	00000100	^D EOT	INR	B
5	5	5	00000101	^E ENQ	DCR	B
6	6	6	00000110	^F ACK	MVI	B,BYTE
7	7	7	00000111	^G BEL	RLC	
8	10	8	00001000	^H BS	****	NOT USED BY 8080
9	11	9	00001001	^I HT	DAD	B
10	12	A	00001010	^J LF	LDAX	B
11	13	B	00001011	^K VT	DCX	B
12	14	C	00001100	^L FF	INR	C
13	15	D	00001101	^M CR	DCR	C
14	16	E	00001110	^N SO	MVI	C,BYTE
15	17	F	00001111	^O SI	RRC	
16	20	10	00010000	^P DLE	****	NOT USED BY 8080
17	21	11	00010001	^Q DC1	LXI	D,ADDRESS
18	22	12	00010010	^R DC2	STAX	D
19	23	13	00010011	^S DC3	INX	D
20	24	14	00010100	^T DC4	INR	D
21	25	15	00010101	^U NAK	DCR	D
22	26	16	00010110	^V SYN	MVI	D,BYTE
23	27	17	00010111	^W ETB	RAL	
24	30	18	00011000	^X CAN	****	NOT USED BY 8080
25	31	19	00011001	^Y EM	DAD	D
26	32	1A	00011010	^Z SUB	LDAX	D
27	33	1B	00011011	^[ESC	DCX	D
28	34	1C	00011100	^\ FS	INR	E
29	35	1D	00011101	^] GS	DCR	E
30	36	1E	00011110	^^ RS	MVI	E,BYTE
31	37	1F	00011111	^_ US	RAR	
32	40	20	00100000	SPACE	****	NOT USED BY 8080
33	41	21	00100001	!	LXI	H,ADDRESS
34	42	22	00100010	"	SHLD	ADDRESS
35	43	23	00100011	#	INX	H

APPENDIX 13-C: CONVERSION CHART (Cont)

DEC	OCT	HEX	BINARY	ASCII	8080 OPCODE	
36	44	24	00100100	$	INR	H
37	45	25	00100101	%	DCR	H
38	46	26	00100110	&	MVI	H,BYTE
39	47	27	00100111	' TICK	DAA	
40	50	28	00101000	(	****	NOT USED BY 8080
41	51	29	00101001	)	DAD	H
42	52	2A	00101010	*	LHLD	ADDRESS
43	53	2B	00101011	+	DCX	H
44	54	2C	00101100	, COMMA	INR	L
45	55	2D	00101101	- HYPHN	DCR	L
46	56	2E	00101110	.	MVI	L,BYTE
47	57	2F	00101111	/	CMA	
48	60	30	00110000	0	****	NOT USED BY 8080
49	61	31	00110001	1	LXI	SP,ADDRESS
50	62	32	00110010	2	STA	ADDRESS
51	63	33	00110011	3	INX	SP
52	64	34	00110100	4	INR	M
53	65	35	00110101	5	DCR	M
54	66	36	00110110	6	MVI	M,BYTE
55	67	37	00110111	7	STC	
56	70	38	00111000	8	****	NOT USED BY 8080
57	71	39	00111001	9	DAD	SP
58	72	3A	00111010	:	LDA	ADDRESS
59	73	3B	00111011	;	DCX	SP
60	74	3C	00111100	<	INR	A
61	75	3D	00111101	=	DCR	A
62	76	3E	00111110	>	MVI	A,BYTE
63	77	3F	00111111	?	CMC	
64	100	40	01000000	@	MOV	B,B
65	101	41	01000001	A	MOV	B,C
66	102	42	01000010	B	MOV	B,D
67	103	43	01000011	C	MOV	B,E
68	104	44	01000100	D	MOV	B,H
69	105	45	01000101	E	MOV	B,L
70	106	46	01000110	F	MOV	B,M
71	107	47	01000111	G	MOV	B,A

APPENDIX 13-C: CONVERSION CHART (Cont)

DEC	OCT	HEX	BINARY	ASCII	8080 OPCODE
72	110	48	01001000	H	MOV C,B
73	111	49	01001001	I	MOV C,C
74	112	4A	01001010	J	MOV C,D
75	113	4B	01001011	K	MOV C,E
76	114	4C	01001100	L	MOV C,H
77	115	4D	01001101	M	MOV C,L
78	116	4E	01001110	N	MOV C,M
79	117	4F	01001111	O	MOV C,A
80	120	50	01010000	P	MOV D,B
81	121	51	01010001	Q	MOV D,C
82	122	52	01010010	R	MOV D,D
83	123	53	01010011	S	MOV D,E
84	124	54	01010100	T	MOV D,H
85	125	55	01010101	U	MOV D,L
86	126	56	01010110	V	MOV D,M
87	127	57	01010111	W	MOV D,A
88	130	58	01011000	X	MOV E,B
89	131	59	01011001	Y	MOV E,C
90	132	5A	01011010	Z	MOV E,D
91	133	5B	01011011	[	MOV E,E
92	134	5C	01011100	\	MOV E,H
93	135	5D	01011101	]	MOV E,L
94	136	5E	01011110	^	MOV E,M
95	137	5F	01011111	_ U.L.	MOV E,A
96	140	60	01100000	`	MOV H,B
97	141	61	01100001	a	MOV H,C
98	142	62	01100010	b	MOV H,D
99	143	63	01100011	c	MOV H,E
100	144	64	01100100	d	MOV H,H
101	145	65	01100101	e	MOV H,L
102	146	66	01100110	f	MOV H,M
103	147	67	01100111	g	MOV H,A
104	150	68	01101000	h	MOV L,B
105	151	69	01101001	i	MOV L,C
106	152	6A	01101010	j	MOV L,D
107	153	6B	01101011	k	MOV L,E

APPENDIX 13-C: CONVERSION CHART (Cont)

DEC	OCT	HEX	BINARY	ASCII	8080 OPCODE
108	154	6C	01101100	l	MOV L,H
109	155	6D	01101101	m	MOV L,L
110	156	6E	01101110	n	MOV L,M
111	157	6F	01101111	o	MOV L,A
112	160	70	01110000	p	MOV M,B
113	161	71	01110001	q	MOV M,C
114	162	72	01110010	r	MOV M,D
115	163	73	01110011	s	MOV M,E
116	164	74	01110100	t	MOV M,H
117	165	75	01110101	u	MOV M,L
118	166	76	01110110	v	HLT
119	167	77	01110111	w	MOV M,A
120	170	78	01111000	x	MOV A,B
121	171	79	01111001	y	MOV A,C
122	172	7A	01111010	z	MOV A,D
123	173	7B	01111011	{	MOV A,E
124	174	7C	01111100	|	MOV A,H
125	175	7D	01111101	}	MOV A,L
126	176	7E	01111110	~	MOV A,M
127	177	7F	01111111	DEL	MOV A,A
128	200	80	10000000	[^@]	ADD B
129	201	81	10000001	[^A]	ADD C
130	202	82	10000010	[^B]	ADD D
131	203	83	10000011	[^C]	ADD E
132	204	84	10000100	[^D]	ADD H
133	205	85	10000101	[^E]	ADD L
134	206	86	10000110	[^F]	ADD M
135	207	87	10000111	[^G]	ADD A
136	210	88	10001000	[^H]	ADC B
137	211	89	10001001	[^I]	ADC C
138	212	8A	10001010	[^J]	ADC D
139	213	8B	10001011	[^K]	ADC E
140	214	8C	10001100	[^L]	ADC H
141	215	8D	10001101	[^M]	ADC L
142	216	8E	10001110	[^N]	ADC M
143	217	8F	10001111	[^O]	ADC A

APPENDIX 13-C: CONVERSION CHART (Cont)

DEC	OCT	HEX	BINARY	ASCII	8080 OPCODE	
144	220	90	10010000	[^P]	SUB	B
145	221	91	10010001	[^Q]	SUB	C
146	222	92	10010010	[^R]	SUB	D
147	223	93	10010011	[^S]	SUB	E
148	224	94	10010100	[^T]	SUB	H
149	225	95	10010101	[^U]	SUB	L
150	226	96	10010110	[^V]	SUB	M
151	227	97	10010111	[^W]	SUB	A
152	230	98	10011000	[^X]	SBB	B
153	231	99	10011001	[^Y]	SBB	C
154	232	9A	10011010	[^Z]	SBB	D
155	233	9B	10011011	[^[ ]	SBB	E
156	234	9C	10011100	[^\ ]	SBB	H
157	235	9D	10011101	[^]]	SBB	L
158	236	9E	10011110	[^^]	SBB	M
159	237	9F	10011111	[^_]	SBB	A
160	240	A0	10100000	[ ]	ANA	B
161	241	A1	10100001	[!]	ANA	C
162	242	A2	10100010	["]	ANA	D
163	243	A3	10100011	[#]	ANA	E
164	244	A4	10100100	[$]	ANA	H
165	245	A5	10100101	[%]	ANA	L
166	246	A6	10100110	[&]	ANA	M
167	247	A7	10100111	[']	ANA	A
168	250	A8	10101000	[( ]	XRA	B
169	251	A9	10101001	[ )]	XRA	C
170	252	AA	10101010	[*]	XRA	D
171	253	AB	10101011	[+]	XRA	E
172	254	AC	10101100	[,]	XRA	H
173	255	AD	10101101	[-]	XRA	L
174	256	AE	10101110	[.]	XRA	M
175	257	AF	10101111	[/]	XRA	A
176	260	B0	10110000	[0]	ORA	B
177	261	B1	10110001	[1]	ORA	C
178	262	B2	10110010	[2]	ORA	D
179	263	B3	10110011	[3]	ORA	E

APPENDIX 13-C: CONVERSION CHART (Cont)

DEC	OCT	HEX	BINARY	ASCII	8080 OPCODE	
180	264	B4	10110100	[4]	ORA	H
181	265	B5	10110101	[5]	ORA	L
182	266	B6	10110110	[6]	ORA	M
183	267	B7	10110111	[7]	ORA	A
184	270	B8	10111000	[8]	CMP	B
185	271	B9	10111001	[9]	CMP	C
186	272	BA	10111010	[:]	CMP	D
187	273	BB	10111011	[;]	CMP	E
188	274	BC	10111100	[<]	CMP	H
189	275	BD	10111101	[=]	CMP	L
190	276	BE	10111110	[>]	CMP	M
191	277	BF	10111111	[?]	CMP	A
192	300	C0	11000000	[@]	RNZ	
193	301	C1	11000001	[A]	POP	B
194	302	C2	11000010	[B]	JNZ	ADDRESS
195	303	C3	11000011	[C]	JMP	ADDRESS
196	304	C4	11000100	[D]	CNZ	ADDRESS
197	305	C5	11000101	[E]	PUSH	B
198	306	C6	11000110	[F]	ADI	BYTE
199	307	C7	11000111	[G]	RST	0
200	310	C8	11001000	[H]	RZ	
201	311	C9	11001001	[I]	RET	
202	312	CA	11001010	[J]	JZ	ADDRESS
203	313	CB	11001011	[K]	****	NOT USED BY 8080
204	314	CC	11001100	[L]	CZ	ADDRESS
205	315	CD	11001101	[M]	CALL	ADDRESS
206	316	CE	11001110	[N]	ACI	BYTE
207	317	CF	11001111	[O]	RST	1
208	320	D0	11010000	[P]	RNC	
209	321	D1	11010001	[Q]	POP	D
210	322	D2	11010010	[R]	JNC	ADDRESS
211	323	D3	11010011	[S]	OUT	BYTE
212	324	D4	11010100	[T]	CNC	ADDRESS
213	325	D5	11010101	[U]	PUSH	D
214	326	D6	11010110	[V]	SUI	BYTE
215	327	D7	11010111	[W]	RST	2

APPENDIX 13-C: CONVERSION CHART (Cont)

DEC	OCT	HEX	BINARY	ASCII	8080 OPCODE	
216	330	D8	11011000	[X]	RC	
217	331	D9	11011001	[Y]	****	NOT USED BY 8080
218	332	DA	11011010	[Z]	JC	ADDRESS
219	333	DB	11011011	[[ ]	IN	BYTE
220	334	DC	11011100	[\ ]	CC	ADDRESS
221	335	DD	11011101	[ ]]	****	NOT USED BY 8080
222	336	DE	11011110	[^]	SBI	BYTE
223	337	DF	11011111	[_]	RST	3
224	340	E0	11100000	[`]	RPO	
225	341	E1	11100001	[a]	POP	H
226	342	E2	11100010	[b]	JPO	ADDRESS
227	343	E3	11100011	[c]	XTHL	
228	344	E4	11100100	[d]	CPO	ADDRESS
229	345	E5	11100101	[e]	PUSH	H
230	346	E6	11100110	[f]	ANI	BYTE
231	347	E7	11100111	[g]	RST	4
232	350	E8	11101000	[h]	RPE	
233	351	E9	11101001	[i]	PCHL	
234	352	EA	11101010	[j]	JPE	ADDRESS
235	353	EB	11101011	[k]	XCHG	
236	354	EC	11101100	[l]	CPE	ADDRESS
237	355	ED	11101101	[m]	****	NOT USED BY 8080
238	356	EE	11101110	[n]	XRI	BYTE
239	357	EF	11101111	[o]	RST	5
240	360	F0	11110000	[p]	RP	
241	361	F1	11110001	[q]	POP	PSW
242	362	F2	11110010	[r]	JP	ADDRESS
243	363	F3	11110011	[s]	DI	
244	364	F4	11110100	[t]	CP	ADDRESS
245	365	F5	11110101	[u]	PUSH	PSW
246	366	F6	11110110	[v]	ORI	BYTE
247	367	F7	11110111	[w]	RST	6
248	370	F8	11111000	[x]	RM	
249	371	F9	11111001	[y]	SPHL	
250	372	FA	11111010	[z]	JM	ADDRESS
251	373	FB	11111011	[{]	EI	

APPENDIX 13-C: CONVERSION CHART (Cont)

DEC	OCT	HEX	BINARY	ASCII	8080 OPCODE	
252	374	FC	11111100	[|]	CM	ADDRESS
253	375	FD	11111101	[}]	****	NOT USED BY 8080
254	376	FE	11111110	[~]	CPI	BYTE
255	377	FF	11111111	[DEL]	SCALL	.SCIN ;ANY VALID SYSTEM CALL

APPENDIX 13-D: MEMORY

```
------------------------------------------------------------------------
               Heath Disk Operating System - Version 3.0
------------------------------------------------------------------------

000.060  0030   VERS    EQU     3*16+0  Version 3.0

------------------------------------------------------------------------
    **    HDOS BASE PAGE DEFINITION
------------------------------------------------------------------------

000.000  0000   HOSBASE EQU     0

000.000  0000   B.INT0  DS      3       System Interrupt

000.003  0003   S.LABEL DS      2       Label Buffer FWA

000.005  0005   S.FMASK DS      1       Feature Mask

000.003  0003   F.CLK   EQU     00000011B   System clock speed
000.000  0000   F.2MHZ  EQU     00000000B    2 MHz
000.001  0001   F.4MHZ  EQU     00000001B    4 MHz

000.014  000C   F.MACH  EQU     00001100B   System type
000.000  0000   F.H8    EQU     00000000B    H8
000.004  0004   F.H89   EQU     00000100B    H89
000.010  0008   F.Z100  EQU     00001000B    Z-100
000.014  000C   F.OMACH EQU     00001100B    PC, ETC.

000.060  0030   F.TERM  EQU     00110000B   Terminal type
000.000  0000   F.TTY   EQU     00000000B    Dumb TTY
000.020  0010   F.H19   EQU     00010000B    H19

000.300  00C0   F.CPU   EQU     11000000B   CPU type
000.000  0000   F.8080  EQU     00000000B    8080
000.100  0040   F.8085  EQU     01000000B    8085
000.200  0080   F.Z80   EQU     10000000B    Z-80
000.300  00C0   F.OCPU  EQU     11000000B    HD64180, V20, ETC.

000.006  0006   S.LWA   DS      2       First free byte following HDOS

000.010  0008   B.INT1  DS      3       Clock Interrupt

000.013  000B   S.REV   DS      1       HDOS Revision #
000.014  000C           DS      2       HDOS Assembly Date
000.016  000E           DS      2       HDOS Assembly Time

000.020  0010   B.INT2  DS      3       Available

000.023  0013   PRIDEV  DS      4       Primary device name
000.027  0017           DS      1
```

APPENDIX 13-D: MEMORY (Cont)

** HDOS BASE PAGE DEFINITION (Cont)

```
000.030  0018   B.INT3   DS      3       Available

000.033  001B   ALTDEV   DS      4       Alternate device name
000.037  001F           DS      1

000.040  0020   B.INT4   DS      3       Available

000.043  0023   LSTDEV   DS      4       List device name
000.047  0027           DS      1

000.050  0028   B.INT5   DS      3       Available

000.053  002B   S.DLY    DS      3       Jump to delay routine
000.053  002B   .DLY     EQU     *

000.056  002E   S.USER   DS      1       Active USER Area (ASCII)
000.057  002F   S.UMASK  DS      1       Active USER Mask (bit pattern)

000.060  0030   B.INT6   DS      3       Available

000.063  0033   S.TFWA   DS      2       Task Table FWA  \3.02\
000.065  0035   S.TFLG   DS      1       Bit flags for task resident
                                           status

000.066  0036   S.UNIT   DS      1       Holding place for line printer
                                           unit
000.067  0037   S.KEY$   DS      1       Holding place for ASK

000.070  0038   B.INT7   DS      3       SCALL Interrupt

000.073  003B   B.SCALL  DS      3       JMP directly to SCALL processor

000.076  003E   CSLIBUF  DS      2       Console Type-Ahead Buffer FWA

000.100  0040   BATNAME  DS      17      Complete Batch File Name
000.121  0051   BATSEC   DS      1       Current Sector Index
000.122  0052   BATGNS   DS      2       Batch File FGN & LGN

000.124  0054   BATBUF   DS      2       Batch Buffer FWA
000.126  0056   BATPTR   DS      2       Pointer into BATBUF

000.130  0058   SUBBUF   DS      2       Substitution Buffer FWA

000.132  005A   S.PATH   DS      2       System Path Buffer FWA

000.134  005C   S.PRMT   DS      2       System Prompt Buffer FWA
000.136  005E   S.EDLIN  DS      2       Line Editor Buffer FWA
```

APPENDIX 13-D: MEMORY (Cont)

```
    **    HDOS BASE PAGE DEFINITION (Cont)
```

```
000.140  0060   S.COUNT DS      1       Batch Counter Byte
000.141  0061   S.SHIFT DS      1       Batch Shift Count
000.142  0062   S.BITS  DS      1       Batch BIT Flags

000.143  0063           DS      1       Used by Pre-Load
000.144  0064           DS      1       Used by Pre-Load
000.145  0065   S.XFLAG DS      1       Aux. SYSCMD Control Flags

000.001  0001   S.HALT  EQU     00000001B   HALT Command has been given
000.002  0002   S.SPC   EQU     00000010B   Show PIP Command before
                                             executing
000.004  0004   *       EQU     00000100B
000.010  0008   *       EQU     00001000B
000.020  0010   *       EQU     00010000B
000.040  0020   *       EQU     00100000B
000.100  0040   S.DSF   EQU     01000000B   Permission to use /DSF in
                                             PIP
000.200  0080   S.ULTRA EQU     10000000B   Ultra ROM is present

000.146  0066   B.NMI   DS      3       NMI handler vector
000.151  0069   B.NMIFL DS      1       NMI flag (0=no NMI's occured)

000.152  006A   S.DFBLK DB      'SY0ABS'   Resident Default Block

000.160  0070   S.CVEC  DS      5*3     console SCALL vectors

000.160  0070           DS      3       Vector to .scin.
000.163  0073           DS      3       Vector to .scout.
000.166  0076           DS      3       Vector to .print.
000.171  0079           DS      3       Vector to .consl.
000.174  007C           DS      3       Vector to .clrco.

000.177  007F   S.FLAG  DS      1       SYSCMD Control Flags

000.001  0001   S.SYSCM EQU     00000001B   SYSCMD.SYS is in memory
000.002  0002   S.VFLG  EQU     00000010B   VERIFY mode is on
000.004  0004   S.ECHO  EQU     00000100B   ECHO mode is off
000.010  0008   S.BATCH EQU     00001000B   BATCH mode is on
000.020  0010   S.EXITC EQU     00010000B   Display exit code on
                                             re-entry
000.040  0020   S.BREAK EQU     00100000B   Break off current operation
000.100  0040   S.TABUF EQU     01000000B   Type-Ahead Buffer stuffed
                                             by user

000.200  0080   S.INIT  EQU     10000000B   SYSCMD INIT has been done

000.200  0080   B.END   EQU     *
```

APPENDIX 13-D: MEMORY (Cont)

```
------------------------------------------------------------------------
    **    HDOS 3.0 Operating System Resident Code
------------------------------------------------------------------------

000.200  0080   FWA OF HDOS CODE
::: :::  ::::
026.156  166E   LWA OF HDOS CODE

026.157  166F           DS      1

------------------------------------------------------------------------
  *     Super 89 Pre-Load Module Resident Code
------------------------------------------------------------------------

026.160  1670   FWA of Code
::: :::  ::::
027.377  17FF   LWA of Code

------------------------------------------------------------------------
    **    H17 ROM Code which is retained by HDOS 3.0
------------------------------------------------------------------------

030.000  1800           DS      3       Vector to S.FASER
030.003  1803           DS      45      Obsolete Code (Memory
                                           Diagnostic)

030.060  1830           DS      354     ROM Subroutines ($TYPTX,
                                           etc.)
031.222  1992           DS      62      H17 Driver ROM Code
031.320  19D0           DS      8       EIGHT CONSTANT ZEROS
031.330  19D8           DS      421     H17 Driver ROM Code
033.175  1B7D           DS      30      Relocation Code ($REL. & $REL)
033.233  1B9B           DS      101     H17 Driver ROM Code

------------------------------------------------------------------------
    **    HDOS 3.0 Buffers
------------------------------------------------------------------------

034.000  1C00           DS      256     SYSTEM LABEL BUFFER
035.000  1D00           DS      256     BATCH BUFFER
036.000  1E00           DS      101     SUBSTITUTION BUFFER
036.145  1E65           DS      101     PATH BUFFER
036.312  1ECA           DS      101     PROMPT BUFFER
037.057  1F2F           DS      101     COMMAND LINE EDITOR BUFFER
037.224  1F94           DS      101     TYPE-AHEAD BUFFER
037.371  1FF9           DS      7       Unused
040.000  2000   .START  EQU     40000A  START DUMP ADDRESS
040.002  2002   .IOWRK  EQU     40002A  IN OR OUT INSTRUCTION
040.005  2005   .REGI   EQU     40005A  DISPLAYED REGISTER INDEX
040.006  2006   .DSPROT EQU     40006A  PERIOD FLAG BYTE
040.007  2007   .DSPMOD EQU     40007A  DISPLAY MODE
```

APPENDIX 13-D: MEMORY (Cont)

```
------------------------------------------------------------------------
    **    RAM CELLS USED BY OLD MONITOR CODE & HDOS
------------------------------------------------------------------------

040.010  2008   .MFLAG  EQU     40010A  USER OPTION BYTE

------------------------------------------------------------------------
  *     USER OPTION BITS
  *     These bits are set in cell .MFLAG
------------------------------------------------------------------------

000.200  0080   UO.HLT  EQU     10000000B   DISABLE HALT PROCESSING
000.100  0040   UO.NFR  EQU     01000000B   NO REFRESH OF FRONT PANEL
000.002  0002   UO.DDU  EQU     00000010B   DISABLE DISPLAY UPDATE
000.001  0001   UO.CLK  EQU     00000001B   ALLOW PRIVATE INTERRUPT
                                             PROCESSING

040.011  2009   .CTLFLG EQU     40011A  PANEL CONTROL BYTE
040.013  200B   .ALEDS  EQU     40013A  ABUSS LEDS
040.021  2011   .DLEDS  EQU     40021A  DBUSS LEDS
040.024  2014   .ABUSS  EQU     40024A  ABUSS REGISTER
040.027  2017   .CRCSUM EQU     40027A  CRCSUM WORD
040.031  2019   .TPERRX EQU     40031A  TAPE ERROR EXIT VECTOR
040.033  201B   .TICCNT EQU     40033A  CLOCK TICK COUNTER
040.035  201D   .REGPTR EQU     40035A  REGISTER POINTER
040.037  201F   .UIVEC  EQU     40037A  USER INTERRUPT VECTORS

040.037  201F           DS      3  CLOCK        Int 1 = .UIVEC
040.042  2022           DS      3  SINGLE STEP  Int 2 = .UIVEC+3
040.045  2025           DS      3  KEYBOARD     Int 3 = .UIVEC+6
040.050  2028           DS      3  H37          Int 4 = .UIVEC+9
040.053  202B           DS      3  MODEM        Int 5 = .UIVEC+12
040.056  202E           DS      3  CLOCK89      Int 6 = .UIVEC+15
040.061  2031           DS      3  SCALL        Int 7 = .UIVEC+18

040.064  2034   .NMIRET DS      2       H88/H89 NMI Return Address
040.066  2036   .CTL2FL DS      1       OP2.CTL Control Byte

040.067  2037           DS      9       Reserved

------------------------------------------------------------------------
    **    HDOS SYSTEM RAM WORKSPACE
------------------------------------------------------------------------

040.100  2040           ORG     040100A   FREE SPACE FROM PAM-8

040.100  2040           DS      8       JUMP TO SYSTEM EXIT

040.100  2040           XRA     A
040.101  2041           STA     SYSMODE   162BH in HDOS 3.0a
```

APPENDIX 13-D: MEMORY (Cont)

```
------------------------------------------------------------------------
    **    HDOS SYSTEM RAM WORKSPACE (Cont)
------------------------------------------------------------------------

040.104  2044           MVI     A,1
040.106  2046           SCALL   .EXIT

------------------------------------------------------------------------
  *     D.CON - DISK CONSTANTS
------------------------------------------------------------------------

040.110  2048   D.CON    DS      16      DISK CONSTANTS

040.110  2048   D.XITA   DS      2       HEAD UNSETTLE & MOTOR ON TIMES
040.112  204A   D.WRITA  DS      1       GUARDBAND COUNT FOR WRITE
040.113  204B   D.WRITB  DS      1       NUMBER OF ZERO CHARS AFTER
                                           HOLE EDGE

040.114  204C   D.WRITC  DS      1       TWO CHAR DELAY BEFORE WRITING
040.115  204D   D.MAIA   DS      1       TRACK-TO-TRACK STEP TIMES
040.116  204E   D.LPSA   DS      1       NUMBER OF TRIES FOR CORRECT
                                           SECTOR
040.117  204F   D.SDPA   DS      1       70 MILLISECOND WAIT FOR HEAD
                                           SETTLE

040.120  2050   D.SDPB   DS      1       1 SECOND WAIT FOR MOTOR ON
040.121  2051   D.STSA   DS      1       MS/2 TO WAIT FOR INDEX HOLE
040.122  2052   D.STSB   DS      1       MS/2 TO WAIT PAST INDEX HOLE

040.123  2053   D.WHDA   DS      1       UDLY COUNT FOR HOLE DEBOUNCE
040.124  2054   D.WNHA   DS      1       UDLY COUNT FOR HOLE DEBOUNCE
040.125  2055   D.WSCA   DS      1       LOOP COUNT FOR 25 CHARS

040.126  2056   D.ERTS   DS      2       TRACK AND SECTOR OF LAST DISK
                                           ERRORS
------------------------------------------------------------------------
  *     SYSTEM DISK VECTORS
------------------------------------------------------------------------

040.130  2058   SYDD     EQU     *       SYSTEM DISK ENTRY POINT
040.130  2058   D.VEC    DS      24*3    SYSTEM ROM ENTRY VECTORS

040.130  2058   D.SYDD   DS      3       SYSTEM DISK DEVICE DRIVER
040.133  205B   D.MOUNT  DS      3       MOUNT NEW DEVICE
040.136  205E   D.XOK    DS      3       EXIT WITH ALL OK FLAG
040.141  2061   D.ABORT  DS      3       ABORT ANY ACTIVE I/O
040.144  2064   D.XIT    DS      3       EXIT
040.147  2067   D.READ   DS      3       READ FROM DISK
040.152  206A   D.READR  DS      3       READ REGARDLESS OF VOLUME
                                           PROTECTION
```

APPENDIX 13-D: MEMORY (Cont)

```
    *     SYSTEM DISK VECTORS (Cont)
```

040.155	206D	D.WRITE	DS	3	WRITE TO DISK
040.160	2070	D.CDE	DS	3	COUNT DISK ERRORS
040.163	2073	D.DTS	DS	3	DECODE TRACK & SECTOR
040.166	2076	D.SDT	DS	3	SEEK DESIRED TRACK
040.171	2079	D.MAI	DS	3	MOVE DISK ARM IN ONE TRACK
040.174	207C	D.MAO	DS	3	MOVE DISK ARM OUT ONE TRACK
040.177	207F	D.LPS	DS	3	LOCATE PROPER SECTOR
040.202	2082	D.RDB	DS	3	READ BYTE FROM DISK
040.205	2085	D.SDP	DS	3	SET DEVICE PARAMETERS
040.210	2088	D.STS	DS	3	SKIP THIS SECTOR
040.213	208B	D.STZ	DS	3	SEEK TRACK ZERO
040.216	208E	D.UDLY	DS	3	MICROSECOND DELAY
040.221	2091	D.WSC	DS	3	WAIT SYNC CHARACTER
040.224	2094	D.WSP	DS	3	WRITE SYNC PATTERN
040.227	2097	D.WNB	DS	3	WRITE NEXT BYTE
040.232	209A	D.ERRT	DS	3	ERROR TEST LOOP
040.235	209D	D.DLY	DS	3	DELAY BY FRONT PANEL CLOCK

```
    *     D.RAM - DISK RAM WORK AREA
    *     Zeroed upon booting up
```

040.240	20A0	D.RAM	DS	31	SYSTEM ROM WORK AREA
040.240	20A0	D.TT	DS	1	TARGET TRACK (CURRENT OPERATION)
040.241	20A1	D.TS	DS	1	TARGET SECTOR (CURRENT OPERATION)
040.242	20A2	D.DVCTL	DS	1	DEVICE CONTROL BYTE
040.243	20A3	D.DLYMO	DS	1	MOTOR ON DELAY COUNT
040.244	20A4	D.DLYHS	DS	1	HEAD SETTLE DELAY COUNTER
040.245	20A5	D.TRKPT	DS	2	ADDRESS IN D.DRVTB FOR TRACK NUMBER
040.247	20A7	D.VOLPT	DS	2	ADDRESS IN D.DRVTB FOR VOLUME NUMBER
040.251	20A9	D.DRVTB	DS	2*4	TRACK AND VOLUME NUMBER FOR 4 DRIVES
040.261	20B1	D.HECNT	DS	1	HARD ERROR COUNT
040.262	20B2	D.SECNT	DS	2	SOFT ERROR COUNT
040.264	20B4	D.OECNT	DS	1	OPERATION ERROR COUNT

APPENDIX 13-D: MEMORY (Cont)

```
  *     GLOBAL DISK ERROR COUNTERS

040.265  20B5   D.ERR   DS      0       *BEGINNING OF ERROR BLOCK
040.265  20B5   D.E.MDS DS      1       *MISSING DATA SYNC
040.266  20B6   D.E.HSY DS      1       *MISSING HEADER SYNC
040.267  20B7   D.E.CHK DS      1       *DATA CHECKSUM
040.270  20B8   D.E.HCK DS      1       *HEADER CHECKSUM
040.271  20B9   D.E.VOL DS      1       *WRONG VOLUME NUMBER
040.272  20BA   D.E.TRK DS      1       *BAD TRACK SEEK
040.273  20BB   D.ERRL  DS      0       *LIMIT OF ERROR COUNTERS

NOTE: *INDICATES THAT THESE COUNTERS ARE NOT USED BY HDOS 3.0.

  *     I/O OPERATION COUNTS

040.273  20BB   D.OPR   DS      2       NUMBER OF READS
040.275  20BD   D.OPW   DS      2       NUMBER OF WRITES

000.037  001F   D.RAML  EQU     *-D.RAM

  *     S.VAL - SYSTEM VALUES
  *     These values are set and maintained by the system

040.277  20BF   S.VAL   DS      36      SYSTEM VALUES

040.277  20BF   S.DATE  DS      9       SYSTEM DATE (IN ASCII)
040.310  20C8   S.DATC  DS      2       CODED DATE
040.312  20CA   S.TIME  DS      3       TIME FROM MIDNIGHT (IN BCD)
040.315  20CD   S.CLKTR DS      1       CLOCK TASK RESIDENT FLAG

040.316  20CE   S.HIMEM DS      2       HARDWARE HIGH MEMORY ADDRESS

040.320  20D0   S.SYSM  DS      2       FWA RESIDENT SYSTEM

040.322  20D2   S.USRM  DS      2       LWA USER MEMORY

  *  CAUTION - The next two bytes used to be "S.OMAX". Older
  *  application programs may have referenced it for setting top of
  *  memory with ".SETTOP".  DO NOT use this word for anything now.
  *  In this way, it should always contain zero and should be
       harmless.

040.324  20D4           DS      2       Reserved  \3.0a\
```

APPENDIX 13-D: MEMORY (Cont)

```
------------------------------------------------------------------------
  *     The following cells should be modified/read ONLY
  *     via the .CONSL SCALL
------------------------------------------------------------------------

000.000  0000   I.CSLMD EQU     0       S.CSLMD IS FIRST BYTE
040.326  20D6   S.CSLMD DS      1       CONSOLE MODE

000.200  0080   CSL.ECH EQU     10000000B   SUPPRESS ECHO
000.004  0004   CSL.RAW EQU     00000100B   Raw Mode I/O
000.002  0002   CSL.WRP EQU     00000010B   WRAP LINES AT WIDTH
000.001  0001   CSL.CHR EQU     00000001B   OPERATE IN CHARACTER MODE

000.001  0001   I.CONTY EQU     1       S.CONTY IS 2ND BYTE
040.327  20D7   S.CONTY DS      1       CONSOLE TYPE FLAGS

000.200  0080   CTP.BKS EQU     10000000B   TERMINAL PROCESSES
                                             BACKSPACES
000.100  0040   CTP.FF  EQU     01000000B   Terminal Processes
                                             Form-Feed
000.040  0020   CTP.MLI EQU     00100000B   MAP LOWER CASE TO UPPER
                                             ON INPUT
000.020  0010   CTP.MLO EQU     00010000B   MAP LOWER CASE TO UPPER
                                             ON OUTPUT
000.010  0008   CTP.2SB EQU     00001000B   TERMINAL NEEDS TWO STOP
                                             BITS
000.004  0004   CTP.HHS EQU     00000100B   Terminal uses hdwr
                                             handshake
000.002  0002   CTP.BKM EQU     00000010B   MAP BKSP (UPON INPUT) TO
                                             RUBOUT
000.001  0001   CTP.TAB EQU     00000001B   TERMINAL SUPPORTS TAB
                                             CHARACTERS

000.002  0002   I.CUSOR EQU     2       S.CUSOR IS 3RD BYTE
040.330  20D8   S.CUSOR DS      1       CURRENT CURSOR POSITION

000.003  0003   I.CONWI EQU     3       S.CONWI IS 4TH BYTE
040.331  20D9   S.CONWI DS      1       CONSOLE WIDTH

000.004  0004   I.CONFL EQU     4       S.CONFL IS 5TH BYTE
040.332  20DA   S.CONFL DS      1       CONSOLE FLAGS

000.001  0001   CO.FLG  EQU     00000001B   CTL-O FLAG
000.200  0080   CS.FLG  EQU     10000000B   CTL-S FLAG

040.333  20DB   S.CAADR DS      2       ADDRESS FOR ABORT PROCESSING
                *                       (GREATER THAN 256 IF VALID)
040.335  20DD   S.CCTAB DS      6       ADDR FOR CTL-A, CTL-B, CTL-C
                                           PROCESSING
```

APPENDIX 13-D: MEMORY (Cont)

```
------------------------------------------------------------------------
  *     S.INT - SYSTEM INTERNAL WORKAREA
  *     These cells are referenced by main code, and
  *     MUST therefore reside in fixed low memory
------------------------------------------------------------------------

040.343  20E3   S.INT   DS      115     SYSTEM INTERNAL WORK AREAS

------------------------------------------------------------------------
  *     CONSOLE STATUS FLAGS
------------------------------------------------------------------------

040.343  20E3   S.CDB   DS      1       CONSOLE DESCRIPTOR BYTE
000.000  0000   CDB.H85 EQU     00000000B
000.001  0001   CDB.H84 EQU     00000001B
040.344  20E4   S.BAUD  DS      2       [0-14]  H8-4 BAUD RATE, =0 IF
                                           H8-5
    *                                [15]    =1 IF BAUD RATE =>
                                             2 STOP BITS
  *     Baud Rate   Divisor
  *     ---------   --------------------
  *     38400       000.003  0003h       3
  *     19200       000.006  0006h       6
  *      9600       000.014  000Ch      12
  *      4800       000.030  0018h      24
  *      2400       000.060  0030h      48
  *      1200       000.140  0060h      96
  *       600       000.300  00C0h     192
  *       300       001.200  0180h     384
  *       150       003.000  0300h     768
  *       110       004.027  0417h    1047
  *        75       006.000  0600h    1536

  *       NOTE:   A clever way to equate baud rate to divisor is
  *               found by knowing that <divisor>*<baud>/10=11520

  *     TABLE ADDRESS WORDS

040.346  20E6   S.DLINK DS      2       ADDRESS OF DATA IN HDOS CODE

------------------------------------------------------------------------
  *     HDOS MONITOR PRIVATE RAM AREA DEFINITIONS
  *     Pointed to by S.DLINK
------------------------------------------------------------------------

000.000  0000   M.SYSM  DS      1       SYSCALL ITERATION COUNT
000.001  0001           DS      1       STAND-ALONE FLAG (OBSOLETE)
000.002  0002   M.CSL   DS      2       Address of console data area
000.004  0004   M.SUNI  DS      1       SYSTEM UNIT NUMBER
000.005  0005   M.SYDD  DS      2       SYSTEM DEVICE DRIVER
```

APPENDIX 13-D: MEMORY (Cont)

```
------------------------------------------------------------------------
  *  CAUTION - The next two bytes used to be "S.OFWA". Older
  *  application programs may have referenced it for something
  *  to do with the obsolete overlay table.  DO NOT use this word
  *  for anything now.  In this way, it should always contain zero
  *  and should be harmless.
------------------------------------------------------------------------

040.350  20E8           DS      2       Reserved  \3.02\

040.352  20EA   S.CFWA  DS      2       FWA CHANNEL TABLE
040.354  20EC   S.DFWA  DS      2       FWA DEVICE TABLE
040.356  20EE   S.RFWA  DS      2       FWA RESIDENT HDOS CODE

------------------------------------------------------------------------
  *     DEVICE DRIVER DELAYED LOAD FLAGS
------------------------------------------------------------------------

040.360  20F0   S.DDLDA DS      2       DRIVER LOAD ADDRESS
        *                 (HIGH BYTE=0 IF NO LOAD PENDING)
040.362  20F2   S.DDLEN DS      2       CODE LENGTH IN BYTES
040.364  20F4   S.DDGRP DS      1       GROUP NUMBER FOR DRIVER
040.365  20F5           DS      1       HOLD PLACE
040.366  20F6   S.DDDTA DS      2       DEVICE'S ADDRESS IN DEVLST
                                          +DEV.RES
040.370  20F8   S.DDOPC DS      1       OPEN OPCODE PENDEDING

040.371  20F9           DS      13      Reserved

------------------------------------------------------------------------
  *     SYSCALL PROCESSING WORK AREAS
------------------------------------------------------------------------

041.006  2106   S.CACC  DS      1       (ACC) UPON SYSCALL
041.007  2107   S.CODE  DS      1       SYSCALL INDEX IN PROGRESS

------------------------------------------------------------------------
  *     JUMPS TO ROUTINES IN RESIDENT HDOS CODE
------------------------------------------------------------------------

041.010  2108   S.JUMPS DS      0       START OF DUMP VECTORS
041.010  2108   S.SDD   DS      3       JUMP TO STAND-IN DEVICE DRIVER
041.013  210B   S.FASER DS      3       JUMP TO FATSERR (FATAL SYSTEM
                                          ERROR)
041.016  210E   S.DIREA DS      3       JUMP TO DIREAD (DISK FILE READ)
041.021  2111   S.FCI   DS      3       JUMP TO FCI (FETCH CHANNEL INFO)
041.024  2114   S.SCI   DS      3       JUMP TO SCI (STORE CHANNEL INFO)
041.027  2117   S.GUP   DS      3       JUMP TO GUP (GET UNIT POINTER)

041.032  211A   S.MOUNT DS      1       <>0 IF THE SYSTEM DISK IS
                                           MOUNTED
041.033  211B           DS      1       Reserved
```

APPENDIX 13-D: MEMORY (Cont)

```
------------------------------------------------------------------------
    *     JUMPS TO ROUTINES IN RESIDENT HDOS CODE (Cont)
------------------------------------------------------------------------

041.034  211C   S.BOOTF DS      1       BOOT FLAGS
000.001  0001   BOOT.P  EQU     00000001B   EXECUTE PROLOGUE UPON BOOT

041.035  211D           DS      3       Reserved

------------------------------------------------------------------------
  **    ACTIVE I/O AREA
------------------------------------------------------------------------
  *     THE AIO.XXX AREA CONTAINS INFORMATION ABOUT THE I/O OPERATION
  *     CURRENTLY BEING PERFORMED. THE INFORMATION IS OBTAINED FROM
  *     THE CHANNEL TABLE, AND WILL BE RESTORED THERE WHEN DONE.
  *
  *     NORMALLY, THE AIO.XXX INFORMATION WOULD BE OBTAINED DIRECTLY
  *     FROM VARIOUS SYSTEM TABLES VIA POINTER REGISTERS. SINCE THE
  *     8080 HAS NO GOOD INDEXED ADDRESSING, THE DATA IS MANUALLY
  *     COPIED INTO THE AIO.XXX CELLS BEFORE PROCESSING, AND
  *     BACKDATED AFTER PROCESSING.

041.040  2120   AIO.VEC DS      3       JUMP INSTRUCTION
041.041  2121   AIO.DDA EQU     *-2     DEVICE DRIVER ADDRESS
041.043  2123   AIO.FLG DS      1       FLAG BYTE
041.044  2124   AIO.GRT DS      2       ADDRESS OF GROUP RESERV TABLE
041.046  2126   AIO.SPG DS      1       SECTORS PER GROUP
041.047  2127   AIO.CGN DS      1       CURRENT GROUP NUMBER
041.050  2128   AIO.CSI DS      1       CURRENT SECTOR INDEX
041.051  2129   AIO.LGN DS      1       LAST GROUP NUMBER
041.052  212A   AIO.LSI DS      1       LAST SECTOR INDEX
041.053  212B   AIO.DTA DS      2       DEVICE TABLE ADDRESS
041.055  212D   AIO.DES DS      2       DIRECTORY SECTOR
041.057  212F   AIO.DEV DS      2       DEVICE CODE
041.061  2131   AIO.UNI DS      1       UNIT NUMBER (0-7)

041.062  2132   AIO.DIR DS      DIRELEN   DIRECTORY ENTRY

041.111  2149   AIO.CNT DS      1       SECTOR COUNT
041.112  214A   AIO.EOM DS      1       END OF MEDIA FLAG
041.113  214B   AIO.EOF DS      1       END OF FILE FLAG
041.114  214C   AIO.TFP DS      2       TEMP FILE POINTERS
041.116  214E   AIO.CHA DS      2       ADDRESS OF CHANNEL BLOCK
                                          (IOC.DDA)

041.120  2150   S.BDA   DS      1       Boot Device Address
                                          (Setup by ROM)
041.121  2151   S.SCR   DS      2       SYSTEM SCRATCH AREA ADDRESS
041.123  2153           DS      3       Reserved
041.126  2156   S.OSI   DS      1       Operating system index
041.127  2157   S.OSO   DS      1       Operating system occurance
```

APPENDIX 13-D: MEMORY (Cont)

```
    **    ACTIVE I/O AREA (Cont)

041.130  2158   S.OSZ   DS      3       Operating system sector zero

041.133  215B           DS      11      Reserved

041.146  2166   S.SOVR  DS      2       STACK OVERFLOW WARNING
041.150  2168           DS      42200A-*   SYSTEM STACK
001.032  011A   STACKL  EQU     *-S.SOVR   STACK SIZE

042.200  2280   STACK   EQU     *       LWA+1 SYSTEM STACK
042.200  2280   USERFWA EQU     *       USER FWA

                ************************************
                ************************************
                **                                **
                **  USERFWA - User Code Starts HERE  **
                **                                **
                ************************************
                ************************************

*********************************************************************
```

APPENDIX 13-E: DIRECTORY ENTRY FORMAT

```
DIRECTORY ENTRY FORMAT
-----------------------------------------------------------------------
000.377  00FF   DF.EMP  EQU     377Q    ENTRY EMPTY
000.376  00FE   DF.CLR  EQU     376Q    ENTRY EMPTY, REMAINDER ALSO
                                          CLEAR

000.000  0000   DIR.NAM DS      8       NAME
000.010  0008   DIR.EXT DS      3       EXTENSION
000.013  000B   DIRIDL  EQU     *       file identification length

000.013  000B   DIR.CTH DS      1       creation time (BCD hours)
000.014  000C   DIR.CTM DS      1       creation time (BCD minutes)

000.015  000D   DIR.NOA DS      1       number of accesses
000.016  000E   DIR.FLG DS      1       FLAGS
000.017  000F   DIR.USR DS      1       user area mask
000.020  0010   DIR.FGN DS      1       FIRST GROUP NUMBER
000.021  0011   DIR.LGN DS      1       LAST GROUP NUMBER
000.022  0012   DIR.LSI DS      1       LAST SECTOR INDEX (IN LAST
                                          GROUP)
000.023  0013   DIR.CRD DS      2       CREATION DATE
000.025  0015   DIR.ACD DS      2       last access date

000.027  0017   DIRELEN EQU     *       DIRECTORY ENTRY LENGTH

-----------------------------------------------------------------------
DIRECTORY FILE FLAGS
-----------------------------------------------------------------------
000.200  0080   DIF.SYS EQU     10000000B   System file
000.100  0040   DIF.LOC EQU     01000000B   Locked from flag changes
000.040  0020   DIF.WP  EQU     00100000B   Write protected
000.020  0010   DIF.CNT EQU     00010000B   Contiguous file
000.010  0008   DIF.ARC EQU     00001000B   File archive attribute
000.004  0004   DIF.BAD EQU     00000100B   File is damaged
000.002  0002   DIF.DL  EQU     00000010B   Locked against delete
000.001  0001   DIF.USR EQU     00000001B   User-defined

-----------------------------------------------------------------------
DIRECTORY BLOCK FORMAT
-----------------------------------------------------------------------
000.000  0000   DIS.ENT EQU     *       FIRST ENTRY ADDRESS

000.000  0000           DS      22*DIRELEN   22 DIRECTORY ENTRYS PER
                                               BLOCK
001.372  01FA           DS      1       0 BYTE = END OF ENTRYS IN THIS
                                               BLOCK

001.373  01FB           ORG     512-5   AT END OF BLOCK
```

APPENDIX 13-E: DIRECTORY ENTRY FORMAT (Cont)

```
------------------------------------------------------------------------
DIRECTORY BLOCK FORMAT (Cont)
------------------------------------------------------------------------

001.373  01FB   DIS.ENL DS      1       LENGTH OF EACH ENTRY (=DIRELEN)
001.374  01FC   DIS.SEC DS      2       BLOCK # OF THIS BLOCK,
001.376  01FE   DIS.LNK DS      2       BLOCK # OF NEXT BLOCK,=0 IF LAST

------------------------------------------------------------------------
DIRECTORY DEVICE FORMAT DEFINITION
------------------------------------------------------------------------
000.000  0000   DDF.BOO DS      9       2K BOOT PROGRAM
000.011  0009   DDF.BOL EQU     *       LENGTH OF BOOT
000.011  0009   DDF.LAB DS      1       LABEL SECTOR
000.012  000A   DDF.USR DS      0       BEGINNING OF OPEN SPACE

------------------------------------------------------------------------
DISK LABEL SECTOR FORMATS
------------------------------------------------------------------------
000.000  0000   LAB.SER DS      1       SERIAL NUMBER OF VOLUME
000.001  0001   LAB.IND DS      2       INITIALIZATION DATE
000.003  0003   LAB.DIS DS      2       SECTOR NUMBER OF 1ST DIRECTORY
                                          SECTOR
000.005  0005   LAB.GRT DS      2       INDEX OF GRT SECTOR
000.007  0007   LAB.SPG DS      1       SECTORS PER GROUP

000.000  0000   LAB.DAT EQU     0       DATA VOLUME ONLY
000.001  0001   LAB.SYS EQU     1       SYSTEM VOLUME
000.002  0002   LAB.NOD EQU     2       VOLUME HAS NO DIRECTORY

000.010  0008   LAB.VLT DS      1       VOLUME TYPE
000.011  0009   LAB.VER DS      1       VERSION OF INIT17 THAT INITED
                                          DISK

000.012  000A   LAB.RGT DS      2       RGT sector number

000.014  000C   LAB.VPR EQU     *       Volume dependant data
000.014  000C   LAB.SIZ DS      2       Volume Size (Bytes/256)
000.016  000E   LAB.PSS DS      2       Physical Sector Size
000.020  0010   LAB.VFL DS      1       Volume dependant Flags
000.001  0001   VFL.NSD EQU     00000001B   Number of Sides:  1 => 2
000.002  0002   VFL.DTD EQU     00000010b   96 tracks per inch
000.004  0004   VFL.FIX EQU     00000100b   Media is fixed
000.005  0005   LAB.VPL EQU     *-LAB.VPR   Length of volume dependant
                                             data

000.021  0011            DS     5-LAB.VPL   Reserved

000.021  0011   LAB.LAB DS      60      LABEL
000.074  003C   LAB.LBL EQU     *-LAB.LAB   LABEL LENGTH
000.115  004D            DS     2       Reserved for 0 bytes
```

APPENDIX 13-E: DIRECTORY ENTRY FORMAT (Cont)

```
------------------------------------------------------------------------
DISK LABEL SECTOR FORMAT
------------------------------------------------------------------------

000.117  004F   LAB.AUX EQU     *       Auxiliary Data
000.117  004F   LAB.SPT DS      1       Sectors per Track
000.120  0050   LAB.LVN DS      2       long volume number
000.003  0003   LAB.AXL EQU     *-LAB.AUX   Length of Aux. Data

------------------------------------------------------------------------
DEVICE TABLE ENTRIES
------------------------------------------------------------------------
000.000  0000   DEV.NAM DS      2       DEVICE NAME
000.000  0000   DV.EL   EQU     00000000B   END OF DEVICE LIST FLAG
000.001  0001   DV.NU   EQU     00000001B   DEVICE ENTRY NOT IN USE

000.002  0002   DEV.RES DS      1       DRIVER RESIDENCE CODE
000.001  0001   DR.IM   EQU     00000001B   DRIVER IN MEMORY
000.002  0002   DR.PR   EQU     00000010B   DRIVER PERMANENTLY RESIDENT
000.004  0004   DR.FX   EQU     00000100B   Driver FIXED in memory
000.010  0008   DR.UNL  EQU     00001000B   Driver unload pending
                 *       EQU     00010000B
000.340  00E0   DR.SPL  EQU     11100000B   SET preamble length mask

000.003  0003   DEV.JMP DS      1       JMP TO PROCESSOR
000.004  0004   DEV.DDA DS      2       DRIVER ADDRESS
000.006  0006   DEV.FLG DS      1       FLAG BYTE
000.001  0001   DT.DD   EQU     00000001B   DIRECTORY DEVICE
000.002  0002   DT.CR   EQU     00000010B   CAPABLE OF READ OPERATION
000.004  0004   DT.CW   EQU     00000100B   CAPABLE OF WRITE OPERATION
000.010  0008   DT.RN   EQU     00001000B   Capable of random access
000.020  0010   DT.CH   EQU     00010000B   Capable of character mode
000.040  0020   DT.FX   EQU     00100000B   Media is Fixed
000.100  0040   DT.P3   EQU     01000000B   Media is Pre-3.0
000.200  0080   DT.UL   EQU     10000000B   Requires Unload Notification

000.007  0007   DEV.MUM DS      1       MOUNTED UNIT MASK
000.010  0008   DEV.MNU DS      1       MAXIMUM NUMBER OF UNITS
000.011  0009   DEV.UNT DS      2       ADDRESS OF UNIT SPECIFIC DATA
                                           TABLE

000.013  000B   DEV.DVL DS      2       DRIVER BYTE LENGTH
000.015  000D   DEV.DVG DS      1       DRIVER ROUTINE GROUP ADDRESS

000.016  000E   DEVELEN EQU     *       DEVICE TABLE ENTRY LENGTH

------------------------------------------------------------------------
UNIT SPECIFIC DEVICE DATA TABLE ENTRIES
------------------------------------------------------------------------
000.000  0000   UNT.FLG DS      1       UNIT SPECIFIC  *DEV.FLG*
000.001  0001   UNT.SPG DS      1       Sectors Per Group
000.002  0002   UNT.GRT DS      2       ADDRESS OF GRT (IF DT.DD)
```

APPENDIX 13-E: DIRECTORY ENTRY FORMAT (Cont)

```
UNIT SPECIFIC DEVICE DATA TABLE ENTRIES (Cont)
-------------------------------------------------------------------

000.004  0004   UNT.GTS DS      2       GRT SECTOR NUMBER
000.006  0006   UNT.DIS DS      2       DIRECTORY FIRST SECTOR NUMBER

000.010  0008   UNT.SIZ EQU     *       SIZE OF UNIT SPECIFIC DATA ENTRY

-------------------------------------------------------------------
DEVICE DRIVER EQUIVALENCES
-------------------------------------------------------------------
000.307  00C7   DVDFLV  EQU     307Q    DEVICE DRIVER FLAG VALUE

000.006  0006           ORG     PIC.COD   STARTS AT PIC CODE AREA

000.006  0006   DVD.DVD DS      1       MUST BE DVDFLV, IDENTIFIES AS
                                          DRIVER
000.007  0007   DVD.CAP DS      1       DEVICE CAPABILITY FLAG
000.010  0008   DVD.MUM DS      1       MOUNTED UNITS MASK
000.011  0009   DVD.MNU DS      1       MAXIMUM NUMBER OF UNITS
000.012  000A   DVD.UFL DS      8       UNIT CAPABILITY FLAGS FOR
                                          DRIVER UNITS 0-7
000.022  0012   DVD.SET DS      1       = DVDFLV IF DRIVER WILL TAKE
                                            SET OPTIONS
000.023  0013   DVD.INP DS      2       Pointer to Init Code
000.025  0015   DVD.V30 DS      1       = DVDFLV IF HDOS 3.0 Driver
                                            /3.0a/
000.026  0016   DVD.SPL DS      1       SET preamble size (pages/2)
                                           /3.0a/
000.027  0017           DS      20      RESERVED, MUST BE 0  /3.0a/
000.053  002B   DVD.STE EQU     *       ENTRY FOR 'SET' INVOCATION

002.000  0200   DVD.ENT EQU     2000A   DRIVER ENTRY POINT(MULT OF 512)

-------------------------------------------------------------------
DEVICE DRIVER COMMUNICATION FLAGS
-------------------------------------------------------------------
000.000  0000   DC.REA  DS      1       READ
000.001  0001   DC.WRI  DS      1       WRITE
000.002  0002   DC.RER  DS      1       READ REGARDLESS
000.003  0003   DC.OPR  DS      1       OPEN FOR READ
000.004  0004   DC.OPW  DS      1       OPEN FOR WRITE
000.005  0005   DC.OPU  DS      1       OPEN FOR UPDATE
000.006  0006   DC.CLO  DS      1       CLOSE
000.007  0007   DC.ABT  DS      1       ABORT
000.010  0008   DC.MOU  DS      1       MOUNT DEVICE
000.011  0009   DC.LOD  DS      1       LOAD DEVICE DRIVER
000.012  000A   DC.RDY  DS      1       Device Ready
000.013  000B   DC.SET  DS      1       Update SET parameters
000.014  000C   DC.UNL  DS      1       Unload device driver
```

APPENDIX 13-E: DIRECTORY ENTRY FORMAT (Cont)

DEVICE DRIVER COMMUNICATION FLAGS (Cont)

```
000.015  000D   DC.INT  DS      1       Interrupt
000.016  000E   DC.DSF  DS      1       Device-specific function
000.017  000F   DC.MAX  DS      1       MAXIMUM ENTRY INDEX
```

STANDARD DISK FORMATS FOR HDOS 3.0

DEVICE	SIDES	DENSITY	TPI	TRACKS	SPT	GROUPS	SPG	SIZE
H17	1	SINGLE	48	40	10	200	2	400
H17	1	SINGLE	96	80	10	200	4	800
H17	2	SINGLE	48	40	10	200	4	800
H17	2	SINGLE	96	80	10	200	8	1600
H37	1	SINGLE	48	40	10	200	2	400
H37	1	SINGLE	96	80	10	200	4	800
H37	2	SINGLE	48	40	10	200	4	800
H37	2	SINGLE	96	80	10	200	8	1600
H37	1	DOUBLE	48	40	16	160	4	640
H37	1	DOUBLE	96	80	16	213	6	1278
H37	2	DOUBLE	48	40	16	213	6	1278
H37	2	DOUBLE	96	80	16	255	10	2550

APPENDIX 13-E: DIRECTORY ENTRY FORMAT (Cont)

STANDARD DISK FORMATS FOR HDOS 3.0 (Cont)								
H47	1	SINGLE	48	77	13	250	4	1000
H47	1	DOUBLE	48	77	26	250	8	2000
H47	2	SINGLE	48	77	13	250	8	2000
H47	2	DOUBLE	48	77	26	250	16	4000
IOMEGA	1	?	?	77	64	244	20	4880
EIGHT (8) LOGICAL UNITS PER CARTRIDGE								
IOMEGA	1	?	?	77	512	1952	20	39040

SUPER 89 RAM DISK by Dean Gibson/Ultimeth								
BANKS	SIDES	DENSITY	TPI	TRACKS	SPT	GROUPS	SPG	SIZE
1	1	N/A	N/A	127	2	127	2	254
2	1	N/A	N/A	254	2	254	2	508
3	1	N/A	N/A	380	2	190	4	760

I/O CHANNEL DEFINITIONS

```
000.000  0000   IOC.LNK DS      2       ADDRESS OF NEXT CHANNEL, =0
                                          IF LAST
000.002  0002   IOC.DDA DS      2       THREAD JUMP TO DEVICE DRIVER
                                          (VIA DEV TABLE)

000.004  0004   IOC.FLG DS      1       FILE TYPE FLAGS
000.001  0001   FT.DD   EQU     00000001B   DIRECTORY DEVICE
000.002  0002   FT.OR   EQU     00000010B   OPEN FOR READ
000.004  0004   FT.OW   EQU     00000100B   OPEN FOR WRITE
000.010  0008   FT.OU   EQU     00001000B   OPEN FOR UPDATE
000.020  0010   FT.OC   EQU     00010000B   OPEN FOR CHARACTER MODE
000.003  0003   IOC.SQL EQU     *-IOC.DDA   LENGTH OF INFO FOR
                                            SEQUENTIAL FILE
```

APPENDIX 13-E: DIRECTORY ENTRY FORMAT (Cont)

```
I/O CHANNEL DEFINITIONS (Cont)
------------------------------------------------------------------------

000.005  0005   IOC.GRT DS      2       ADDRESS OF GROUP RESERVATION
                                         TABLE
000.007  0007   IOC.SPG DS      1       SECTORS PER GROUP, THIS DEVICE
000.010  0008   IOC.CGN DS      1       CURRENT GROUP NUMBER
000.011  0009   IOC.CSI DS      1       CURRENT SECTOR INDEX (IN
                                         CURRENT GROUP)
000.012  000A   IOC.LGN DS      1       LAST GROUP NUMBER
000.013  000B   IOC.LSI DS      1       LAST SECTOR INDEX (IN LAST
                                         GROUP)
000.010  0008   IOC.DRL EQU     *-IOC.FLG   LENGTH OF INFO NORMALLY
                                             COPIED
                 *                           BACK TO THE CHANNEL TABLE
000.014  000C   IOC.DTA DS      2       DEVICE TABLE ADDRESS FOR THIS
                                         DEVICE
000.016  000E   IOC.DES DS      2       SECTOR NUMBER OF DIRECTORY
                                         ENTRY
000.020  0010   IOC.DEV DS      2       DEVICE CODE
000.022  0012   IOC.UNI DS      1       UNIT NUMBER (0-9)

000.021  0011   IOC.DIL EQU     *-IOC.DDA   LENGTH OF INFO FOR
                                             DIRECTORY FILE
000.023  0013   IOC.DIR DS      DIRELEN     DIRECTORY ENTRY

000.052  002A   IOCELEN EQU     *       IOC ENTRY LENGTH

000.001  0001   IOCCTD  EQU     1       INDEX OF USER CHANNEL #0 IN
                                         CHANTAB (FIRST = 0)

------------------------------------------------------------------------
FILE BLOCK DEFINITIONS
------------------------------------------------------------------------
000.000  0000   FB.CHA  DS      1       CHANNEL NUMBER
000.001  0001   FB.FLG  DS      1       FLAGS
000.002  0002   FB.FWA  DS      2       BUFFER FWA
000.004  0004   FB.PTR  DS      2       BUFFER POINTER
000.006  0006   FB.LIM  DS      2       LIMIT OF DATA IN BUFFER
                                         (READ OPERATIONS)
000.010  0008   FB.LWA  DS      2       LWA OF BUFFER
000.012  000A   FB.NAM  DS      4+8+4+1 NAME OF FILE
000.021  0011   FB.NAML EQU     *-FB.NAM
000.033  001B   FBENL   EQU     *       ENTRY LENGTH
```

APPENDIX 13-E: DIRECTORY ENTRY FORMAT (Cont)

```
------------------------------------------------------------------------
FILDEF - FILE TYPE DEFINITIONS
------------------------------------------------------------------------
DB      377Q,FT.XXX
------------------------------------------------------------------------
000.000  0000   FT.ABS  EQU     0       ABSOLUTE BINARY
000.001  0001   FT.PIC  EQU     1       POSITION INDEPENDANT CODE
000.002  0002   FT.REL  EQU     2       RELOCATABLE CODE
000.003  0003   FT.BAC  EQU     3       COMPILED BASIC CODE

000.020  0010   FT.BSX  EQU     10H     Compiled BASEX Code

------------------------------------------------------------------------
ABS FORMAT EQUIVALENCES
------------------------------------------------------------------------
000.000  0000   ABS.ID  DS      1       377Q = BINARY FILE FLAG
000.001  0001           DS      1       FILE TYPE (FT.ABS)
000.002  0002   ABS.LDA DS      2       LOAD ADDRESS
000.004  0004   ABS.LEN DS      2       LENGTH OF ENTIRE RECORD
000.006  0006   ABS.ENT DS      2       ENTRY POINT

000.010  0008   ABS.COD DS      0       CODE STARTS HERE

------------------------------------------------------------------------
PIC FORMAT EQUIVALENCES
------------------------------------------------------------------------
000.000  0000   PIC.ID  DS      1       377Q = BINARY FILE FLAG
000.001  0001           DS      1       FILE TYPE (FT.PIC)
000.002  0002   PIC.LEN DS      2       LENGTH OF ENTIRE RECORD
000.004  0004   PIC.PTR DS      2       INDEX OF START OF PIC TABLE

000.006  0006   PIC.COD DS      0       CODE STARTS HERE

------------------------------------------------------------------------
REL FORMAT EQUIVALENCES
------------------------------------------------------------------------
000.000  0000   REL.ID  DS      1       377Q = BINARY FILE FLAG
000.001  0001           DS      1       FILE TYPE (FT.REL)
000.002  0002   REL.LEN DS      2       LENGTH OF ENTIRE RECORD
000.004  0004   REL.PTR DS      2       INDEX OF START OF REL TABLE
000.006  0006   REL.COD DS      0       CODE STARTS HERE

------------------------------------------------------------------------
BASEX HEADER EQUIVALENCES
NOTE: For more information about BASEX, contact Mighty/Soft
------------------------------------------------------------------------
000.000  0000   BSX.ID  DS      1       377Q = Binary File Flag
000.001  0001           DS      1       File Type (FT.BSX)
000.002  0002   BSX.PSA DS      2       Program Start Address
000.004  0004   BSX.PEA DS      2       Program End Address
000.006  0006   BSX.SSA DS      2       Symbol_Table Start Address
000.010  0008   BSX.SEA DS      2       Symbol_Table End Address
```

APPENDIX 13-E: DIRECTORY ENTRY FORMAT (Cont)

```
------------------------------------------------------------------------
BASEX HEADER EQUIVALENCES (Cont)
------------------------------------------------------------------------

000.012  000A   BSX.COD DS      0       Code Starts Here
------------------------------------------------------------------------
TASK PROCESSOR DEFINITIONS
------------------------------------------------------------------------
000.101  0041   .TASK   EQU     101Q    Process TASK function
000.317  00CF   TASKID  EQU     317Q    Task Identification Flag
000.327  00D7   TASKID. EQU     327Q

  **    TASK Processor Function Codes
  *
  *     Calling sequences.

  **    TAS.ID - Identify TASK
  *
  *       TAS.ID  is  used to identify a TASK to the Task Manager and
  *       the system.  The TASK is flagged as ACTIVE upon return with
  *       the Task Sequence Number (TSN) in (A).
  *       Entry:  (HL) = Address of TASK Block
  *           (B)  = TAS.ID
  *     Exit: 'C' Set
  *            (A) = Error code
  *           'C' Clear if Ok.
  *            (A) = Task Sequence Number
  *     Uses: ALL

000.000  0000   TAS.ID  DS      1       Identify TASK to System

  **    TAS.INQ - Inquire About TASKs in System
  *
  *       TAS.INQ  returns  a  pointer to the task block table in the
  *       Task Manager or to a task block within an individual  task.
  *       The  task block table consists of the addresses of the task
  *       blocks of all tasks known to the system.  the Task Manager.
  *       This  table consists of the addresses An address of 000000A
  *       signifies an unused entry, while 377377A marks the  end  of
  *       the table.  The general form of the table is:
  *
  *           DW    Address_of_TASK_Block
  *           DW    TASK_Process_Flag
  *
  *       The TASK Process flag contains a '1' bit for each interrupt
  *       vector the TASK services.  The bits  are  assigned  in  the
  *       following manner:
```

APPENDIX 13-E: DIRECTORY ENTRY FORMAT (Cont)

TASK PROCESSOR DEFINITIONS (Cont)

TAS.INQ - Inquire About Tasks in System (Cont)

```
  *
  *           Byte 0 :
  *
  *               bit 7 = High order bit, Interrupt Level 7
  *                 .
  *                 .
  *               bit 1 =           Interrupt Level 1
  *               bit 0 = Low order bit,  TASK is active
  *
  *       For  example, a value of 10001000B (or 210Q) would indicate
  *       that the TASK services the  SCALL  vector  (INT7)  and  the
  *       console  vector  (INT3).   Also  the  task is not currently
  *       active.
  *
  *       Byte 1 :
  *
  *               Reserved for future use
  *
  *     Entry : (A) = TSN desired, or -1 for base of task block
  *
  *     Exit: 'C' Clear
  *            (HL) = Address of TASK block table
  *           'C' Set
  *            (A) = Error code  (Illegal SCALL)
  *            TASK Monitor not STARTed
  *     Uses: A,F,H,L

000.001  0001   TAS.INQ DS      1       Inquire about TASKs

  **    TAS.DEA - Deactivate a TASK
  *
  *       TAS.DEA  flags  a  TASK  as  inactive  and discontinues all
  *       processing of interrupts by the TASK.  That  is,  the  Task
  *       Manager  will  not  pass  control  to the task at interrupt
  *       time.  The task's function processor will receive  control,
  *       (with  the  value  of TAS.DEA in the (A) register), to take
  *       care of disabling interrupts, and  other  'clean  up'  type
  *       activity before deactivation.
  *
  *     *WARNING*
  *
  *       If  an  interrupt  occurs  after deactivation which must be
  *       serviced by this TASK, the system will  crash  due  to  the
  *       unserviced interrupt.  The ONLY way to re-activate any task
  *       is via the TAS.REA function.
```

APPENDIX 13-E: DIRECTORY ENTRY FORMAT (Cont)

TASK PROCESSOR DEFINITIONS (Cont)

TAS.INQ - Inquire About Tasks in System (Cont)

```
  *
  *     Entry:      (A) = Task Sequence Number
  *     Exit: 'C' Clear, TASK deactivated
  *           'C' Set
  *            (A) = Error code
  *            No such TASK, TASK monitor not STARTed,
  *            TASK not abortable, or TASK not active
  *     Uses: A,F

000.002  0002   TAS.DEA DS      1       Deactivate Task

  **    TAS.REA - Re-activate a TASK
  *
  *       TAS.REA   Re-activates   a   previously  deactivated  task.
  *       Control is passed to the task's function  processor,  (with
  *       the  value of TAS.REA in the (A) register), to take care of
  *       any   initialization   which   may   be   required.    Such
  *       initialization  may include, but not be limited to, setting
  *       up of ports, and re-requesting interrupt service  from  the
  *       Task manager
  *
  *     Entry:      (A) = Task Sequence Number
  *     Exit: 'C' Clear, TASK re-activated
  *           'C' Set
  *            No such TASK, TASK monitor not STARTed, or
  *            TASK already active
  *     Uses: A,F


000.003  0003   TAS.REA DS      1       Reactivate TASK

  **    TAS.RIS - Request Interrupt Service
  *
  *       TAS.RIS  is  called  by  a  task  to request that interrupt
  *       service be provided by the Task Manager to  the  task.   It
  *       allows  the  Manager  to control access of interrupt-driven
  *       tasks, and removes the burden from the user of  setting  up
  *       and/or clearing of interrupt service vectors.
  *
  *     *WARNING*
  *
  *     This function must be called before any interrupts occur.
  *
```

APPENDIX 13-E: DIRECTORY ENTRY FORMAT (Cont)

TASK PROCESSOR DEFINITIONS (Cont)

TASK.REA - Reactiveate A Task (Cont)

```
  *       Entry:  (A)  = Task Sequence Number
  *           (B)  = TAS.RIS
  *           (C)  = Interrupt Level  (1-7)
  *           (HL) = Interrupt processor address
  *     Exit: 'C' clear
  *            Vectors installed
  *           'C' set
  *            (A) = Error code
  *            Unknown TSN, or Task Manager not present
  *     Uses: A,F
  *
  *       Upon entry to the particular interrupt service routine the
  *       user must follow the following conventions;
  *
  *     Vectors 1-6 :     All registers pushed on the stack.
  *           (SP+0) = return to task manager
  *           (SP+2) = return to user program via USR.RST
  *           (SP+4 - SP+6) = Task Manager registers
  *           (SP+8) = return to user program via $RSTALL
  *           (SP+10 - SP+18) = user registers
  *           (SP+20) = users interrupted PC
  *
  *     Vector 7 (SCALL) :
  *
  *           (HL) = users return address (pointer to SCALL)
  *           (SP+0) = return to task manager
  *           (SP+2) = task managers (HL)
  *           (SP+4) = users PSW
  *           (SP+6) = users HL
  *
  *       All  other  registers  unaffected.   If the task intends to
  *       handle the SCALL the task managers return should be  popped
  *       and the SCALL handled normally

000.004  0004   TAS.RIS DS      1       Request interrupt service

000.005  0005   TAS.MAX DS      0
```

APPENDIX 13-E: DIRECTORY ENTRY FORMAT (Cont)

TASK PROCESSOR DEFINITIONS (Cont)

```
  ***   TASK Block Definition.
  *
  *       The TASK block is a structure used to identify TASKs to the
  *       TASK monitor, and the system.  The format of the  block  is
  *       as follows:
  *
  *           TASK Name   DB    'TASKNAME'
  *           Version (BCD)     DB    BCD_VERSION
  *           TASK Id     DB    'XXXX'
  *           Task Status DB    STATUS_BYTE
  *           Task Start  DW    START_ADDRESS
  *           Task End    DW    END_ADDRESS
  *           Task Processor    DW    PROCESSOR_ADDRESS

000.000  0000   TSB.NAM DS      8       Name of the TASK
000.010  0008   TSB.VER DS      1       Version (BCD) of TASK
000.011  0009   TSB.ID  DS      4       Other Id information
000.015  000D   TSB.STA DS      1       Task status byte
                                         (See definitions)
000.016  000E   TSB.STR DS      2       Starting address of TASK
                                         in memory
000.020  0010   TSB.END DS      2       Ending address of TASK in
                                         memory
000.022  0012   TSB.PRC DS      2       Address of TASK function
                                         processor
                *                       ... If = 0, then no processor.
000.024  0014   TSB.LEN EQU     *       20 BYTES
```

STATUS BYTE DEFINITIONS

```
000.001  0001   TSS.ACT EQU     00000001B   Task is active
000.000  0000   TSS.DEA EQU     00000000B   Task is inactive
                                             (suspended)
000.002  0002   TSS.UFP EQU     00000010B   Task uses H8 front panel
000.010  0008   TSS.MEM EQU     00001000B   Task uses B/S memory
000.200  0080   TSS.TCA EQU     10000000B   Task may not be
                                             de-activated.

TASMAX  EQU     16              Maximum number of tasks in system
*****************************************************************
```

APPENDIX 13-F: ROMCALLS

USEFUL SUBROUTINES FROM THE ORIGINAL H17-ROM

```
***     $COMP - COMPARE TWO CHARACTER STRINGS.

*
*       $COMP COMPARES TWO BYTE STRINGS.
*
*       ENTRY:  (C) = COMPARE COUNT
*               (DE) = FWA OF STRING #1
*               (HL) = FWA OF STRING #2
*       EXIT:   'Z' CLEAR, IS MIS-MATCH
*               (C) = LENGTH REMAINING
*               (DE) = ADDRESS OF MISMATCH IN STRING #1
*               (HL) = ADDRESS OF MISMATCH IN STRING #2
*               'Z' SET, HAVE MATCH
*               (C) = 0
*               (DE) = (DE) + (0C)
*               (HL) = (HL) + (0C)
*       USES:   A,F,C,D,E,H,L

$COMP   EQU     30060A  1830H   6192
.....................................................................

***     $DADA - PERFORM (H,L) = (H,L) + (0,A)
*
*       ENTRY:  (H,L) = BEFORE VALUE
*               (A) = BEFORE VALUE
*       EXIT:   (H,L) = (H,L) + (0,A)
*               'C' SET IF OVERFLOW
*       USES:   F,H,L

$DADA   EQU     30072A  183AH   6202
.....................................................................

***    $DADA. - ADD (0,A) TO (H,L)
*
*       ENTRY:  NONE
*       EXIT:   (HL) = (HL) + (0A)
*       USES:   A,F,H,L

$DADA.  EQU     30101A  1841H   6209
.....................................................................
```

APPENDIX 13-F: ROMCALLS (Cont)
++++++++++++++++++++++++++++++

```
***     $DU66 - UNSIGNED 16 / 16 DIVIDE.
*
*       (HL) = (BC)/(DE)
*
*       ENTRY:  (BC), (DE) PRESET
*       EXIT:   (HL) = RESULT
*               (DE) = REMAINDER
*       USES:   ALL

$DU66   EQU     30106A  1846H   6214
.....................................................................

***     $HLIHL - LOAD HL INDIRECT THROUGH HL.
*
*       (HL) = ((HL))
*
*       ENTRY:  NONE
*       EXIT:   NONE
*       USES:   A,H,L

$HLIHL  EQU     30211A  1889H   6281
.....................................................................

***     $CDEHL - COMPARE (DE) TO (HL)
*
*       $CDEHL COMPARES (DE) TO (HL) FOR EQUALITY.
*
*       ENTRY:  NONE
*       EXIT:   'Z' SET IF (DE) = (HL)
*       USES:   A,F

$CDEHL  EQU     30216A  188EH   6286
.....................................................................

***     $CHL - COMPLEMENT (HL).
*
*       (HL) = -(HL)            TWO'S COMPLEMENT
*
*       ENTRY:  NONE
*       EXIT:   NONE
*       USES:   A,F,H,L

$CHL    EQU     30224A  1894H   6292
.....................................................................

***     $INDL - INDEXED LOAD.
*
*       $INDL LOADS DE WITH THE TWO BYTES AT (HL)+DISPLACMENT
*
*       THIS ACTS AS AN INDEXED FULL WORD LOAD.
*       (DE) = ( (HL) + DSPLACEMENT )
```

APPENDIX 13-F: ROMCALLS (Cont)
++++++++++++++++++++++++++++++

```
.$INDL - INDEXED LOAD (Cont)

*       ENTRY:  ((RET)) = DISPLACMENT (FULL WORD)
*               (HL) = TABLE ADDRESS
*       EXIT:   TO (RET+2)
*       USES:   A,F,D,E

$INDL   EQU     30234A  189CH   6300
.....................................................................

***     $MOVE - MOVE DATA
*
*       $MOVE MOVES A BLOCK OF BYTES TO A NEW MEMORY ADDRESS.
*       IF THE MOVE IS TO A LOWER ADDRESS, THE BYTES ARE MOVED FROM
*       FIRST TO LAST.
*
*       IF THE MOVE IS TO A HIGHER ADDRESS, THE BYTES ARE MOVED FROM
*       LAST TO FIRST.
*
*       THIS IS DONE SO THAT AN OVERLAPED MOVE WILL NOT 'RIPPLE'.
*
*       ENTRY:  (BC) = COUNT
*               (DE) = FROM
*               (HL) = TO
*       EXIT:   MOVED
*               (DE) = ADDRESS OF NEXT FROM BYTE
*               (HL) = ADDRESS OF NEXT *TO* BYTE
*               'C' CLEAR
*       USES:   ALL

$MOVE   EQU     30252A  18AAH   6314
.....................................................................

***     $MU10 - MULTIPLY UNSIGNED 16 BIT QUANTITY BY 10
*
*       (HL) = (DE) * 10
*
*       ENTRY:  (DE) = MULTIPLIER
*       EXIT:   'C' CLEAR IF OK
*               (HL) = PRODUCT
*               'C' SET IF ERROR
*       USES:   F,D,E,H,L

$MU10   EQU     30324A  18D4H   6356
.....................................................................
```

APPENDIX 13-F: ROMCALLS (Cont)
++++++++++++++++++++++++++++++

```
***     $MU66 - UNSIGNED 16 X 16 MULTIPLY.
*
*       ENTRY:  (BC) = MULTIPLICAND
*               (DE) = MULTIPLIER
*       EXIT:   (HL) = RESULT
*               'Z' SET IF NOT OVERFLOW
*       USES:   ALL

$MU66   EQU     30337A  18DFH   6367
.....................................................................

***     $MU86 - MULTIPLY 8 X 16 UNSIGNED.
*
*       ENTRY:  (A) = MULTIPLIER
*               (DE) = MULTIPLICAND
*       EXIT:   (HL) = RESULT
*               'Z' SET IF NOT OVERFLOW
*       USES:   A,F,H,L

$MU86   EQU     31007A  1907H   6407
.....................................................................

***     $RSTALL - RESTORE ALL REGISTERS.
*
*       $RSTALL RESTORES ALL THE REGISTERS OFF THE STACK, AND

*       RETURNS TO THE PREVIOUS CALLER.
*
*       ENTRY:  (SP) = PSW
*               (SP+2) = BC
*               (SP+4) = DE
*               (SP+6) = HL
*               (SP+8) = RET
*       EXIT:   TO *RET*, REGISTERS RESTORED
*       USES:   ALL

$RSTALL EQU     31047A  1927H   6439
.....................................................................

***     $SAVALL - SAVE ALL REGISTERS ON STACK.
*
*       $SAVALL SAVES ALL THE REGISTERS ON THE STACK.
*
*       ENTRY:  NONE
*       EXIT:   (SP) = PSW
*               (SP+2) = BC
*               (SP+4) = DE
*               (SP+6) = HL
*       USES:   H,L

$SAVALL EQU     31054A  192CH   6444
```

APPENDIX 13-F: ROMCALLS (Cont)

```
***     $TJMP - TABLE JUMP.
*
*       USAGE
*
*       CALL    $TJMP           (A) = INDEX
*       DW      ADDR1           INDEX = 0
*       .         .
*       .         .
*       DW      ADDRN           INDEX = N - 1
*
*       ENTRY:  (A) = INDEX
*       EXIT:   TO PROCESSOR
*               (A) = INDEX * 2
*       USES:   NONE.

$TJMP   EQU     31061A  1931H   6449

*       ENTRY:  (A) = INDEX * 2

$TJMP.  EQU     31062A  1932H   6450
.....................................................................

***     $TBRA - BRANCH RELATIVE THROUGH TABLE.
*
*       $TBRA USES THE SUPPLIED INDEX TO SELECT A BYTE FROM THE
*       JUMP TABLE. THE CONTENTS OF THIS BYTE ARE ADDED TO THE
*       ADDRESS OF THE BYTE, YIELDING THE PROCESSOR ADDRESS.
*
*       CALL    $TBRA
*       DB      LAB1-*          ; INDEX = 0 FOR LAB1
*       DB      LAB2-*          ; INDEX = 1 FOR LAB2
*       DB      LABN-*          ; INDEX = N-1 FOR LABN
*
*       ENTRY:  (A) = INDEX
*               (RET) = TABLE FWA
*       EXIT:   TO COMPUTED ADDRESS
*       USES:   F,H,L

$TBRA   EQU     31076A  193EH   6462
.....................................................................

***     $TBLS - TABLE SEARCH
*
*       TABLE FORMAT
*
*       DB      KEY1,VAL1,
*       .       .
*       .       .
*       DB      KEYN,VALN
*       DB      0
*
```

APPENDIX 13-F: ROMCALLS (Cont)
++++++++++++++++++++++++++++++

```
.$TBLS (Cont)

*       ENTRY:  (A) = PATTERN
*               (H,L) = TABLE FWA
*       EXIT:   (A) = PATTERN IF FOUND
*               'Z' SET IF FOUND
*               'Z' CLEAR IF NOT FOUND OR PATTERN=0
*       USES:   A,F,H,L

$TBLS   EQU     31111A  1949H   6473
.....................................................................

***     $TYPTX - TYPE TEXT.
*
*       $TYPTX IS CALLED TO TYPE A BLOCK OF TEXT ON THE SYSTEM CONSOLE.
*
*       A BYTE WITH THE 200Q BIT SET IS THE LAST BYTE IN THE MESSAGE.
*
*       ENTRY:  (RET) = TEXT
*       EXIT:   TO (RET+LENGTH)
*       USES:   A,F

$TYPTX  EQU     31136A  195EH   6494

*       ENTRY:  (HL) = TEST
*       EXIT:   (HL) = (HL) + LENGTH
*       USES:   A,F,H,L

$TYPTX. EQU     31144A  1964H   6500
.....................................................................

***     $UDD - UNPACK DECIMAL DIGITS.
*
*       UDD CONVERTS A 16 BIT VALUE INTO A SPECIFIED NUMBER OF
*       DECIMAL DIGITS. THE RESULT IS ZERO FILLED.
*
*       ENTRY:  (B,C) = ADDRESS VALUE
*               (A) = DIGIT COUNT
*               (H,L) = MEMORY ADDRESS
*       EXIT:   (HL) = (HL) + (A)
*       USES:   ALL

$UDD    EQU     31157A  196FH   6511
.....................................................................
```

APPENDIX 13-F: ROMCALLS (Cont)
++++++++++++++++++++++++++++++

```
***     $ZERO - ZERO MEMORY
*
*       $ZERO ZEROS A BLOCK OF MEMORY.
*
*       ENTRY:  (HL) = ADDRESS
*               (B) = COUNT
*       EXIT:   (A) = 0
*       USES:   A,B,F,H,L

$ZERO   EQU     31212A  198AH   6538
.....................................................................

***     $FILL - FILL MEMORY
*
*       $FILL FILLS A BLOCK OF MEMORY.
*
*       ENTRY:  (HL) = ADDRESS
*               (B) = COUNT
*               (A) = FILL BYTE
*       EXIT:   (A) = 0
*       USES:   A,B,F,H,L

$FILL   EQU     31213A  198BH   6539


***     8 CONSTANT ZERO BYTES.

$ZEROS  EQU     31320A  19D0H   6608
.....................................................................

***     $REL - RELOCATE CODE.
*
*       REL PROCESSES A RELOCATION LIST.
*
*       ENTRY:  (BC) = DISPLACEMENT FROM ASSEMBLED ADDRESS
*                      & RELOCATION FACTOR (FROM CURRENT ADDRESS)
*               (HL) = FWA RELOCATION LIST
*       EXIT:   NONE
*       USES:   ALL
*
*       (DE) WILL BE EQUAL TO (BC), CALL $REL.

$REL.   EQU     33175A  1B7DH   7037

*       ENTRY:  (BC) = DISPLACEMENT FROM ASSEMBLED ADDRESS
*               (DE) = RELOCATION FACTOR (FROM CURRENT ADDRESS)
*               (HL) = FWA RELOCATION LIST
```

APPENDIX 12-F: ROMCALLS (Cont)

$REL (Cont)

```
*       EXIT:   NONE
*       USES:   ALL

$REL    EQU     33177A  1B7FH   7039
```

SUMMARY OF H17 ROM SUBROUTINE ADDRESSES

Name		S/OCT	HEX	DEC	Description
$COMP	EQU	30060A	1830H	6192	String Compare
$DADA	EQU	30072A	183AH	6202	Add A to HL
$DADA.	EQU	30101A	1841H	6209	Add A to HL
$DU66	EQU	30106A	1846H	6214	16 bit / 16 bit
$HLIHL	EQU	30211A	1889H	6281	Indirect Load HL Thru HL
$CDEHL	EQU	30216A	188EH	6286	Compare DE to HL
$CHL	EQU	30224A	1894H	6292	Compliment HL
$INDL	EQU	30234A	189CH	6300	Indexed Load
$MOVE	EQU	30252A	18AAH	6314	Move Block of Memory
$MU10	EQU	30324A	18D4H	6356	Multiply by 10
$MU66	EQU	30337A	18DFH	6367	16 bit X 16 bit
$MU86	EQU	31007A	1907H	6407	8 bit X 16 bit
$RSTALL	EQU	31047A	1927H	6439	Restore All Registers
$SAVALL	EQU	31054A	192CH	6444	Save All Registers
$TJMP	EQU	31061A	1931H	6449	Table Jump
$TJMP.	EQU	31062A	1932H	6450	Table Jump
$TBRA	EQU	31076A	193EH	6462	Table Branch
$TBLS	EQU	31111A	1949H	6473	Table Search
$TYPTX	EQU	31136A	195EH	6494	Type Text

APPENDIX 13-F: ROMCALLS (Cont)

SUMMARY OF H17 ROM SUBROUTINE ADDRESSES Cont)

$TYPTX.	EQU	31144A	1964H	6500	Type Text
$UDD	EQU	31157A	196FH	6511	Unpack Decimal Digits
$ZERO	EQU	31212A	198AH	6538	Zero Block of Memory
$FILL	EQU	31213A	198BH	6539	Fill Block of Memory
$ZEROS	EQU	31320A	19D0H	6608	8 Constant Zeros
$REL.	EQU	33175A	1B7DH	7037	Relocate Code
$REL	EQU	33177A	1B7FH	7039	Relocate Code

INDEX

NOTE: THIS INDEX IS NOT YET DONE. NOTICE ALL THE REFERENCES TO CHAPTER 7. IT WAS CHAPTER 7 FOR HDOS VERSION 2.0.

INDEX (Cont)

INDEX (Cont)

INDEX (Cont)

INDEX (Cont)

INDEX (Cont)

INDEX (Cont)

****

Last Edited: 11-Apr-90

HDOS SOFTWARE REFERENCE MANUAL

HDOS DISK OPERATING SYSTEM

VERSION 3.02

CHAPTER 14

DATA BITS

HEATH DISK OPERATING SYSTEM

SOFTWARE REFERENCE MANUAL

VERSION 3.02

 Through the years it continued to be improved by successive revisions which included 1.5, 1.6, and finally 2.0. It was entered into public domain on 19 July 1989 per letter by Jim Buszkiewicz, Managing Editor, Heath Users' Group, P.O. Box 217, Benton Harbor, MI 49022-0217 (616)982-3463. A copy of this letter is available for public inspection. Indeed, HDOS is still alive and well!

This manual is indicative of further improvements and provides for the latest revision, HDOS 3.0 and HDOS 3.02. Revision 3.0 is detailed in chapters 1, 2, and 3, while chapters 4, 5, 6, 7 and 8, 13 and 14, are related to revision 3.02. Chapters 9 through 12, with minor improvements, are essentially picked up from the original HDOS 2.0 manual.

Chapter 14, Data Bits, is a mixed collection of various bits of datum that are worthy of being contained in the HDOS 3.02 manual. It will give the reader some interesting background information that couldn't be obtained from any other source.

SPECIAL DISCLAIMER: The Heath Company cannot provide consultation on either the HDOS Operating System or user-developed or modified versions of Heath software products designed to operate under the HDOS Operating System. Do not refer to Heath for questions.

Instead, you are invited to direct any questions concerning the Heath Disk Operating System (HDOS) to Mr. Kirk L. Thompson, Editor "Staunch 89/8" Newsletter, P.O. Box 548, #6 West Branch Mobile Home Village, West Branch, IA 52358.

TABLE OF CONTENTS

INTRODUCTION

This chapter is provided as a convenience to furnish reference data which is quickly available. For instance, if you desire to scan 'New Commands,' 'New Batch Commands,' or wants to find the files on 'PIP Capsule Review' before using PIP to resolve a specific problem, the TABLE OF CONTENTS can quickly lead you to the appropriate page.

Also, since the 'Filelist for HDOS 3.0a' contains a list of all of the original source code files you don't have to turn on your computer to locate specific files, or sort thru a pile of disks to find the files you want to check. To shorten the time required to perform this task, first go to the Table of Contents, determine which disk is likely to contain your file, and turn to the appropriate page(s).

Further, if you desire to learn more details about the HDOS 3.02 mod, all that is necessary is to refer to 'Disk Contents,' 'Notes' section, and you will find all the data is laid out before you when you turn to the page(s) of interest.

Finally, the data is available in transportable sections on the disk. If you want to make notes, all you have to do is to print the file that interests you, and you don't need to format it first, as you would have to do if the original copy of the file resided on disk. Therefore, this chapter provides convenience and saves time for the user.

It also will be a help to those who do not yet have a printer, since all the data is already printed.

WHAT'S NEW IN HDOS 3.02 ?

A capsule review of the differences between SYSCMD 3.0a and 3.02.

BIT has been enhanced.
CLR has been changed to CFLAGS with no arguments.
CLS has been enhanced.
COPY has been enhanced; see discussion in PIP section below.
COUNT has been enhanced.
DMM has been enhanced.
END has been enhanced.
FLAGS has been changed to SFLAGS and CFLAGS with arguments.
LOADF has been changed to FLOAD.
IF KEY has been enhanced.
SI has been enhanced.
WAIT has been enhanced.

New Commands	Meaning
ALT	Show alternate device name
ALT DVn:	Set alternate device name
ALT :	Set alternate device name to default name
BAT[CH] fname [args]	Bypass .ABS link & try to run BATCH file
CF[LAGS] file(s)	Clear all flags on specified file(s)
CF[LAGS] file(s)=flags	Clear flags on specified file(s)
CLS	Clear console screen (reset graphics, reverse, 25th line)
CLS <any arg>	Reset graphics, reverse, 25th line
DEF[AULT] ~	Set system default to all nulls
D	; Dismount primary device unit 0
FLO[AD] xx[:]	Same as LOAD plus Fix in memory
HA[LT]	Try SHUTDOWN.ABS(.BAT) then exit HDOS
M	; Mount primary device unit 0
MOV[E] dest=source	Copy file(s), verify, delete source file(s)
Pn	Set current list device unit to #n. n=0..7
PRN	Show current list device name & unit
PRN DVn:	Set current list device name to xx (unit 0)
PU[SER] file(s)=users	Put specified file(s) in specified user areas
QD	; Quiet Dismount (All available units of default device)
QD SY:, DK:, Etc	; Quiet Dismount (All available units of xx:)
QM	; Quiet Mount (All available units of default device)
QM SY:, DK:, Etc	; Quiet Mount (All available units of xx:)
R	; Reset primary device unit 0
RU[SER] file(s)	Remove specified file(s) from all active user areas
RU[SER] file(s)=users	Remove specified file(s) from specified user areas
SF[LAGS] file(s)=flags	Set specified flags on specified file(s)

WHAT'S NEW (Cont)

A capsule review of differences between SYSCMD 3.0a and 3.02. (Cont)

New Commands	Meaning
U[SER]	Show active user area
U[SER] n	Set active user area to #n. n=0..7
Un	Set active user area to #n. n=0..7
XYZ[ZY]	Toggle display of exit codes upon return to SYSCMD
XYZ[ZY] <any arg>	Toggle display of PIP command syntax within SYSCMD

New BATCH Commands	Meaning
BIT	Show BIT values
BIT T	Toggle all BIT flags
BIT T digit	Toggle specific BIT flag (0..7)
CB[UF]	Clear console buffer
COU[NT]	Show system counter value
END	Exit BATCH file (usually before physical end)
END C	Exit BATCH file & clear console screen & modes
END <any arg>	Exit BATCH file & clear ONLY console modes
IF [NOT] KEY = value command	Test ASK or TRAP keystroke value
KEY	Show current ASK keystroke value
KEY alpha	Preset ASK keystroke
KEY ?<cr|space|tab|?>	Preset special value. CR = null
' [text]	Remark, do nothing
TR[AP]	Grab keystroke on the fly & save it
WAIT	Wait indefinitely for user to touch any key

New special replaceable parameters:

%# = active user area (0)
%p = active LP unit (0)
%k = ASK keystroke

New special characters:

$@ = the NULL char
$< = the BACKSPACE char
$# = active user area (0)
$p = active LP unit (0) Note: The old "$p" has changed to "$>"
$k = the ASK keystroke
$> = default system prompt

WHAT'S NEW (Cont)

A capsule review of differences between SYSCMD 3.0a and 3.02. (Cont)

'&' has been added to the possible flags. It equals 'SLWD.'
't' has been added to the possible sort fields. It is time ascending.
'tr' has been added to the possible sort fields. It is time descending.

When copying files to a different disk, if the destination disk is filled or there is not enough room left on it to copy the next file in your list, PIP will give you the opportunity to reset the destination drive and insert another disk.

****

A capsule review of the differences between PIP 3.0a and 3.02

/SUPRESS has been enhanced with subswitches.
/PAGE has been enhanced to paginate DIR listings to the console.
/FULL header has been rearranged.

New Verb Switches	Meaning
/NOP	Do absolutely nothing
/PUT[USER]:f..	Put file in user areas
/PUT[USER]:f.!.	Put in these user areas & Remove from others except 0
/PUT[USER]:*	Put file in all user areas
/REM[USER]	Remove file from all user areas except 0
/REM[USER]:f..	Remove file from specified user areas ('0' invalid)
/REM[USER]:*	Remove file from all user areas except 0
/USR	Set active user area to 0
/USR:n	Set active user area to #n. n=0..7
/TAB[LE]	Build source file list
/W[IDE]	Same as /B

New Modifier Switches	Meaning
/.	Override automatic setting of /US:<Active user area>
/CLS	Clear console screen on H19
/DSF	Delete source file after verifying destination
/HOLD	Set Hold Screen mode on H19
/NOU[SER]	Include files ONLY in user area 0
/NOU[SER]:u..	Include files NOT in specified user areas
/P[AGE]	Paginate directory listings sent to console
/SO[RT]:t[r]	Sort files for DEST usage t = creation Time ascending tr = reverse sort

WHAT'S NEW (Cont)

A capsule review of the differences between PIP 3.0a and 3.02: (Cont)

New Modifier Switches	Meaning
/SU[PRESS]	Supress trailing message & files selected count
/SU[PRESS]	[a][h][t][s][c][*] Supress selected item(s) a = audit trail s = status (25th) line h = header lines c = files selected count t = trailing messa * = all possible items
/T[ODAY]	Include files created today
/UA[REAS]	Set DEST file user areas to SOURCE file user areas
/UA[REAS]:u..	Set DEST file user areas
/US[ER]:u..	Include files in specified user areas

FILELIST FOR HDOS 3.0A

The following are /FULL listings of the seven (7) single-sided hard-sector distribution disks and the four (4) source disks for HDOS 3.0, Revision A.

Volume: 0 on 11-Aug-88 Type: System Init Date: 14-Dec-86
Label: HDOS 3.0, Issue #50.07.00 [System Distribution]

Name	.Ext	Size	Alloc	Created	Time	Flags---	Accessed	A/C
HDOS30	.SYS	40	40	4-Oct-86	3:16a	SLWC	D	
TT	.DVD	13	14	5-Oct-86	5:28p	SL C	D	
SYSCMD	.SYS	38	38	4-Oct-86	6:25p	SLWC	D	
PIP	.ABS	42	42	4-Oct-86	6:29p	SLWC	D	
SY	.DVD	18	18	5-Oct-86	5:32p	SL C	D	
ERRORMSG	.SYS	8	8	10-Aug-86	11:15a	SLWC	D	
SET	.ABS	8	8	20-Sep-86	10:12p	WC	D	
SYSHELP	.DOC	25	26	19-Oct-86	8:52p	SLWC	D	
HELP	.	12	12	19-Oct-86	8:56p	SLWC	D	
INIT	.ABS	29	30	5-Oct-86	5:00p	WC	D	
SYSGEN	.ABS	20	20	5-Oct-86	5:12p	WC	D	
MAP	.ABS	8	8	18-Oct-86	11:35p	WC	D	
SYSHELP	.H19	26	26	21-Sep-86	3:30p	WC	D	
ONECOPY	.ABS	21	22	13-Oct-86	11:00p	WC	D	
WHAT	.ABS	16	16	5-Oct-86	3:10p	WC	D	
SYS	.ABS	1	2	7-Aug-86	11:15a	WC	D	
HELP	.H19	13	14	14-Sep-86	1:52p	WC	D	
MAKMSD	.ABS	3	4	5-Oct-86	4:09p	WC	D	
EDIT	.ABS	16	16	5-Oct-86	4:42p	WC	D	
RGT	.SYS	1	2	14-Dec-86	5:29p	SLWC	D	
GRT	.SYS	1	2	14-Dec-86	5:29p	SLWC	D	
DIRECT	.SYS	18	18	14-Dec-86	5:29p	SLWC	D	

22 Files, Using 377 Sectors (386 Allocated, 4 Free, 1.0 % Free)

FILELIST FOR HDOS 3.0a (Cont)

Volume: 1 on 11-Aug-88 Type: Data Init Date: 14-Nov-86
Label: HDOS 3.0, Issue #50.07.00 [Utilities and Drivers]

Name	.Ext	Size	Alloc	Created	Time	Flags---	Accessed	A/C
BASIC	.ABS	41	42	6-Aug-86	9:39p	WC	D	
PATCH	.ABS	10	10	6-Aug-86	9:39p	WC	D	
ASM	.ABS	32	32	6-Aug-86	9:40p	WC	D	
XREF	.ABS	12	12	6-Aug-86	9:40p	WC	D	
TT	.DVD	13	14	5-Oct-86	5:28p	WC	D	
ND	.DVD	3	4	5-Oct-86	3:23p	WC	D	
H17	.DVD	18	18	5-Oct-86	5:32p	WC	D	
H37	.DVD	20	20	5-Oct-86	5:36p	WC	D	
H47	.DVD	13	14	5-Oct-86	3:39p	WC	D	
AT84	.DVD	5	6	6-Oct-86	10:07p	WC	D	
AT85	.DVD	5	6	6-Oct-86	10:09p	WC	D	
H1484	.DVD	6	6	7-Oct-86	9:44p	WC	D	
H1485	.DVD	6	6	7-Oct-86	9:51p	WC	D	
H2484	.DVD	6	6	7-Oct-86	10:15p	WC	D	
H2584	.DVD	10	10	5-Oct-86	5:30p	WC	D	
H4484	.DVD	8	8	9-Oct-86	8:40p	WC	D	
MX8084	.DVD	8	8	10-Oct-86	12:58a	WC	D	
MX8011	.DVD	8	8	10-Oct-86	12:56a	WC	D	
IOMEGA	.DVD	10	10	5-Oct-86	5:39p	WC	D	
README	.DOC	35	36	14-Nov-86	2:51a	WC	D	
CLOCK89	.TAS	3	4	5-Oct-86	4:37p	WC	D	
CLOCK89	.H8A	22	22	5-Oct-86	4:36p	WC	D	
CLOCK	.TAS	3	4	5-Oct-86	5:45p	WC	D	
CLOCK	.H8A	19	20	5-Oct-86	5:45p	WC	D	
RGT	.SYS	1	2	14-Nov-86	4:18p	SLWC	D	
GRT	.SYS	1	2	14-Nov-86	4:18p	SLWC	D	
DIRECT	.SYS	18	18	14-Nov-86	4:18p	SLWC	D	

27 Files, Using 336 Sectors (348 Allocated, 42 Free, 10.5 % Free)

Volume: 2 on 11-Aug-88 Type: Data Init Date: 14-Nov-86
Label: HDOS 3.0, Issue #50.07.00 [Driver Source 1]

Name	.Ext	Size	Alloc	Created	Time	Flags---	Accessed	A/C
H17DVD	.H8A	91	92	5-Oct-86	5:24p	WC	D	
H17INIT	.H8A	81	82	5-Oct-86	5:24p	WC	D	
H17ABT	.ACM	3	4	10-Aug-86	11:23a	WC	D	
H17CLK	.ACM	3	4	10-Aug-86	11:23a	WC	D	
H17LOA	.ACM	13	14	13-Aug-86	11:23a	WC	D	
H17MOU	.ACM	16	16	21-Sep-86	2:36p	WC	D	
H17RDY	.ACM	5	6	10-Aug-86	11:23a	WC	D	
H17REA	.ACM	14	14	10-Aug-86	11:23a	WC	D	
H17RER	.ACM	2	2	13-Aug-86	11:23a	WC	D	
H17ROD	.ACM	31	32	10-Aug-86	11:23a	WC	D	
H17SET	.ACM	14	14	20-Sep-86	9:59p	WC	D	

FILELIST FOR HDOS 3.0a Cont)

Volume: 2 on 11-Aug-88 Type: Data Init Date: 14-Nov-86
Label: HDOS 3.0, Issue #50.07.00 [Driver Source 1]

Name	.Ext	Size	Alloc	Created	Time	Flags---	Accessed	A/C
H17SET2	.ACM	2	2	10-Aug-86	11:23a	WC	D	
H17UNL	.ACM	3	4	10-Aug-86	11:23a	WC	D	
H17WRI	.ACM	13	14	10-Aug-86	11:23a	WC	D	
H17SKEW	.MBA	2	2	25-Nov-81	11:23a	WC	D	
NDDVD	.H8A	12	12	5-Oct-86	2:55p	WC	D	
ATDVD	.H8A	39	40	6-Oct-86	10:08p	WC	D	
MAKE	.BAT	1	2	13-Aug-86	11:17a	WC	D	
MAKEDVD	.BAT	1	2	1-Sep-86	12:56p	WC	D	
MAKEDVD2	.BAT	1	2	31-Aug-86	11:18a	WC	D	
RGT	.SYS	1	2	14-Nov-86	4:18p	SLWC	D	
GRT	.SYS	1	2	14-Nov-86	4:18p	SLWC	D	
DIRECT	.SYS	18	18	14-Nov-86	4:18p	SLWC	D	

23 Files, Using 367 Sectors (382 Allocated, 8 Free, 2.0 % Free)

Volume: 3 on 11-Aug-88 Type: Data Init Date: 14-Nov-86
Label: HDOS 3.0, Issue #50.07.0 [Driver Source 2]

Name	.Ext	Size	Alloc	Created	Time	Flags---	Accessed	A/C
H37DVD	.H8A	53	54	5-Oct-86	5:25p	WC	D	
H37INIT	.H8A	93	94	5-Oct-86	5:25p	WC	D	
H37LIB	.ACM	87	88	13-Aug-86	11:24a	WC	D	
IODVD	.H8A	42	42	5-Oct-86	5:25p	WC	D	
IOINIT	.H8A	22	22	5-Oct-86	5:25p	WC	D	
IODEF	.ACM	14	14	15-Mar-85	11:24a	WC	D	
IOSUBS	.ACM	21	22	19-Aug-86	11:24a	WC	D	
RGT	.SYS	1	2	14-Nov-86	4:19p	SLWC	D	
GRT	.SYS	1	2	14-Nov-86	4:19p	SLWC	D	
DIRECT	.SYS	18	18	14-Nov-86	4:19p	SLWC	D	

10 Files, Using 352 Sectors (358 Allocated, 32 Free, 8.0 % Free)

Volume: 4 on 11-Aug-88 Type: Data Init Date: 14-Nov-86
Label: HDOS 3.0, Issue #50.07.00 [Driver Source 3]

Name	.Ext	Size	Alloc	Created	Time	Flags---	Accessed	A/C
H47DVD	.H8A	34	34	5-Oct-86	2:50p	WC	D	
H47INIT	.H8A	64	64	5-Oct-86	2:51p	WC	D	
H47LIB	.ACM	56	56	14-Aug-86	11:24a	WC	D	
TTDVD	.H8A	103	104	5-Oct-86	5:24p	WC	D	
H14DVD	.H8A	66	66	7-Oct-86	9:50p	WC	D	
RGT	.SYS	1	2	14-Nov-86	4:20p	SLWC	D	

FILELIST FOR HDOS 3.0a Cont)

[Continued]

Volume: 4 on 11-Aug-88 Type: Data Init Date: 14-Nov-86
Label: HDOS 3.0, Issue #50.07.00 [Driver Source 3]

Name	.Ext	Size	Alloc	Created	Time	Flags---	Accessed	A/C
GRT	.SYS	1	2	14-Nov-86	4:20p	SLWC	D	
DIRECT	.SYS	18	18	14-Nov-86	4:20p	SLWC	D	

8 Files, Using 343 Sectors (346 Allocated, 44 Free, 11.0 % Free)

Volume: 5 on 11-Aug-88 Type: Data Init Date: 14-Nov-86
Label: HDOS 3.0, Issue #50.07.00 [Driver Source 4]

Name	.Ext	Size	Alloc	Created	Time	Flags---	Accessed	A/C
H24DVD	.H8A	51	52	7-Oct-86	10:14p	WC	D	
H25DVD	.H8A	59	60	5-Oct-86	5:24p	WC	D	
H44DVD	.H8A	54	54	9-Oct-86	8:39p	WC	D	
MX80DVD	.H8A	61	62	10-Oct-86	12:58a	WC	D	
MAKMSD	.H8A	11	12	5-Oct-86	4:08p	WC	D	
SET	.H8A	81	82	5-Oct-86	4:12p	WC	D	
SYS	.H8A	11	12	7-Aug-86	11:25a	WC	D	
RGT	.SYS	1	2	14-Nov-86	4:21p	SLWC	D	
GRT	.SYS	1	2	14-Nov-86	4:21p	SLWC	D	
DIRECT	.SYS	18	18	14-Nov-86	4:21p	SLWC	D	

10 Files, Using 348 Sectors (356 Allocated, 34 Free, 8.5 % Free)

Volume: 6 on 11-Aug-88 Type: Data Init Date: 14-Nov-86
Label: HDOS 3.0, Issue #50.07.00 [Common Decks 1]

Name	.Ext	Size	Alloc	Created	Time	Flags---	Accessed	A/C
ABSDEF	.ACM	1	2	15-Mar-85	11:19a	WC	D	
ASCII	.ACM	4	4	22-Sep-86	8:25p	WC	D	
BITC	.ACM	2	2	15-Mar-85	11:21a	WC	D	
BITS	.ACM	2	2	2-Mar-86	12:00a	WC	D	
BOODEF	.ACM	3	4	2-Aug-86	11:20a	WC	D	
CDEHL	.ACM	1	2	15-Mar-85	11:19a	WC	D	
CHL	.ACM	1	2	15-Mar-85	11:19a	WC	D	
CPA	.ACM	6	6	27-Jul-86	11:22a	WC	D	
BPDEHL	.ACM	1	2	21-Sep-86	9:22p	WC	D	
CVD	.ACM	2	2	21-Sep-86	11:17p	WC	D	
DADA	.ACM	1	2	15-Mar-85	11:19a	WC	D	
DADA2	.ACM	1	2	15-Mar-85	11:19a	WC	D	
DDD	.ACM	3	4	21-Sep-86	10:28p	WC	D	
DDDEF	.ACM	3	4	21-Sep-86	10:30p	WC	D	
DDFDEF	.ACM	1	2	21-Sep-86	10:31p	WC	D	
DDS	.ACM	5	6	15-Mar-85	11:21a	WC	D	

FILELIST FOR HDOS 3.0a Cont)

[Continued]

Volume: 6 on 11-Aug-88 Type: Data Init Date: 14-Nov-86
Label: HDOS 3.0, Issue #50.07.00 [Common Decks 1]

Name	.Ext	Size	Alloc	Created	Time	Flags---	Accessed	A/C
DEVDEF	.ACM	7	8	17-Aug-86	11:20a	WC	D	
DIRDEF	.ACM	3	4	19-Mar-85	11:19a	WC	D	
DNV	.ACM	7	8	21-Sep-86	11:20p	WC	D	
DU66	.ACM	1	2	15-Mar-85	11:19a	WC	D	
DVDDEF	.ACM	3	4	25-Mar-85	11:19a	WC	D	
DVDIO	.ACM	13	14	6-Oct-86	9:57p	WC	D	
DVDIO2	.ACM	5	6	9-Oct-86	10:13p	WC	D	
ECDEF	.ACM	9	10	10-Aug-86	11:22a	WC	D	
ECVEC	.ACM	2	2	17-Jul-86	11:20a	WC	D	
EDCON	.ACM	2	2	21-Sep-86	10:36p	WC	D	
EDRAM	.ACM	4	4	21-Sep-86	10:38p	WC	D	
EDVEC	.ACM	4	4	10-Aug-86	11:22a	WC	D	
ESINT	.ACM	13	14	20-Aug-86	11:23a	WC	D	
ESVAL	.ACM	8	8	27-Jul-86	11:19a	WC	D	
FILDEF	.ACM	1	2	21-Sep-86	10:40p	WC	D	
FLTDEF	.ACM	2	2	7-Aug-86	11:22a	WC	D	
FST	.ACM	6	6	22-Sep-86	12:42a	WC	D	
H14	.ACM	1	2	7-Oct-86	9:32p	WC	D	
H17DEF	.ACM	5	6	15-Mar-85	11:20a	WC	D	
H17ROM	.ACM	3	4	10-Dec-81	11:19a	WC	D	
H37DEF	.ACM	10	10	10-Aug-86	11:22a	WC	D	
H47DEF	.ACM	12	12	6-Aug-86	11:22a	WC	D	
H47PAR	.ACM	1	2	6-Aug-86	11:22a	WC	D	
HDSROM	.ACM	3	4	9-Aug-86	11:21a	WC	D	
RGT	.SYS	1	2	14-Nov-86	4:21p	SLWC	D	
GRT	.SYS	1	2	14-Nov-86	4:21p	SLWC	D	
DIRECT	.SYS	18	18	14-Nov-86	4:21p	SLWC	D	
HLIHL	.ACM	1	2	15-Mar-85	11:20a	WC	D	
HOSBASE	.ACM	10	10	14-Sep-86	4:45p	WC	D	
HOSDEF	.ACM	7	8	1-Sep-86	8:45p	WC	D	
HOSEQU	.ACM	3	4	9-Aug-86	11:19a	WC	D	
HROM	.ACM	5	6	10-Aug-86	11:21a	WC	D	
INDL	.ACM	2	2	15-Mar-85	11:20a	WC	D	
INIDEF	.ACM	4	4	11-Aug-86	11:21a	WC	D	
IOCDEF	.ACM	5	6	15-Mar-85	11:20a	WC	D	
ITL	.ACM	2	2	15-Mar-85	11:21a	WC	D	
LABDEF	.ACM	5	6	9-Aug-86	11:19a	WC	D	
LBD	.ACM	5	6	12-Aug-86	11:23a	WC	D	
MCU	.ACM	2	2	21-Sep-86	10:48p	WC	D	
MLU	.ACM	2	2	21-Sep-86	10:49p	WC	D	
MOVE	.ACM	3	4	15-Mar-85	11:20a	WC	D	
MTR	.ACM	8	8	19-Aug-86	11:23a	WC	D	
MTRDEF	.ACM	2	2	27-Jul-86	11:22a	WC	D	
MTRRAM	.ACM	7	8	17-Jul-86	11:22a	WC	D	

FILELIST FOR HDOS 3.0a Cont)

[Continued]

Volume: 6 on 11-Aug-88 Type: Data Init Date: 14-Nov-86
Label: HDOS 3.0, Issue #50.07.00 [Common Decks 1]

Name	.Ext	Size	Alloc	Created	Time	Flags---	Accessed	A/C
MU86	.ACM	1	2	15-Mar-85	11:20a	UC	D	
PBF	.ACM	3	4	15-Feb-85	11:23a	WC	D	
PBV	.ACM	4	4	15-Feb-85	11:23a	WC	D	
PICDEF	.ACM	1	2	15-Mar-85	11:20a	WC	D	
RCHAR	.ACM	1	2	15-Mar-85	11:20a	WC	D	
RTL	.ACM	4	4	21-Sep-86	10:57p	WC	D	
SAVALL	.ACM	3	4	15-Mar-85	11:20a	WC	D	
SETCAL	.ACM	3	4	20-Sep-86	9:56p	WC	D	
SOB	.ACM	2	2	21-Sep-86	11:03p	WC	D	
SOP	.ACM	5	6	11-Aug-86	11:23a	WC	D	
TBLS	.ACM	3	4	15-Mar-85	11:20a	WC	D	
TBRA	.ACM	2	2	15-Mar-85	11:20a	WC	D	
TDD	.ACM	3	4	15-Mar-85	11:21a	WC	D	
THD	.ACM	2	2	15-Feb-85	11:23a	WC	D	
TJMP	.ACM	2	2	15-Mar-85	11:20a	WC	D	
TOD	.ACM	2	2	15-Feb-85	11:23a	WC	D	
TRACE	.ACM	1	2	15-Mar-85	11:20a	WC	D	
TYPTX	.ACM	2	2	15-Mar-85	11:20a	WC	D	
U8250	.ACM	9	10	15-Mar-85	11:20a	WC	D	
U8251	.ACM	5	6	15-Mar-85	11:20a	WC	D	
U8255	.ACM	6	6	10-Oct-86	12:28a	WC	D	
UDD	.ACM	2	2	15-Mar-85	11:20a	WC	D	
UOW	.ACM	4	4	19-Aug-86	11:23a	WC	D	
WTBLS	.ACM	3	4	15-Feb-85	11:23a	WC	D	
ZERO	.ACM	1	2	15-Mar-85	11:20a	WC	D	
ZEROS	.ACM	1	2	15-Mar-85	11:20a	WC	D	

86 Files, Using 330 Sectors (382 Allocated, 8 Free, 2.0 % Free)

Volume: 0 on 11-Aug-88 Type: Data Init Date: 25-Jun-87
Label: HDOS 3.0, Issue #50.07.00 [System Source 1]

Name	.Ext	Size	Alloc	Created	Time	Flags---	Accessed	A/C
HOS3	.H8A	388	400	4-Oct-86	3:13a	LWC	D	
SYSCMD	.H8A	94	96	4-Oct-86	5:35p	LWC	D	
PIP	.H8A	53	64	4-Oct-86	5:21p	LWC	D	
SYSGEN	.H8A	212	224	5-Oct-86	5:04p	LWC	D	
INIT	.H8A	215	224	5-Oct-86	4:58p	LWC	D	
ONECOPY	.H8A	238	240	13-Oct-86	10:59p	LWC	D	
RGT	.SYS	1	16	25-Jun-87	6:03p	SLWC	D	
GRT	.SYS	1	16	25-Jun-87	6:03p	SLWC	D	
DIRECT	.SYS	32	32	25-Jun-87	6:03p	SLW	D	

9 Files, Using 1234 Sectors (1312 Allocated, 2656 Free, 66.4 % Free)

FILELIST FOR HDOS 3.0a Cont)

Volume: 0 on 11-Aug-88 Type: Data Init Date: 25-Jun-87
Label: HDOS 3.0, Issue #50.07.00 [System Source 2]

Name	.Ext	Size	Alloc	Created	Time	Flags---	Accessed	A/C
EDIT	.H8A	163	176	5-Oct-86	4:40p	LWC	D	
LABEL	.H8A	20	32	11-Jul-86	11:25a	LWC	D	
FIX	.H8A	26	32	19-Oct-86	8:14p	LWC	D	
MAP	.H8A	31	32	18-Oct-86	11:35p	LWC	D	
WHAT	.C	10	16	5-Oct-86	3:09p	LWC	D	
CREDITS	.H8A	4	16	10-Aug-86	11:17a	LWC	D	
TITLES	.ACM	17	32	13-Aug-86	11:15a	LWC	D	
RGT	.SYS	1	16	25-Jun-87	6:05p	SLWC	D	
GRT	.SYS	1	16	25-Jun-87	6:05p	SLWC	D	
DIRECT	.SYS	32	32	25-Jun-87	6:05p	SLW	D	

10 Files, Using 305 Sectors (400 Allocated, 3568 Free, 89.2 % Free)

Volume: 0 on 11-Aug-88 Type: Data Init Date: 25-Jun-87
Label: HDOS 3.0, Issue #50.07.00 [Common Decks 2]

Name	.Ext	Size	Alloc	Created	Time	Flags---	Accessed	A/C
ABR	.ACM	16	16	2-Aug-86	11:19a	LWC	D	
AGT	.ACM	9	16	25-Mar-85	11:21a	LWC	D	
ALP	.ACM	2	16	15-Mar-85	11:21a	LWC	D	
BSXDEF	.ACM	2	16	14-Sep-86	1:35p	LWC	D	
CAB	.ACM	2	16	15-Mar-85	11:21a	LWC	D	
CAC	.ACM	2	16	25-Mar-85	11:21a	LWC	D	
CAD	.ACM	14	16	12-Jul-86	12:00a	LWC	D	
CCO	.ACM	2	16	15-Mar-85	11:19a	LWC	D	
CCT	.ACM	1	16	29-Mar-86	12:00a	LWC	D	
CDM	.ACM	4	16	25-Mar-85	11:21a	LWC	D	
CDS	.ACM	8	16	3-Aug-86	11:21a	LWC	D	
CDT2	.ACM	14	16	13-Aug-86	12:00a	LWC	D	
CDU	.ACM	3	16	25-Mar-85	11:21a	LWC	D	
CFC	.ACM	5	16	25-Mar-85	11:21a	LWC	D	
CFD	.ACM	2	16	21-Sep-86	11:15p	LWC	D	
CFI	.ACM	3	16	26-Mar-85	11:21a	LWC	D	
CFP	.ACM	2	16	25-Mar-85	11:22a	LWC	D	
CLL	.ACM	2	16	14-Sep-86	7:49p	LWC	D	
COF	.ACM	4	16	15-Mar-85	11:21a	LWC	D	
COMP	.ACM	2	16	15-Mar-85	11:19a	LWC	D	
CPDEHL	.ACM	1	16	21-Sep-86	9:22p	LWC	D	
CPF	.ACM	3	16	22-Sep-86	12:41a	LWC	D	
CRLF	.ACM	1	16	15-Mar-85	11:19a	LWC	D	
DAD	.ACM	7	16	3-Oct-86	8:12p	LWC	D	
DCF	.ACM	2	16	17-Aug-86	11:22a	LWC	D	
DDS2	.ACM	5	16	15-Mar-85	11:21a	LWC	D	

FILELIST FOR HDOS 3.0a (Cont)

[Continued]
Volume: 0 on 11-Aug-88 Type: Data Init Date: 25-Jun-87
Label: HDOS 3.0, Issue #50.07.00 [Common Decks 2]

Name	.Ext	Size	Alloc	Created	Time	Flags---	Accessed	A/C
DDS3	.ACM	6	16	17-Jul-81	12:00a	LWC	D	
DFA	.ACM	4	16	25-Mar-85	11:22a	LWC	D	
DFC	.ACM	4	16	28-Mar-85	11:22a	LWC	D	
DFD	.ACM	9	16	17-Aug-86	11:23a	LWC	D	
DIFDEF	.ACM	2	16	15-Mar-85	11:19a	LWC	D	
DISDEF	.ACM	2	16	15-Mar-85	11:19a	LWC	D	
DNS	.ACM	4	16	21-Sep-86	11:18p	LWC	D	
DNT	.ACM	4	16	25-Mar-85	11:22a	LWC	D	
DOS	.ACM	3	16	13-Jul-86	11:23a	LWC	D	
DREAD	.ACM	3	16	15-Mar-85	11:19a	LWC	D	
DRS	.ACM	15	16	21-Sep-86	11:34p	LWC	D	
DTB	.ACM	4	16	21-Sep-86	11:35p	LWC	D	
FBDEF	.ACM	2	16	21-Sep-86	10:39p	LWC	D	
FCC	.ACM	2	16	25-Mar-85	11:22a	LWC	D	
RGT	.SYS	1	16	25-Jun-87	6:07p	SLWC	D	
GRT	.SYS	1	16	25-Jun-87	6:07p	SLWC	D	
DIRECT	.SYS	32	32	25-Jun-87	6:07p	SLW	D	
FCLEAR	.ACM	3	16	15-Mar-85	11:19a	LWC	D	
FCLO	.ACM	6	16	21-Sep-86	11:44p	LWC	D	
FDB	.ACM	2	16	25-Mar-85	11:22a	LWC	D	
FEC	.ACM	2	16	29-Mar-86	12:00a	LWC	D	
FERROR	.ACM	3	16	21-Sep-86	11:51p	LWC	D	
FGC	.ACM	4	16	25-Mar-85	11:22a	LWC	D	
FOE	.ACM	4	16	25-Mar-85	11:22a	LWC	D	
FOPE	.ACM	9	16	21-Sep-86	11:58p	LWC	D	
FREAB	.ACM	9	16	15-Mar-85	11:19a	LWC	D	
FREAL	.ACM	11	16	15-Mar-85	11:19a	LWC	D	
FST2	.ACM	9	16	22-Sep-86	12:42a	LWC	D	
FUTIL	.ACM	8	16	22-Sep-86	12:01a	LWC	D	
FWRIB	.ACM	13	16	22-Sep-86	12:17a	LWC	D	
FWRIL	.ACM	3	16	15-Mar-85	11:19a	LWC	D	
GETLAB	.ACM	2	16	3-Aug-86	11:19a	LWC	D	
GNL	.ACM	2	16	21-Sep-86	10:41p	LWC	D	
GUP	.ACM	2	16	15-Mar-85	11:19a	LWC	D	
H17SUBS	.ACM	24	32	20-Aug-86	12:00a	LWC	D	
BCTT	.ACM	4	16	15-Mar-85	11:21a	LWC	D	
IDN	.ACM	2	16	15-Mar-85	11:21a	LWC	D	
ILDEHL	.ACM	1	16	22-Sep-86	12:49a	LWC	D	
IMM	.ACM	4	16	3-Aug-86	11:21a	LWC	D	
INCHA	.ACM	5	16	14-Sep-86	7:50p	LWC	D	
INDXX	.ACM	7	16	22-Sep-86	12:53a	LWC	D	
ISDEHL	.ACM	2	16	21-Sep-86	10:45p	LWC	D	
LDE	.ACM	6	16	25-Mar-85	11:22a	LWC	D	
LDI	.ACM	9	16	17-Aug-86	11:22a	LWC	D	
LFD	.ACM	3	16	17-Aug-86	11:22a	LWC	D	
LUD	.ACM	3	16	25-Mar-85	11:21a	LWC	D	

FILELIST FOR HDOS 3.0a (Cont)

[Continued]
Volume: 0 on 11-Aug-88 Type: Data Init Date: 25-Jun-87
Label: HDOS 3.0, Issue #50.07.00 [Common Decks 2]

Name	.Ext	Size	Alloc	Created	Time	Flags---	Accessed	A/C
MND	.ACM	9	16	17-Aug-86	11:23a	LWC	D	
MOVEL	.ACM	4	16	21-Sep-86	10:50p	LWC	D	
MOVL	.ACM	6	16	14-Sep-86	7:51p	LWC	D	
MOVLL	.ACM	4	16	14-Sep-86	7:50p	LWC	D	
MU10	.ACM	1	16	15-Mar-85	11:20a	LWC	D	
NAMDEF	.ACM	2	16	14-Jul-86	11:21a	LWC	D	
NREDY	.ACM	2	16	15-Mar-85	11:21a	LWC	D	
OVLDEF	.ACM	2	16	15-Mar-85	11:20a	LWC	D	
PCL	.ACM	4	16	4-Jul-86	12:00a	LWC	D	
PDD	.ACM	3	16	15-Mar-85	11:21a	LWC	D	
PGT	.ACM	16	16	17-Aug-86	11:21a	LWC	D	
RBF	.ACM	2	16	25-Mar-85	11:22a	LWC	D	
RDL	.ACM	5	16	1-Sep-86	9:27p	LWC	D	
READY	.ACM	2	16	15-Mar-85	11:21a	LWC	D	
RTL2	.ACM	5	16	21-Sep-86	11:03p	LWC	D	
RVD	.ACM	3	16	15-Mar-85	11:21a	LWC	D	
SCU	.ACM	5	16	22-Sep-86	1:08a	LWC	D	
SGT	.ACM	3	16	25-Mar-85	11:22a	LWC	D	
TASKDEF	.ACM	43	48	22-Jun-86	12:00a	LWC	D	
TFN	.ACM	2	16	2-Aug-86	11:19a	LWC	D	
TFNS	.ACM	3	16	13-Oct-86	10:23p	LWC	D	
TYPCC	.ACM	2	16	21-Sep-86	11:05p	LWC	D	
TYPCH	.ACM	2	16	15-Mar-85	11:20a	LWC	D	
TYPET	.ACM	8	16	15-Mar-85	11:20a	LWC	D	
TYPLN	.ACM	5	16	15-Mar-85	11:20a	LWC	D	
TYPT2	.ACM	2	16	15-Mar-85	11:20a	LWC	D	
UAD	.ACM	4	16	1-Sep-86	4:27p	LWC	D	
UDDN	.ACM	4	16	15-Mar-85	11:20a	LWC	D	
UDDX	.ACM	4	16	21-Sep-86	11:07p	LWC	D	
UDE	.ACM	4	16	25-Mar-85	11:22a	LWC	D	
UDS	.ACM	2	16	25-Mar-85	11:22a	LWC	D	
UHW	.ACM	3	16	21-Sep-86	11:08p	LWC	D	
UNUM	.ACM	2	16	15-Mar-85	11:21a	LWC	D	
WDO	.ACM	2	16	15-Mar-85	11:22a	LWC	D	
WER	.ACM	2	16	15-Mar-85	11:20a	LWC	D	
XCHGBC	.ACM	2	16	11-Jul-81	12:00a	LWC	D	

108 Files, Using 552 Sectors (1792 Allocated, 2176 Free, 54.4 % Free)

FILELIST FOR HDOS 3.0a (Cont)

Volume: 0 on 11-Aug-88 Type: Data Init Date: 25-Jun-87
Label: HDOS 3.0, Issue #50.07.00 [Common Decks 3]

Name	.Ext	Size	Alloc	Created	Time	Flags---	Accessed	A/C
BATCH	.ACM	63	64	7-Sep-86	12:00a	LWC	D	
BYE	.ACM	4	16	9-Aug-86	12:00a	LWC	D	
CHECK	.ACM	2	16	22-Jun-86	12:00a	LWC	D	
CLS	.ACM	1	16	9-Aug-86	12:00a	LWC	D	
COPY	.ACM	1	16	9-Aug-86	12:00a	LWC	D	
DATE	.ACM	3	16	9-Aug-86	12:00a	LWC	D	
DEFAULT	.ACM	14	16	13-Aug-86	12:00a	LWC	D	
DELETE	.ACM	2	16	22-Jun-86	12:00a	LWC	D	
DEV	.ACM	23	32	17-Sep-86	12:00a	LWC	D	
DFSS	.ACM	7	16	29-May-86	12:00a	LWC	D	
DIR	.ACM	9	16	9-Aug-86	12:00a	LWC	D	
DMM	.ACM	11	16	18-Sep-86	12:00a	LWC	D	
DMMBIG	.ACM	34	48	17-May-86	12:00a	LWC	D	
DSSS	.ACM	7	16	9-Mar-86	12:00a	LWC	D	
EDLINE	.ACM	32	32	1-Sep-86	12:00a	LWC	D	
FLAG	.ACM	3	16	9-Aug-86	12:00a	LWC	D	
H19SUBS	.ACM	4	16	24-Jul-86	12:00a	LWC	D	
HDOS30	.ACM	2	16	14-Sep-86	1:35p	LWC	D	
HELP	.ACM	1	16	13-Aug-86	12:00a	LWC	D	
INDLB	.ACM	3	16	2-Mar-86	12:00a	LWC	D	
LOADD	.ACM	14	16	<No-Date>	12:00a	LWC	D	
LOG	.ACM	3	16	9-Aug-86	12:00a	LWC	D	
MDR	.ACM	23	32	22-Sep-86	8:28p	LWC	D	
PATH	.ACM	4	16	9-Aug-86	12:00a	LWC	D	
PIP	.ACM	8	16	9-Sep-86	12:00a	LWC	D	
PIPCMDS	.ACM	74	80	18-Sep-86	11:43p	LWC	D	
PIPCOPY	.ACM	50	64	20-Sep-86	12:00a	LWC	D	
PIPLIST	.ACM	86	96	14-Sep-86	5:11p	LWC	D	
PIPSUBS	.ACM	84	96	20-Sep-86	10:28p	LWC	D	
PIPSWI	.ACM	54	64	9-Sep-86	12:00a	LWC	D	
PRINT	.ACM	3	16	4-Sep-86	12:00a	LWC	D	
PROMPT	.ACM	3	16	9-Aug-86	12:00a	LWC	D	
PROMSHO	.ACM	9	16	3-Sep-86	12:00a	LWC	D	
PRSCL	.ACM	2	16	4-Aug-86	12:00a	LWC	D	
RENAME	.ACM	2	16	22-Jun-86	12:00a	LWC	D	
RUN	.ACM	2	16	30-Jul-86	12:00a	LWC	D	
RVL	.ACM	5	16	4-Aug-86	12:00a	LWC	D	
SI	.ACM	13	16	11-Sep-86	12:00a	LWC	D	
SORT	.ACM	20	32	2-Mar-86	12:00a	LWC	D	
SSM	.ACM	7	16	4-Aug-86	12:00a	LWC	D	
RGT	.SYS	1	16	25-Jun-87	6:09p	SLWC	D	
GRT	.SYS	1	16	25-Jun-87	6:09p	SLWC	D	
DIRECT	.SYS	32	32	25-Jun-87	6:09p	SLWC	D	
START	.ACM	9	16	4-Oct-86	6:22p	LWC	D	
TIME	.ACM	13	16	7-Sep-86	12:00a	LWC	D	

FILELIST FOR HDOS 3.0a (Cont)

[Continued]

Volume: 0 on 11-Aug-88 Type: Data Init Date: 25-Jun-87
Label: HDOS 3.0, Issue #50.07.00 [Common Decks 3]

Name	.Ext	Size	Alloc	Created	Time	Flags---	Accessed	A/C
TYPE	.ACM	3	16	26-May-86	12:00a	LWC	D	
VERIFY	.ACM	4	16	9-Aug-86	12:00a	LWC	D	
VERSN	.ACM	4	16	3-Sep-86	12:00a	LWC	D	
XYZZY	.ACM	1	16	9-Aug-86	12:00a	LWC	D	

49 Files, Using 760 Sectors (1264 Allocated, 2704 Free, 67.6 % Free)

DISK CONTENTS FOR HDOS 3.0

This file briefly describes the contents of the HDOS 3.0 Distribution Disks. The files included here are the final versions, except where indicated.

Disk	File	Description
2	ASM.ABS	This will not assemble much of HDOS. The Gibson Assembler, available from Quikdata was used to develop HDOS 3.0. The Gibson assembler is about 10 times faster than this one and if you plan on changing HDOS and reassembling, I strongly recommend it. The features I used which are not supported by this assembler are relatively benign and should be easily worked around.
2	AT84.DVD	Device driver for alternate terminal for H8-4 interface.
2	AT85.DVD	Device driver for alternate terminal for H8-5 interface.
2	BASIC.ABS	Benton Harbor BASIC. This has minor changes for HDOS 3.0.
2	CLOCK.TAS	Standard H89 real time clock processor. Type 'START CLOCK<RTN>.' Applies to the typical H89/Z90 computer systems. Place this command in your AUTOEXEC.BAT file.
2	CLOCK89.TAS	Super-89 real time clock processor. Type 'START CLOCK89<RTN>.' Applies to computer systems with the D.G. Super89 CPU Board. If you have one, place this command in your AUTOEXEC.BAT file.
1	EDIT.ABS	Heath's Line Editor crossed over to HDOS 3.0.
1	ERRORMSG.SYS	A listing of HDOS 3.0a/3.02 error messages. This listing has been revised and improved over the listing of HDOS 2.0 and below.
2	H1484.DVD	Device driver for H14 printer with H8-4 interface.
2	H1485.DVD	Device driver for H14 printer with H8-5 interface.
2	H17.DVD	H17 device driver.
2	H2484.DVD	Device driver for H24 printer (TI-810) with H8-4 interface.

DISK CONTENTS FOR HDOS 3.0 (Cont)

Disk	File	Description
2	H2584.DVD	Device driver for H25 printer with H8-4 interface.
2	H37.DVD	H37 device driver.
2	H4484.DVD	Device driver for H44 Diablo printer with H8-4 interface.
2	H47.DVD	H47 device driver.
1	HDOS30.SYS	This is the entire operating system. There are no overlays.
1	HELP	Help for PIP.ABS for the non-H19 terminal.
1	HELP.H19	Help for PIP.ABS for the H19 terminal.
1	SYSHELP.DOC	Help with SYSCMD.SYS for the non-H19 terminal.
1	HELP.H19	Help with SYSCMD.SYS for the H19 terminal.
1	INIT.ABS	Initializes HDOS 3.0 disks.
2	IOMEGA.DVD	Bernoulli Box device driver. (See the source code before attempting to use this driver!!)
1	MAKMSD.ABS	Used to create mass storage (disk) drivers.
1	MAP.ABS	Fun facts.
2	MX8011.DVD	Device driver for Epson MX-80 printer with H8-4 interface.
2	MX8084.DVD	Device driver for Epson MX-80 printer with Z89-11 interface. This is a parallel driver.
2	ND.DVD	Device driver for the null device.
1	ONECOPY.ABS	Copy files with one disk drive.
1	PATCH.ABS SYSPATCH.ABS	For fixing bugs and patching programs.

DISK CONTENTS FOR HDOS 3.0 (Cont)

Disk	File	Description
1	PIP.ABS	Peripheral Interchange Program.
1	SET.ABS	Driver and HDOS SET Utility.
1	SY.DVD	Device driver for your primary system device.
1	SYS.ABS	Sets the system bit in a disk volume label.
1	SYSCMD.SYS	System Command Processor.
1	SYSGEN.ABS	Copies the HDOS 3.0 operating system onto newly-initialized disks.
2	TT.DVD	Device driver for the console. Also processes all terminal-related scalls.
1	WHAT.ABS	Tells what files are. Try "What HDOS30.SYS."
1	XREF.ABS	A program that goes with ASM.ABS. Used in creating machine code files.

****

NOTES

[A] MEMORY MAP

HDOS 3.0 is ORG-0. This does not mean that the program area (USERFWA) is near zero, but rather the system itself, HDOS30.SYS, is loaded in low memory. This buys the user about 4-5K of additional memory for programs. A brief memory map would appear as follows:

	Start	End	Description
	000000	027377	HDOS30.SYS
	030000	033315	H17 ROM Subroutines
*	033316	037377	HDOS buffers and work areas [Note 1]
*	040000	040077	Monitor work cells [Note 2]
	040100	042177	HDOS data area
	042200	S.SYSM	User program area
	S.SYSM	S.RFWA	Loaded (but not locked) drivers
	S.RFWA	S.HIMEM	GRT tables, locked drivers, buffers

Refer to Chapter 8, Appendix 8-A: Memory Layouts - Memory Map, page 8-10 for further detail concerning the memory map for HDOS 3.02.

NOTES (Cont)

NOTES:

(1) The H17 driver code which formerly resided here is GONE, and should not be referenced! If a person calls this code directly, the disk WILL CRASH under this HDOS version!

(2) This is where the PAM-8 or MTR-88/89/90 monitors kept their scratch pad data. Since HDOS runs in low memory, consider it safe to assume that there is no monitor. HDOS, however, uses selected cells in this area in the same fashion as the monitor. Software which references this area should function properly.

Software which calls the monitor code itself will NOT work. The only monitor point retained under 3.0 is .DLY. Calling any other monitor routine will crash the system.

[B] DIRECTORY STRUCTURE AND FLAGS

The directory structure has changed slightly. You will NOT be able to read HDOS 3.0 diskettes with earlier versions of HDOS. You will be able to read earlier diskettes with HDOS 3.0. Non-standard HDOS diskettes (using various time-of-day and other patches) may or may not work. Mount any diskettes in question with the write-enable notch COVERED. The directory entry now contains the following information:

* file name

* file type

* time and date of file's creation

* number of times the file has been accessed (up to 255)

* flags
  - A - File has been backed up [Note 2]
  - B - File contains bad sectors [Note 3]
  - C - File is contiguous on disk [Note 1]
  - D - File may not be deleted [Note 4]
  - L - Flags are locked
  - S - System file
  - U - User flag [Note 5]
  - W - File is read-only

NOTES:

(1) The contiguous flag [C] is automatically set by HDOS whenever a file is closed if that file happens to be contiguous on disk. The .OPENC SCALL may still be used as before to create 'C' files, as may the '/C' switch in PIP.

NOTES (Cont)

NOTES: (Cont)

(2) The archive flag [A] is used by a file archive (ARC) utility. The Archive utility is available separately from Kirk Thompson.

(3) The bad sector flag [B] is used by a disk verify utility which is called BAD.ABS. This utility is available separately from Kirk Thompson.

(4) The flag [D] locks a file against deletion. This does not imply write-protection, as the file may still be freely read or written. However, a file with the 'D' flag set may not be opened for .WRITE as this would cause the file to be deleted. Instead, open for .UPDATE must be used.

(5) The flag [U] is provided for user's use.

* user area mask (not implemented)

* first group number of file

* last group number of file

* last sector index of file

* file's creation date

* date of the file's last access

[C] DEVICE DRIVERS

Device drivers may be cleanly UNLOADED. They may also process interrupts. The user should refer to H17DVD.H8A and H37DVD.H8A for examples of how this works. Pre-3.0 drivers which process interrupts should not be used. The techniques used under 2.0 to process interrupts may crash the system under 3.0.

The device table size is determined dynamically at boot time. If you have two drivers (the minimum, allowing for SY: and TT:) you get two entries. If you have fifteen drivers (!) HDOS will build a table sufficient to hold all entries.

TT: is no longer part of HDOS but is an independent device driver. In addition to the standard device driver entry points, TT: includes routines to process the following SCALLs: .SCIN, .SCOUT, .PRINT, .CONSL, and .CLRCO. TT: also supports operation at 19200 and 38400 baud.

A Device driver preamble (the SET part of the driver) may be larger than two sectors. It may extend to 16 sectors in multiples of two sectors.

NOTES (Cont)

[D] SYSCMD/Plus and PIP/Plus

SYSCMD.SYS and PIP.ABS remain co-resident whenever possible, eliminating the repeated re-loading of PIP.

PIP has approximately 50 switches. See the "HELP." file for a brief description of them.

SYSCMD supports many new commands. It also supports execution of "batch" files. (A batch file is a text file containing commands which is read by SYSCMD). Batch file names end in ".BAT". SYSCMD will automatically search for and execute "SY0:AUTOEXEC.BAT" when the system boots. (This is in addition to, but AFTER, running of SY0:PROLOGUE.SYS by HDOS.) Operation of batch files is nearly identical to that of MS-DOS, with the exception of FOR/IN/DO which is not implemented.

[E] DEFAULT DEVICE DATA

A default device may be "logged in" from the SYSCMD prompt.

A search path is implemented which causes SYSCMD to search for commands which are not found on the default device.

All programs distributed with HDOS 3.0 use the default device for reading and writing files.

[F] LIST OF FILES FOR HDOS 3.02

HDOS30.SYS	version 3.02 of HDOS
TT.DVD	H19 driver
DK.DVD	Secondary disk drives driver
SY.DVD	Primary disk drives driver (less grinding sounds from drive)
H47.DVD	H47 driver (8-inch disks)
H37.DVD	H37 driver (soft-sector)
H17.DVD	H17 driver (hard-sector)
ND.DVD	Null device driver
RX.DVD	A Null device that tells you what its doing (debugging tool)
SYSCMD.SYS	3.02 System Command Processor
ERRORMSG.SYS	3.02 error list
HELP.	3.02 help file
PIP.ABS	3.02 peripheral interchange program
SYSHELP.DOC	3.02 help file
ACT.ABS	Show what tasks are loaded in "task manager"
BLANK.BAT	Screen blanking batch file. Touch any key to restore the screen

NOTES (Cont)

[F] LIST OF FILES FOR HDOS 3.02 (Cont)

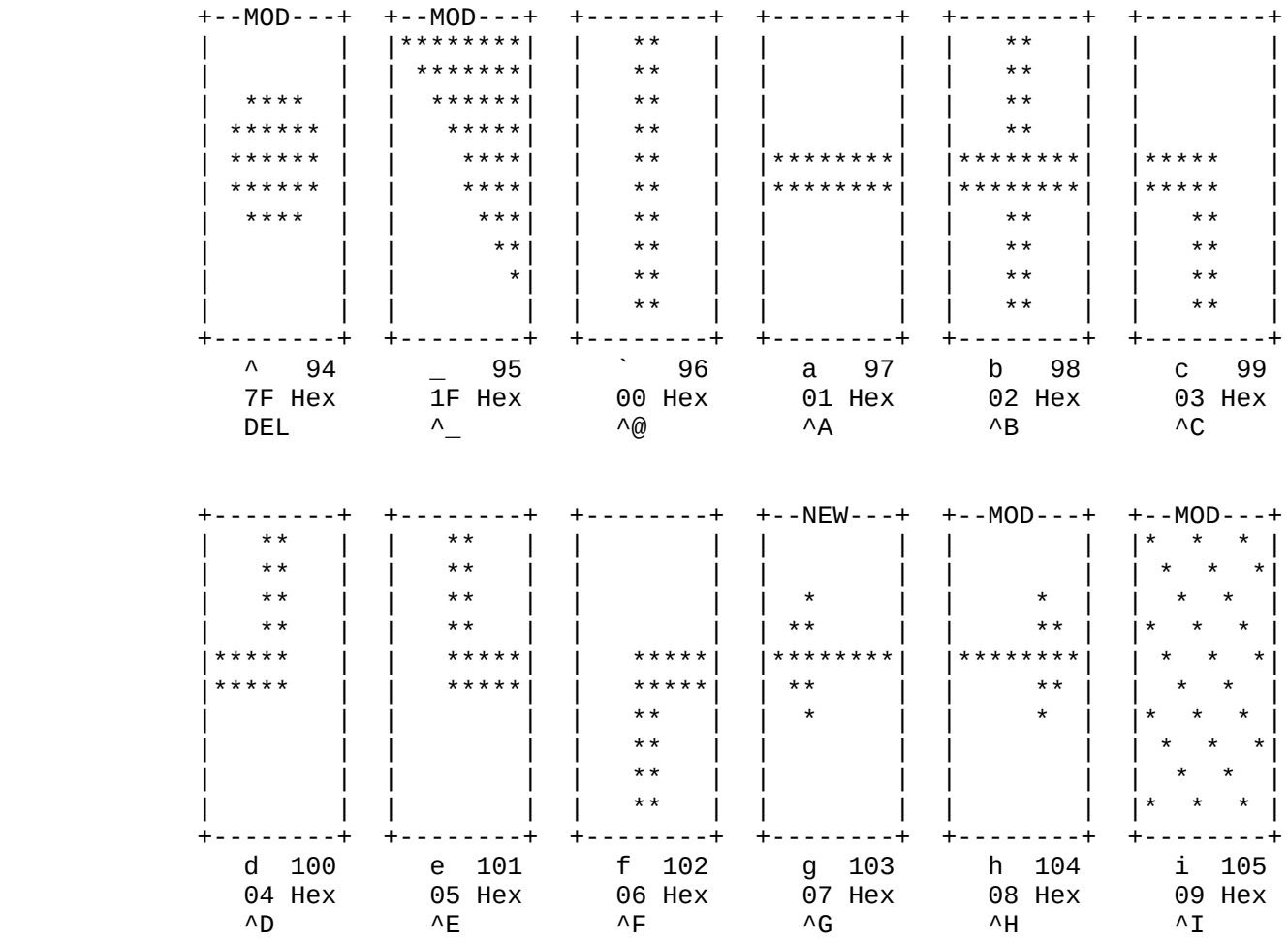
```
BLINK.BAT       Silly way to clear the screen
CALC.ABS        Newer calculator that includes date codes
CHAN.TAS <TMG>  Shows I/O channel activity on 25th line
CLOCK.TAS <TMG> Standard software clock
CRASH.TAS <TMG> Touch BREAK key to crash system
DFD.ABS         Deleted files directory
DS.ABS          Directory sort
DVL.ABS         Display volume label sector
DVT.ABS         Show contents of device table
ECHO.TAS <TMG>  Send screen output to LP: (First load LP:)
IOT.ABS         Show contents of I/O table
JTRA.ABS        Job translator utility
KAL.ABS         Pretty patterns on your screen
KEYS.TAS <TMG>  Program all 8 function keys
MAP.ABS         Print magic addresses for 3.02
MDRC.BAT        A tool for looking at lots of disks
MP.ABS          MEGAPIP, an HDOS file-handling utility
OC.ABS          Newer ONECOPY
OPE.ABS         A utility to alter memory
SHOWALL.BAT     Show lots of HDOS information
SORT.ABS        File sorting utility
SYSHELP.DOC     3.02 help file
SYSMON.TAS <TMG>Monitor STACK for overflow and S.FASER syscalls
SYSPATCH.ABS    PATCH without codes
TAS.ABS         Activates and deactivates tasks in 'Task Manager'
TDU.TAS <TMG>   Terminal debug utility
TICTOC.BAT      Start clock first; then try this
TMAP.ABS        Task map.  Shows which are in memory
TMG.TAS         The 'Task Manager.'  Must be started first before
                certain tasks will work
TSR.ABS         Task status report for 'Task Manager'
USR.ABS         Show system speed with or without user clock vector
ZZ.ABS          Zig-Zag
```

NOTES (Cont)

GRAPHICS CHARACTERS

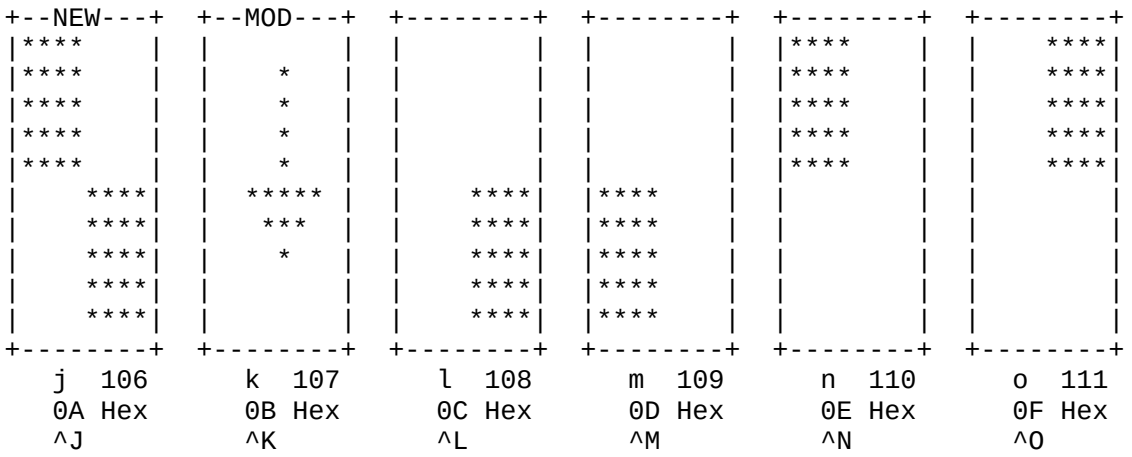
Below the character map is the lower case character (and it's decimal equivalent) which gives you the graphic character; followed by the Hex code and Control code for the native mode of the graphic character.

CHARACTER NATIVE CODE

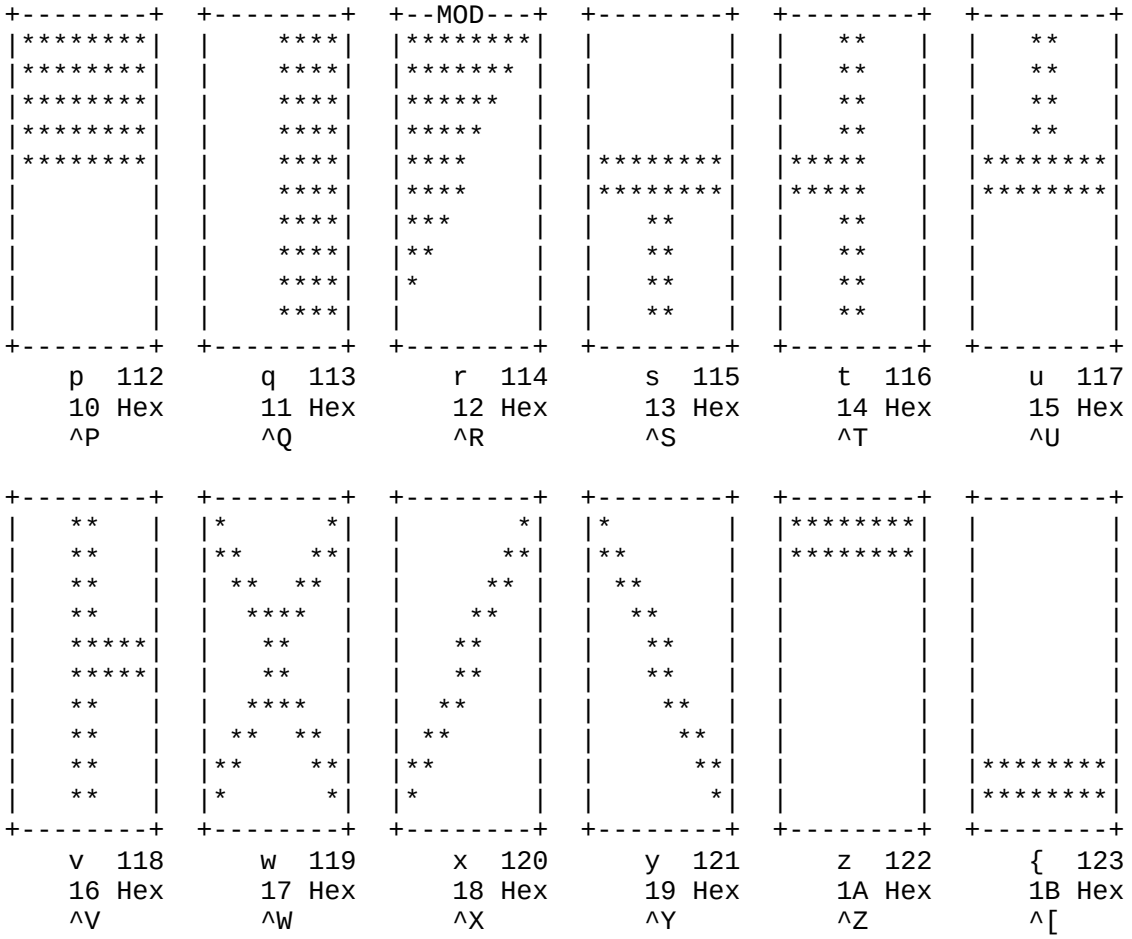


GRAPHICS CHARACTERS (Cont)

CHARACTER NATIVE CODE

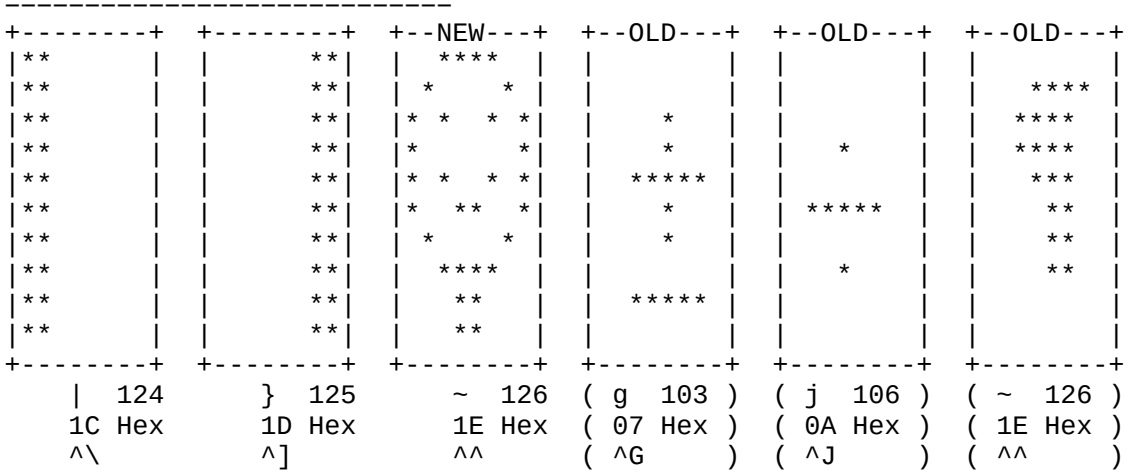


CHARACTER NATIVE CODE



GRAPHICS CHARACTERS (Cont)

CHARACTER NATIVE CODE (Cont)



NOTE: Reverse video for these characters has the high bit set in native mode.

****

THE ULTRA ROM

SPECIAL KEYBOARD SEQUENCES

Version 2.8 of the Ultra ROM Firmware

Control-Shift-ESC	Clear Transparent Mode if set. Clear Native Mode if set. Unlock keyboard if locked. No code is sent to host.
Control-TAB	Transmit 14H to host.
Shift-SCROLL	Enter Scroll Mode if not already set. Advance one page if in scroll mode. No code is sent to host.
Control-SCROLL	Exit Scroll Mode if set. No code is sent to host.
Control-Shift-DELETE	Soft Reset if H19 terminal. No code is sent to host.
Control-RETURN	Return cursor to column one. No code is sent to host.
Control-Shift-RETURN	Transmit current line edited. Line is terminated with CR.
Control-ERASE	Erase to end of screen. No code is sent to host.
Control-Shift-ERASE	Erase entire screen. Home cursor.

Note: Upon receipt of a 12H the terminal will emit the 'click' sound, similar to the tick of a clock.

NORMAL MODE FUNCTION KEYS

Values of function keys in their normal mode.
Native mode values are included for reference.

THE ULTRA-ROM (Cont)

NORMAL MODE FUNCTION KEYS (Cont)

SHIFTED

f 1	f2	f 3	f 4	f 5
ESC s	ESC t	ESC u	ESC v	ESC w
F3 Hex	F4 Hex	F5 Hex	F6 hex	F7 Hex

UNSHIFTED

ESC S	ESC T	ESC U	ESC V	ESC W
D3 Hex	D4 Hex	D5 Hex	D6 Hex	D7 Hex

SHIFTED

ERASE	BLUE	RED	WHITE
ESC E	ESC p	ESC q	ESC r
C5 Hex	F0 Hex	F1 Hex	F2 Hex

UNSHIFTED

ESC J	ESC P	ESC Q	ESC R
CA Hex	D0 Hex	D1 Hex	D2 Hex

THE ULTRA ROM (Cont)

NORMAL MODE FUNCTION KEYS (Cont)

How they function with the OFF LINE key down:

SHIFTED

f 1	f 2	f 3	f 4	f 5
ESC s swap page	ESC t enter shifted keypad	ESC u exit shifted keypad	ESC v wrap at end of line	ESC w discard at end of line

UNSHIFTED

ESC S [+ arg] cursor type	ESC T enter transparent mode	ESC U set HALF duplex	ESC V set FULL duplex	ESC W transmit character at cursor

SHIFTED

ERASE	BLUE	RED	WHITE
ESC E cls and home cursor	ESC p enter reverse video	ESC q exit reverse video	ESC r [+ arg] change baud rate

UNSHIFTED

ESC J erase to end of screen	ESC P enter native mode	ESC Q exit native mode	ESC R [+ arg] copy page to other

^

(SEE SECOND NOTE ON NEXT PAGE)

THE ULTRA ROM (Cont)

NORMAL MODE FUNCTION KEYS (Cont)

NOTE 1: This information is given so you will know what is going on if you get strange results from the function keys while the OFF LINE key is down.

NOTE 2: The entry noted as means that while in native mode, pressing the unshifted RED key will transmit the native mode code for that key, not 'ESC Q'. Therefore, it is not possible to exit native mode with that key. You can exit native mode by pressing the ESC key followed by the 'Q' key or by using CONTROL-SHIFT-ESC.

USER-DEFINED FUNCTION KEYS

There are two built in sets of defined strings for the unshifted function keys. Notice that with the shift key they are unchanged from normal mode. You can, of course, assign your own values to the unshifted keys.

SHIFTED (no change)

f 1	f 2	f 3	f 4	f 5
ESC s	ESC t	ESC u	ESC v	ESC w

UNSHIFTED CPM

f 1	f 2	f 3	f 4	f 5
'dir '	'type '	'list '	'stat '	'pip '
User ????????	User ????????	User ????????	User ????????	User ????????

UNSHIFTED HDOS

f 1	f 2	f 3	f 4	f 5
'mount'	'dis'	'reset'	'copy'	'type'

THE ULTRA ROM (Cont)

USER-DEFINED FUNCTION KEYS (Cont)

SHIFTED (no change)

ERASE	BLUE	RED	WHITE
ESC E	ESC p	ESC q	ESC r

UNSHIFTED CPM

ESC J	'ren '	'era '	'user '
	User ????????	User ????????	User ????????

UNSHIFTED HDOS

ESC J	'cat '	'SY1:'	'SY2:'
	User ????????	User ????????	User ????????

NOTE: Spaces follow some of the predefined strings.

The ERASE key is unaffected by this mode.

CONTROL-KEY MODE FUNCTION KEYS

At any time that the CONTROL key is depressed, the function keys perform the following:

SHIFTED

f 1	f 2	f 3	f 4	f 5
disable graphics mode	disable reverse video	disable wrap at end of line	disable shifted keypad mode	enter transparent mode

THE ULTRA ROM (Cont)

CONTROL KEY MODE FUNCTION KEYS (Cont)

UNSHIFTED

f 1	f 2	f 3	f 4	f 5
enable graphics mode	enable reverse video	enable wrap at end of line	enable shifted keypad mode	enable native mode

SHIFTED

ERASE	BLUE	RED	WHITE
ESC E	disable user function keys	copy 2nd page ram to video ram	zero the 25th line clock

UNSHIFTED

ESC J	enable user function keys	copy video ram to 2nd page ram	swap page

NOTE: The ERASE key is not affected by this mode.

These keys do not function this way in native mode. However, they will in transparent mode. While in transparent mode the graphics mode and reverse video mode will not function except where transparent mode uses reverse video normally. If you have graphics or reverse video turned on and then exit transparent mode then will still be enabled and functioning the way you expect.

If you are in transparent mode and turn on native mode then the function keys revert to sending native mode codes only.

THE ULTRA ROM (Cont)

CODES FOR THE KEYPAD

	insert char	up cursor	delete char
SHIFTED	ESC @ [O]	ESC A	ESC N
ALTERNATE	ESC ? w	ESC ? x	ESC ? y
SHIFTED NATIVE	97 Hex	98 Hex	99 Hex
UNSHIFTED NATIVE	87 Hex	88 Hex	89 Hex
UNSHIFTED	7	8	9

	left cursor	home cursor	right cursor
SHIFTED	ESC D	ESC H	ESC C
ALTERNATE	ESC ? t	ESC ? u	ESC ? v
SHIFTED NATIVE	94 Hex	95 Hex	96 Hex
UNSHIFTED NATIVE	84 Hex	85 Hex	86 Hex
UNSHIFTED	4	5	6

	insert line	down cursor	delete line
SHIFTED	ESC L	ESC B	ESC M
ALTERNATE	ESC ? q	ESC ? r	ESC ? s
SHIFTED NATIVE	91 Hex	92 Hex	93 Hex
UNSHIFTED NATIVE	81 Hex	82 Hex	83 Hex
UNSHIFTED	1	2	3

	zero	period	return
SHIFTED	0	.	CR
ALTERNATE	ESC ? p	ESC ? n	ESC ? M
SHIFTED NATIVE	90 Hex	9A Hex	9B Hex
UNSHIFTED NATIVE	80 Hex	8A Hex	8B Hex
UNSHIFTED	0	. (DOT)	ENTER

NOTE: The shifted 7 key has two sequences. The first one (ESC @) sets insert character mode and the second one (ESC O) exits insert character mode.

THE ULTRA ROM (Cont)

ESCAPE SEQUENCES

In alphabetical order, the '*' means an added feature of this ROM.

ESC #	transmit page
ESC :	* transmit current line
ESC ;	* transmit current line edited
ESC <	* NOT used - formerly ANSI MODE enable
ESC =	enter alternate keypad mode
ESC >	exit alternate keypad mode
ESC ?	* send configuration report
ESC @	enter insert character mode
ESC A	cursor up
ESC B	cursor down
ESC C	cursor right
ESC D	cursor left
ESC E	erase screen and home cursor
ESC F	enter graphics mode
ESC G	exit graphics mode
ESC H	home cursor
ESC I	reverse line feed
ESC J	erase to end of page
ESC K	erase to end of line
ESC L	insert line
ESC M	delete line
ESC N	delete character
ESC O	exit insert character mode
ESC P	* enter native keyboard mode

THE ULTRA ROM (Cont)

ESCAPE SEQUENCES (Cont)

```
  ESC Q       * exit  native keyboard mode

    ESC R <arg>   * copy display memory to/from 2nd page memory <arg>
                    is '1' or '2' which is the target of the copy.

    ESC S <arg> * set cursor type
                  <arg> is '1' thru '8' which is cursor type
                  1 underscore - steady          5 block - steady
                  2 underscore - invisible       6 block - invisible
                  3 underscore - fast blink      7 block - fast blink
                  4 underscore - slow blink      8 block - slow blink

  ESC T       * enter transparent mode

  ESC U       * set half duplex

  ESC V       * set full duplex

  ESC W       * transmit character at cursor

    ESC X <arg> * set clock
                  <arg>  is  a  seven-character  string  in  the  form:
                  'hhmmss' followed by any character, usually a return.

  ESC Y <r> <c>     direct cursor addressing
              <r> is row, <c> is column

  ESC Z       identify as VT-52 ( ESC / K )

  ESC [       enter hold screen mode

  ESC \       exit  hold screen mode

  ESC ]       transmit 25th line

  ESC ^       * reset clock to 00:00:00

  ESC _ <arg>   * reverse characters on screen
              <arg> is a count of how many characters to reverse

  ESC `       * reverse entire screen

  ESC a <n> <$> * load programmable function keys
              <n> is '1' thru '8' indicating which function key
              <$> is up to an 8 character string.  if not using
                    all 8 characters then must terminate with DEL.
```

THE ULTRA ROM (Cont)

ESCAPE CODES (Cont)

```
  ESC b       erase to beginning of page

  ESC c       * enable    clock display

  ESC d       * disable clock display

  ESC e       * send time to host

  ESC f <n> <$> * expand bytes vertical
              <n> is count, <$> is character to expand

  ESC g <n> <$> * expand bytes horizontal
              <n> is count, <$> is character to expand

  ESC h <arg>   * set/clear MODE 2 settings
              <arg> is '1' thru '8', mode to set/clear
              1 enable  software handshake
              2 disable software handshake
              3 start screen clock
              4 stop      screen clock
              5 enable  programmed function keys

              6 disable programmed function keys
              7 select CPM  function keys
              8 select HDOS function keys
              9 NOT USED

  ESC i <$>     * fill screen with byte
              <$> is the character to fill screen with

  ESC j       save cursor position

  ESC k       restore cursor position

  ESC l       erase entire line

  ESC m       * reset programmable function keys

  ESC n       cursor position report

  ESC o       erase to beginning of line

  ESC p       enter reverse video mode

  ESC q       exit  reverse video mode
```

THE ULTRA ROM (Cont)

ESCAPE SEQUENCES (Cont)

ESC r <arg> * set baud rate (NOT a new feature, but modified)
<arg> is 'A' thru 'H', new baud rate

A 110	E 4800	
B 300	F 9600	
C 1200	G 19200	
D 2400	H 38400	

from original ROM
<arg> is 'A' thru 'L', new baud rate

A 110	E 1200	I 3600
B 150	F 1800	J 4800
C 300	G 2000	K 7200
D 600	H 2400	L 9600

ESC s * swap display memory with 2nd page memory

ESC t enter shifted keypad mode

ESC u exit shifted keypad mode

ESC v set wrap at end of line

ESC w set discard at end of line

ESC x <arg> Heath set mode
<arg> is '1' thru '9', mode to set
1 enable 25th line
2 disable key click
3 enter hold screen mode
4 block cursor
5 cursor off
6 enter keypad shifted mode
7 enter alternate keypad mode
8 auto line feed on receipt of CR
9 auto CR on receipt of line feed

ESC y <arg> Heath reset mode
<arg> is '1' thru '9', mode to reset
1 disable 25th line
2 enable key click
3 exit hold screen mode
4 underscore cursor
5 cursor on
6 exit keypad shifted mode
7 exit alternate keypad mode
8 no auto line feed
9 no auto CR

THE ULTRA ROM (Cont)

ESCAPE SEQUENCES (Cont)

ESC z reinitialize to power-up configuration

ESC { enable keyboard input

ESC | * execute terminal self-test

ESC } disable keyboard input

NOTES ON THE ULTRA ROM:

This file was prepared for those people who have the Ultra ROM.

The Ultra ROM was designed by Bill Parrott III. For a time it was sold by Software Wizardry. Unfortunately, at the completion of this manual it is no longer commercially available.

CREDITS:

WRITER/TYPIST	Dan Jerome (SMUGH)
TECHNICAL ADVISOR #1	John Toscano (SMUGH)
TECHNICAL ADVISOR #2	Bill Cordes (SMUGH)
HDOS 3.0 PROGRAMMER	Bill Parrott III
HDOS 3.02 PROGRAMMER	Richard Musgrave
CHIEF OF QUALITY CONTROL:	Terry Hall

Key Vendor Name and Address	**Products for HDOS 3.0/3.02**
Lindley Systems c/o William Lindley 4257 Berwick Place Woodbridge, VA 22192 (703) 590-8890	Ultimate Printer Driver Misc software for HDOS
Quikdata, Inc. c/o Henry E. Fale 2618 Penn Circle Sheboygan, WI 53081 (414) 452-4172	Gibson HDOS 3.0/3.02 Assembler; Various H89 hardware and software; HDOS Software Reference Manuals;
Staunch 8/89'er c/o Kirk Thompson P.O. Box 548 Lot #6 West Branch Mobile Home Village, West Branch, IA 52358	Associated utilities for HDOS; HDOS Software Reference manual; manuals; all of the HDOS 3.02 files on disk or hardcopy; miscellaneous software

MEGAPIP DOCUMENTATION

MEGAPIP is an excellent file-handling utility for HDOS 3.02. This is the first shell type of program developed for the HDOS Operating System.

When you call up MP.ABS, a graphics rectangle will be "painted" on the screen. In the bottom of the large rectangle, a shorter rectangle will appear. In a few seconds, the directory of SY0: will appear. There will be a highlighted bar over the first file of the disk. This shadow bar may be moved by the arrow keys. In conjunction with the program keys, it will enable you to select which function you want to perform.

Below the list of files in the shorter rectangle is a comment stating that if you want help, press the question mark [?] key. When you press the question mark key -- you don't need to press <RTN> -- the first screen blanks out and another screen takes its place. The contents of the first Help screen is as follows:

```
-- Tagging Functions --            -- File Functions --

T = Tag file                       C = Copy files
U = Untag file                     R = Rename files
W = Wild tag/untag                 D = Delete files
                                   I = File info
                                   N = File CRC
-- Misc. Functions --              F = Alter file(s) Flags
                                   A = File(s) user Areas
L   = new Login
S   = free Space
E   = Edit file                    -- Viewing Functions --
X   = eXecute file
+/- = next/previous screen         V = View file(s)
?   = Help                         P = Print file(s)
Q   = Quit MegaPip                 H = Hex Dump
```

The contents of the second Help screen is as follows:

```
                f1 = Run PIP
            f2 = Sort file table
            f4 = active user area
          f5 = mount/dismount/reset

             arrows move cursor
             home -> first file
             white -> last file

             esc = abort at key
         ^D (CTRL-D) = abort at text

                blue = refresh
                 red = quit
```

How to Use MegaPip:

BACKGROUND:
The first time you call MegaPip to your screen, note the screenful of files and the beautiful graphics rectangles drawn to contain them. MegaPip will bring up the list of files from SY0:. If you want to bring up the file list of SY1: or SY2:, just type 'L' at the cursor. You will be asked for an argument. In this case, just type 'SY1:', or the desired drive name, and press '<RTN>'. MegaPip will then log onto SY1:.

In the upper right hand corner a legend will say: "Screen 1 of 1." If you are looking at an 80-track double-sided drive, the legend may say "Screen 1 of 2." If you have more files than that, it could say: "Screen 1 of n," where n stands for a number.

You can move the shadow bar with the use of the arrow keys. To move from screen to screen, place the shadow bar on the last column of files to the right and then place it on the last file in the column. Then type '<RTN>'.

HOW TO USE THE UTILITY:
Probably one of the most often used tasks is copying files from one drive to another. To copy files, first you must tag them. First place the shadow bar on the first file that you want to copy. Type 'T' where the shadow bar is. (Ignore the apostrophes.) Continue to move the shadow bar to the file(s) that you want to copy and type 'T' for each one. When you are done with tagging the files to be copied, type 'C' at the cursor down at the bottom of the page.

The program will ask: "COPY - <T>agged, <U>ntagged, <F>ile." Type 'T.' The program will ask: "Copy TAGGED to?" Type 'SY1:'. The program goes to a plain screen and prints the following:

"PIP SY1:=SY0:Filename.Ext/S/SU:CST"

"SY0:Filename.Ext --> SY1:Filename.Ext ... Copied"

This expression made by the computer is made for each file that you want copied. When it is done, the program instructs: "Touch Any Key .... " When you touch any key, the screen shifts, and you find yourself back into the main screen of MegaPip.

When you want to exit the program, type "Q" at the cursor. The program will ask: "Are You Sure ? Y/N." Just type 'Y,' and you will be back at the HDOS prompt, DAN+>.

This is but one example of the versatility of this fine program. For example, when you are in MegaPip, you can sort the file table, enter a user area, mount or dismount a disk, or do any of the options shown in the tables above.

CAUTION

MegaPip is a fine program, but it is recommended that you approach it

carefully, since it can be a dangerous. The first time one uses it, you can inadvertently delete all of your files on the disk being displayed. To cause this to happen, all you have to do is to place the shadow bar on the file that you want deleted and type 'D," at the cursor, and press <RTN>. All of the files on disk will be deleted! To prevent this catastrophe from occuring, first tag the specific files that you want deleted. THEN tell the program to delete only the tagged ones.

MegaPip.DOC, HDOS 3.02
Last Edited: 11-Apr-90