HP 300

Service Handbook

Part IIa — Diagnostic Tools

HEWLETT PACKARD

Hewlett-Packard Company
19447 Pruneridge Ave.,
Cupertino, California 95014

Printed in U.S.A., April 1979.
Reprinted July 1981, incorporating updates 1,2,3, and 4
(July 1979, Feb 1980,
June 1980, May 1981)

NOTICE

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

Hewlett-Packard Company
19447 Pruneridge Ave., Cupertino, California 95014 U.S.A.

Printed in U.S.A.

MAP OF SECTIONS

PART I SUMMARIES	PART II DIAGNOSTIC TOOLS	PART III TUTORIALS
SYSTEM		
GENERAL		
100 Documentation		700 Documentation
102 Site & Installation		702 Site & Installation
103 Preventive Maintenance		
106 Parts & Logistics		
DIAGNOSIS		
110 Diagnosis Overview		710 Diagnosis Overview
		711 Diagnosis Forms
112 System Symptoms		712 System Symptoms
113 System Procedures		713 System Procedures
114 Load/Dump/Startup		714 Load/Dump/Startup
TOOLS		
	440 DUS	
	441 AID Language	
	442 IOMAP	
	446 I/O Exerciser	
	447B FMDISCUT	
	447C FMUTILIT	
150 PMP (31321)	447D FMPATCHV	
	448 WORKOUT 300	
	450 PMP (31321)	
	455 SLEUTHSM	
I/O		
160 IMB		760 IMB
161 HP-IB		761 HP-IB
162 I/O Overview		762 I/O Overview
HARDWARE		
170 Block Diagram		
171 Configuration		
172 Instruction Set		
OPERATING SYSTEM		
	482 System Debug	
REFERENCE		
190 Character Codes		
191 Base Conversion		
192 Mnemonics		
		794 Notation
198 RS-232		798 RS-232
199 Error and Status Codes		
HARDWARE SUBSYSTEMS		
MAINFRAMES		
201 Mainframe (31213)		801 Mainframe (31213)
202 Power Supply (31213)		802 Power Supply (31213)
CPUs		
211 CPU (-60052)	511B CPU Test Prog. (-60052)	811 CPU (-60052)
	511C CPU Self-Test (-60052)	
MEMORY		
221 Memory (31202)	521B Mem. Diag. (31202)	821 Memory (31202)
CHANNELS		
231 GIC (31262)	531B GIC Diag. (31262)	831 GIC (31262)
232 ADCC (31264)	532B ADCC Diag. (31264)	832 ADCC (31264)
	532C ASYNCOMM Exer/Diag.	
CONSOLES		
251 IDS (31213)	551B IDS Diag. (31213)	851 IDS (31213)
	551C IDS Self-Test (31213)	
DISCS		
260 Discs		860 Disc/Sys Rel'p
	560C RATES	
261 7902 FDU	561B 7902 FDU Diag.	861 7902 FDU
	561C 7902 FDU Self-Test	
	561D 7902 FTEST	
262 7910 DSU	562B 7910 DSU Diag.	862 7910 DSU
	562C 7910 DSU Self-Test	
263 13037 Discs	563B 13037 Diagnostic	863 13037 Discs
PRINTERS		
	581B 2608 Exerciser	
282 2631 Printer	582B 2631 Exerciser	
TERMINALS		
292 264X Terminals		
293 262X Terminals		
398 Tool Characterization		
399 Service Notes		

CHANGE HISTORY

Sect No.	Title	Current Dates	Comment
	PART II		
440	DUS	JUN 1980	Revised
441	AID Language		
A	Introduction	JUN 1980	Revised
B	Essentials of AID	JUL 1979	Revised
C	Commands	JUN 1980	Revised
D	Statements (Non-I/O)	APR 1979	Revised
E	Special Characters	APR 1979	Revised
F	Operators	APR 1979	Revised
G	Reserved Variables	JUL 1979	Revised
H	I/O Non-Channel Prog	APR 1979	Revised
I	Channel Program	APR 1979	Revised
J	Function Statements	APR 1979	Revised
K	Quick Reference	JUL 1979	New
442	IOMAP	JUN 1980	Revised
446	I/O Exerciser	APR 1979	Revised
447B	FMDISCUT	MAY 1981	Revised
447C	FMUTILIT	MAY 1981	Revised
447D	FMPATCHV	JUN 1980	Rewritten
448	WORKOUT300	MAY 1981	Revised/renamed
450	PMP (31321)	APR 1979	Revised
455	SLEUTHSM	JUN 1980	New
482	System Debug	MAY 1981	Revised
511B	CPU Test Prog.(-60052)	JUN 1980	Revised
511C	CPU Self-Test (-60052)	APR 1979	Revised
521B	Memory Diag. (31202)	MAY 1981	Revised
531B	GIC Diag. (31262)	JUN 1980	Revised
532B	ADCC Diag. (31264)	JUN 1980	Revised
532C	ASYNCOMM Exer/Diag.	JUN 1980	New
551B	IDS Diag. (31213)	JUN 1980	Revised
551C	IDS Self-Test (31213)	MAY 1981	Revised
560B	FDISC	JUN 1980	Deleted
560C	RATES	JUN 1980	New Rev. level
561B	7902 FDU Diag.	JUN 1980	New Rev. level
561C	7902 FDU Self-Test	APR 1979	Revised
561D	FTEST Program	JUN 1980	Rewritten
562B	7910 DSU Diag.	JUN 1980	New version "K"
562C	7910 DSU Self-Test	APR 1979	Revised
563B	13037 Diagnostic	JUN 1980	New version "C"
581B	2608 Exerciser	JUN 1980	Revised
582B	2631 Exerciser	MAY 1981	Revised

SAFETY SUMMARY

The following general safety precautions must be observed during all phases of operation, service, and repair of this system. Failure to comply with these precautions or with specific warnings elsewhere in this manual violates safety standards of design, manufacture, and intended use of the system. Hewlett-Packard Company assumes no liability for the customer's failure to comply with these requirements.

GROUND THE SYSTEM.

To minimize shock hazard, the system chassis and cabinet must be connected to an electrical ground. The system is equipped with a three-conductor ac power cable. The power cable must be plugged into an approved three-contact electrical outlet with a safety ground. The power jack and mating plug of the power cable meet International Electrotechnical Commission (IEC) safety standards.

DO NOT OPERATE IN AN EXPLOSIVE ATMOSPHERE.

Do not operate the system in the presence of flammable gases or fumes. Operation of any electrical system in such an environment constitutes a definite safety hazard.

KEEP AWAY FROM LIVE CIRCUITS.

Operating personnel must not remove system covers. Component replacement and internal adjustments must be made by qualified maintenance personnel. Do not replace components with power cable connected. Under certain conditions, dangerous voltages may exist even with the power cable removed. To avoid injuries, always disconnect power and discharge circuits before touching them.

SAFETY SUMMARY

DO NOT SERVICE OR ADJUST ALONE.

Do not attempt internal service or adjustment unless another person, capable of rendering first aid and resuscitation, is present.

DO NOT SUBSTITUTE PARTS OR MODIFY SYSTEM.

Because of the danger of introducing additional hazards, do not install substitute parts or perform any unauthorized modification to the system. Refer the system to a Hewlett-Packard Sales and Service Office for service and repair to ensure that safety features are maintained.

DANGEROUS PROCEDURE WARNINGS.

Warnings, such as the example below, precede potentially dangerous procedures throughout this manual. Instructions contained in the warnings must be followed.

WARNING

Dangerous voltages, capable of causing injury, are present in this system. Use extreme caution when handling, testing, and adjusting.

SECTION 440

DIAGNOSTIC/UTILITY SYSTEM

(DUS)

EXTERNAL REFERENCE SPECIFICATION

Program Revision: 01.xx

References:

AID ERS [Handbook Section 441]
Diagnostic and Test Program ERSs [4xx, 5xx Handbook Sections]

NOTICE

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

Hewlett-Packard Company
19447 Pruneridge Ave., Cupertino, California 95014 U.S.A.

Printed in U.S.A.

TABLE OF CONTENTS

LIST OF FIGURES

1.0 INTRODUCTION

The Diagnostic/Utility System is a memory-resident means of running diagnostic and utility programs on the HP 300 and HP 3000/33 systems.

The Stand Alone File Manager (hereafter referred to as SAFM) is a disc-based software module forming the heart of the Diagnostic/Utility System.

In addition to the SAFM, the Diagnostic/Utility System includes the AID (Advanced Interactive Diagnosis) interpreter together with a set of SPL-II and AID programs and supporting files. Generally, those programs provided in support of the Operating System are classified as Utilities and programs whose primary function is to test hardware and firmware subsystems or peripherals are classified as Diagnostics.

Independent of operating systems, the SAFM gives you access to files (located on a disc) and enables you to modify, delete, add or create those files. A disc-based directory allows interchange of file information with other discs.

1.1 Required Hardware:

HP 300 or HP 3000/33 consisting of:

(1) Memory - 128K words minimum

(2) Console- HP 300: IDS or HP 264X/262X terminal and ADCC board

HP 3000/33: Console or HP 264X terminal

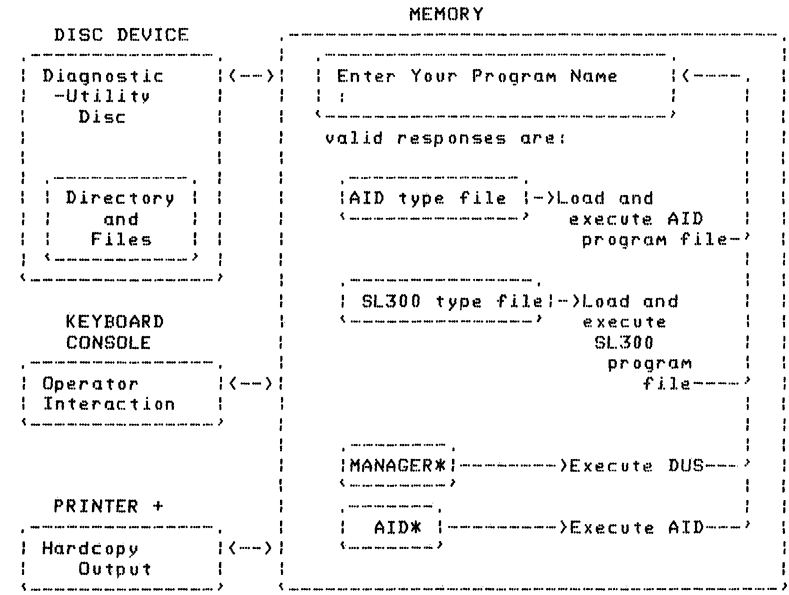
(Terminal is strapped and configured same as for operating system use. Terminal may be set for any buad rate -- DUS uses ENQ/ACK tests to find the terminal's baud rate.)

(3) Disc - HP 7902 Flexible Disc Unit

(4) Printer- HP 2631 printer (optional)

1.2 Overview

The following diagram provides a picture of how the SAFM integrates with other program modules and the Diagnostic/Utility System:



* See Security Code, paragraph 2.6, for further details.
+ Optional

Figure 1.1 - Diagnostic/Utility System Structure

2.0 OPERATING INSTRUCTIONS

2.1 Loading the System

Loading DUS on an HP 300:

(1) Bring any application programs and the operating system to an orderly halt.

(2) Insert a Diagnostic/Utility flexible disc in the Flexible Disc Unit.

(3) Set the Control Panel CHANNEL and DEVICE switches to the CHAN ADDR and DEVICE ADDR of the 7902 FDU.

(4) Press HALT, press SYSTEM RESET, then press LOAD.

(5) The Control Panel READY lamp will illuminate.

(6) SELECTED lamp will extinguish when the cold load is complete. RUN light heartbeat; NIR LEDs show DUS program halt !35.

(7) Set the Control Panel CHANNEL and DEVICE switches to the CHAN ADDR and DEVICE ADDR of the System Console. (For a terminal on an ADCC, set the DEVICE switch to the port number of the terminal -- terminal baud rate is determined automatically by DUS.)

(8) Press RUN.

(9) RUN light on. The welcome message and prompt are displayed:

```
Diagnostic/Utility System (revision XX.XX)
Enter your program name (Type HELP for program information)
:
```

(The revision is determined by the latest release date of the DUS program; HELP is an AID program that presents file and command information.)

If no device or bad device at address -- RUN light heartbeat; NIR LEDs show DUS program halt !31.

Loading DUS on an HP 3000/33:

(1) Perform an MPE 'SHUTDOWN' to properly logoff every current session, if applicable.

(2) Run the console Self-Test by pressing TEST on the keyboard and verify the displayed results (see 2645 User's Manual).

(3) Turn PROCESSOR power OFF and then ON to place the system hardware in MICRO RUN and PROGRAM RUN. (The DUS system will not load if this step fails). The RUN light on the front panel should be lighted.

(4) Fully reset the console by depressing the RESET TERMINAL key rapidly twice.

(5) Insure that the console is in REMOTE. (REMOTE key in depressed position.)

(6) Insert a Diagnostic/Utility flexible disc into the 7902 Flexible Disc Unit.

(7) Set front panel COLD LOAD thumbwheels to the CHAN ADDR and DEVICE ADDR of the 7902 FDU.

(8) Press HALT, then press LOAD.

(9) The welcome message and prompt are displayed:

```
Diagnostic/Utility System (revision XX.XX)
Enter your program name (Type HELP for program information)
:
```

(The revision is determined by the latest release date of the DUS program; HELP is an AID program that presents file and command information.)

2.2 Console Control

All user inputs are terminated with ENTER or carriage return/line feed on the console device.

To interrupt program execution, press:

- ATTN if the console is an HP 300 IDS
- Control-Y if console is a terminal

2.3 Running Programs

To execute an AID or SL300/SLMS program file, enter the program name as follows:

```
Enter your program name (Type HELP for program information)
:PROGNAME (The program PROGNAME will now be loaded and executed)
    ¦
    ¦
(Upon completion of the program the Diagnostic/Utility System
  returns to its entry mode)
    ¦
Enter your program name
:
```

2.4 Using the File Manager

If you wish to create, modify, or inquire about files type "MANAGER"*. You will be prompted with:

```
Stand Alone File Manager (revision XX.XX)
Enter Command (LC for List Command)
>  (Any DUS command may now be executed)
```

* See Security Code, paragraph 2.6, for further details.

2.5 Using AID

If you wish to create, modify, or make ad hoc changes to an AID programs you may do so by typing "AID"*. The resulting interaction is described in the AID ERS. [Handbook Section 441]

* See Security Code, paragraph 2.6, for further details.

2.6 Security Code

The system and program security code is the ASCII value (not character) of one. This character is generated as follows:

HP 300 IDS - any Soft Key or CNTL in unison with any numeric key

264X - CNTL in unison with 'A'

The security code is normally required to execute a file or command which allows file access (e.g. SAVE, PURGE) or alteration of a test sequence (i.e., the TEST command in AID).

3.0 FILE STRUCTURES AND FORMATS

3.1 Filenames

Filenames are restricted to eight alphanumeric ASCII characters starting with an alpha character.

Valid Filenames	Invalid Filenames
TEST	4DIAG
D44TEST	TEST.
ADCCDIAG	.TEST
B	AB/TEST

Note - The filenames AID, DIREC, IDSBOOT and SCRATCH are reserved.

3.2 File Types

Internally, files are typed as follows:

Type	Description	Created by
AID	AID program	AID SAVE Command
SL300	SL300 program	DUS SAVE Command
DATA	data file	CREATE Command
SLMS	Multiple Code Segment SL300 program	DUS SAVE Command

The AID, SL300, and SLMS types constitute the programs available to the user. The DATA files are transparent to the user but are used by development or accessed by some of the programs.

3.3 File Classes

File classes have no significance to the Diagnostic/Utility System. They are provided for the support of software which reads the directory. AID and SL300/SLMS program files are classified according to the service they provide. There are two classes of program files: UTILITY (U) and DIAGNOSTIC (D).

Data files are classified by content: ASCII (A), BINARY (B), CPUCODE* (C), and MCCCODE* (M).

* HP 300 only -- see IDS Diagnostic ERS [Handbook Section 551B]

At file creation time each AID program file is classified as a DIAGNOSTIC, each SL300/SLMS program file as a UTILITY and all DATA files as ASCII. The CLASSIFY Command may be employed to change the classification as required.

3.4 File Access

The Stand Alone File Manager user may access any file on any Diagnostic/Utility Disc.

- If you have entered an AID program and cannot save it on disc because of lack of space, you may remove the currently installed Diagnostic/Utility Disc and insert another Diagnostic/Utility Disc and again attempt to save your program (this process is repeatable indefinitely).

- Similiarly if you need a file that doesn't reside on the currently installed Diagnostic/Utility Disc, you can simply insert another Diagnostic/Utility Disc and determine whether or not the new disc contains the file you want.

There are some restrictions when accessing certain types of files. Most of these restrictions will be pointed out throughout this document, however a few general rules apply:

- The directory file (DIREC) is a permanent file which can be read but can never be modified or deleted.

- The IDSBOOT file (see IDS Diagnostic ERS, Handbook Section 551B) is a permanent file which can be read but can never be deleted. Because of the requirements of the HP 300 IDS microcode, IDSBOOT is handled in a special manner by DUS. A new version of IDSBOOT may be installed on a DUS disc via the SAVE command; however IDSBOOT may not be PURGE'd, RENAME'd, CHANGE'd, or CLASSIFY'd.

- Files which are protected must be unprotected before alteration. (See the CHANGE command.)

- The SCRATCH file is a 60 sector scratch area usable by anyone for general temporary data storage.

- As of DUS Version 01.01, the HP 300 DUP was given the capability to read and write files from/to an HP 3000/33 or HP 3000/30 DUP diskette. This capability is intended to allow the exchange of AID programs and data files between HP 300 and HP 3000/33 (etc.) systems.

  CAUTION

  Be careful not to try to exchange SL300/SLMS or SPL program files between HP 300 and HP 3000 systems! They are not cross-compatible!

4.0 FILE COMMANDS

The Stand Alone File Manager contains a command set that allows alteration of and access to files. The commands are explained in detail on the following pages. For convenience, some parameters are optional; optional parameters are enclosed in brackets[]. The operator may input any valid command after the DUS prompts with:

```
Enter Command (LC for List Commands)
>
```

Any error in syntax or errors which occur during command execution are identified by a message. Should difficulty arise understanding an error message refer to Error Messages, Section 5.2 of this document.

4.1 CHANGE

OPERATION NAME: Change file security

MNEMONIC:
```
CHANGE filename TO U[NPROTECTED]
CHANGE filename TO P[ROTECTED]
```

DESCRIPTION: Allows the operator to protect or unprotect a file. A protected file indicates it is not PURGEable and is read-only.

EXAMPLES:
```
Enter Command (LC for List Commands)
>CHANGE DIAG4 TO P (changes the file DIAG4 to
                    a non-PURGEable and read
                    only file)
Enter Command (LC for List Commands)
>CHANGE DIAG4 TO U (change DIAG4 to a PURGEable
                    read/write file)
```

4.2 CHANGEIO

OPERATION NAME: Change I/O device number

MNEMONIC: CHANGEIO CONSOLE TO channel number, device number
[DISC]
[PRINTER]

DESCRIPTION: Changes the default I/O device used by the DUS. The channel number is accepted as decimal in the range 0<=channel number<=15 and the device number must be in the range 0<=device number<=7. There must be a legal device at that location. A channel and device number equal to 0 implies that device is not available to the DUS. see LISTIO command.

EXAMPLE:

```
Enter Command (LC for List Commands)
>LISTIO

DEVICE TYPE          CHANNEL      DEVICE
-----------          -------      ------

CONSOLE                3            0
DISC                   2            6
PRINTER                2            3

Enter Command (LC for List Commands)
>CHANGEIO PRINTER TO 3,1 (change printer to
                          CHANNEL 3,DEVICE 1)
Enter Command (LC for List Commands)
>LISTIO

DEVICE TYPE          CHANNEL      DEVICE
-----------          -------      ------

CONSOLE                3            0
DISC                   2            6
PRINTER                3            1
```

4.3 CLASSIFY

OPERATION NAME: Reclassify a file

MNEMONIC: CLASSIFY filename AS class

DESCRIPTION: This Command has no significance to the Diagnostic/ Utility System but provides support for software which accesses the directory. It allows the user to reclassify the file filename to a new class where

class = U[TILITY]
D[IAGNOSTIC]
A[SCII]
B[INARY]
C[PUCODE]*
M[CCCODE]*

EXAMPLE: Enter Command (LC for List Commands)
>CLASSIFY DATA1 AS B (changes the file DATA1 to a BINARY classification)

* HP 300 only -- see IDS Diagnostic ERS [Handbook Section 551B]

4.4 CREATE

OPERATION NAME: Create a data file

MNEMONIC: CREATE filename, number of sectors [,revision]

DESCRIPTION: Creates (i.e. adds to the directory of files) an ASCII data file named "filename" which will be "number of sectors" long. The range on the number of sectors is 1(=sectors(=310. If the optional revision is not added then the revision 00.00 is used. (See LF Command for the format of revision).

EXAMPLE: Enter Command (LC for List Commands)
>CREATE TEST,4, 01.02 (creates an ASCII data file TEST with a length of 4 sectors and a revision of 01.02)

4.5 EXIT

OPERATION NAME: Leave file manager

MNEMONIC: EXIT

DESCRIPTION: Causes computer to leave the file manager and return to the Diagnostic/Utility System entry mode.

EXAMPLE:
```
Enter Command (LC for List Commands)
)EXIT

Enter Your Program Name
:
```

4.6 LC

OPERATION NAME: List the file management commands

MNEMONIC: LC

DESCRIPTION: Lists the File Manager Commands followed by a short description of what the command does.

EXAMPLE:
```
Enter Command (LC for List Commands)
)LC

LF              List the file directory
 .                   .
 .                   .
 .                   .
 .                   .
```

4.7 LF

OPERATION NAME: List the file directory

MNEMONIC: LF [P[RINTER]]

DESCRIPTION: Lists the file directory of the resident Diagnostic/Utility Disc which contains all pertinent information for the user. If the optional PRINTER is used the directory will be listed on the system printer device.

EXAMPLE:
```
Enter Command (LC for List Commands)
>LF

Stand Alone File Directory

Filename Type Class P/U Length Cyl Hd Sec Revision Prog  Data  Stack
--------------------------------------------------------------------
TEST     AID    U     P   427    4   0   7   00.00    320   28123 107
DIAG     SL300  D     U   400    4   0   11  00.00    300   100   200
ABC      DATA   A     U   1280   4   0   16  00.00    0     0     0
DIAG1    SLMS   D     P   750    4   0   26  00.00    400   350   600

CYLINDERS USED=5
```

The list header has the following meaning:

Filename - the name of the file.
Type - the file type that filename is currently designated as (see File Types Section for explanation of type meanings).
Class - classification of the file.
P/U - designates whether the file is Protected or Unprotected.
Length - the length of the file in words. This length is calculated as follows:

AID type = size of the AID program before executionon
SL300/SLMS type = size of the program (PL-PB) + size of the data area (DL-DB). The stack (Z-SB) occupies no space in the file.
DATA type = created size

Cyl - the physical disc cylinder address of the file.
Hd - the physical disc head address of the file.
Sec - the physical disc sector address of the file.

Revision - a five-digit code with the following format:

01.02

where 01 signifies the major revision level and
02 signifies the minor revision level.

Prog - this length in words is calculated as follows:

AID type = program area (object code) of the AID program.
SL300/SLMS type = Program Limit-Program Base Register (PL=PB)
DATA type = no significance.

Data - this length in words is calculated as follows:

AID type = buffer area available for the AID program.
SL300/SLMS type = Data Limit-Data Base register (DL-DB).
DATA type = No significance.

Stack - this length in words is calculated as follows:

AID type = (number of AID statements in the program x 2)+4.
SL300/SLMS type = Stack Limit-Stack Base register (Z-SB).
DATA type = No significance.

The "CYLINDERS USED" message indicates the amount of cylinders allocated by the system including the "holes" left by PURGE and SAVE.

4.8 LISTIO

OPERATION NAME: List the System I/O

MNEMONIC: LISTIO

DESCRIPTION: Lists the current I/O configuration of the System Console, System Disc and System Line Printer. This configuration may be modified by hardware (changing a device's device number) or by software (see the CHANGEIO command). A channel and device number equal to 0 imples that device is not available to the DUS.

EXAMPLE: See CHANGEIO command example.

4.9 LOAD

OPERATION NAME: Load file into memory

MNEMONMIC: LOAD filename

DESCRIPTION: Loads a file into the memory, This command would typically be used for modifying a file (i.e. LOAD, modify memory, SAVE) or for transferring a file from one disc to to another (i.e. LOAD, switch discs, SAVE).

4.10 PACK

OPERATION NAME: Pack files

MNEMONIC: PACK

DESCRIPTION: The disc is never packed until this command is executed so "holes" may develop in the file structure as a result of the PURGE and SAVE commands. To remove these "holes" a PACK should be executed so that the files on the disc will be contiguous. Note however, an unpacked disc presents no problem until a file cannot be stored because of no room left on the disc.

4.11 PURGE

OPERATION NAME: Purge File

MNEMONIC: PURGE filename

DESCRIPTION: Allows the operator to remove a file from the disc. All files may be purged except protected files. If a protected file must be purged the operator must change the file to unprotected and then purge it (See CHANGE command).

EXAMPLE:
```
Enter Command (LC for List Commands)
>PURGE DIAG
```

4.12 RENAME

OPERATION NAME: Rename File

MNEMONIC: RENAME old name, new name

DESCRIPTION: Allows the operator to change the name of a file. No other characteristic of the file is changed.

EXAMPLE:
```
Enter Command (LC for List Commands)
>RENAME DIAG1, DIAG44 (DIAG1 becomes DIAG44 i.e.
                       DIAG1 no longer exists).
```

4.13 SAVE

OPERATION NAME: Save a file by storing it on disc

MNEMONIC: SAVE filename [,revision]

DESCRIPTION: Stores the AID, SL300/SLMS or DATA file that is currently in memory onto the System disc. This command would typically be used for modifying a file (i.e. LOAD, modify memory, SAVE) or transferring a file to another disc (i.e. LOAD, switch discs, SAVE). If the optional revision is not added the current revision of the file is used (See LF Command for revision format).

EXAMPLE:
```
Enter Command (LC for List Commands)
>SAVE DIAG, 01.02
```

5.0 ERROR INTERPRETATION

5.1 Firmware Traps

If the machine firmware detects a condition that takes control from the executing user program (e.g. Bounds Violation, Stack Overflow) because of either a software or hardware problem, the following message is printed on the System Console:

```
Example:      **SYSTEM FAILURE**
              While executing FILENAME
              Delta P=!341  Code Segment=!3
              Stack Overflow
```

Delta P equals the hex offset from PB+0. Code segment equals the code segment that was executing when the failure occurred and finally, a descriptive message indicating the nature of the failure (i.e. Stack Overflow in this example). The system is halted and if RUN is pressed an attempt to recover back to the Diagnostic/Utility System entry mode is made.

5.2 Error Messages

Message	Meaning
Invalid Filename	The filename parameter did not meet the requirements of a valid filename (See Filenames, Section 3.1 of this document.)
Disc Failure!! Did not respond within 10 seconds	The system disc didn't complete a seek, read or write within a reasonable time (approximately 10 sec). Possible hardware failure.
Printer Failure!! Did not respond within 10 seconds	A line printer output was requested but the line printer did not complete its operation in normal time (approximately 10 seconds). Check for printer on-line, printer device correct and printer attached to HP-IB correctly.

Disc error on Directory write! Disc is probably no longer usable!	While writing an updated directory onto the disc a disc write and subsequent retry failed. The directory may be in one of the following states: 1) invalid data was written meaning the Diagnostic/ Utility Disc file system is no longer usable. 2) no write actually occurred meaning the Diagnostic/Utility Disc is intact with the last file operation disregarded. 3) enough data was written before the error occurred meaning the Diagnostic/ Utility Disc would be intact with the last file store operation successful. In any case try a new cold-load and LF Command to ascertain the condition of the Diagnostic/Utility Disc.
File Directory Full	The current disc operation would exceed the 58 filename directory entries limit. Alternatives include PURGEing a file or using another disc.
Insufficient Disc Space	There's no disc space available for this file. Alternatives include inserting a different Diagnostic/Utility Disc and retrying the store operation or executing the PACK Command and then retrying the store operation.
File access violation	This error occurs when a file access is attempted on a file or file type that isn't compatible with the command or operation, (e.g. RENAME oldfile,newfile where newfile exists already). Also occurs when an attempt is made to alter the files DIREC or IDSBOOT.
File system unaltered	Can occur during a command such as SAVE. A recoverable disc error occurred with no alteration of the directory. Retries of the last operation may be attempted.
No such file	Indicates that the specified file doesn't exist in the resident Diagnostic/ Utility Disc Directory. Check for misspelling or try another Diagnostic/ Utility Disc.

Not a Diagnostic/ Utility Disc	Indicates an attempt was made to store a file onto a disc other than a Diagnostic/Utility Disc.
Invalid Command or Input	The command requested or parameters following it do not conform to the required command structure. Execute an LC command or refer to the command description to ascertain the correct format.
Abort!! System not usable	The last operation resulted in an irrecoverable error. Verify correctness of the last operation. Cold load to attempt restart.
SYSTEM FAILURE	See Firmware Traps, Section 5.1 of this document.
No File in memory	A SAVE Command was attempted when no valid file is resident in memory. See LOAD command.
File Protected	An attempt was made to PURGE a file which has been designated protected. See CHANGE Command for changing protected status.
Pack Aborted!!	An irrecoverable disc error occurred during the PACK Command.
Invalid Revision	The revision input did not meet the syntax requirement. See LF Command for the expected format.
SEEK/READ/WRITE FAILURE!!	A disc error occurred while accessing the the system disc. This message will be preceded by a message indicating the two word status (in hex) returned by the disc as follows: SEEK/READ/WRITE STATUS=!XXXX !XXXX
Disc is Write Protected or not in the drive.	A disc access was attempted when a diskette was not in the drive. Also, a a disc write attempt to a write protected disc will produce this message.
Not a Printer device	A CHANGEIO command attempted to designate a device which doesn't identify as a supportable printer.

Not an HP 7902 Disc	A CHANGEIO command attempted to designate a device which doesn't identify as a HP 7902 Disc.

-hp-

SECTION 441A

AID LANGUAGE

EXTERNAL REFERENCE SPECIFICATION

Program Revision: 01.XX

References:

DUS Summary [Handbook Section 140]
DUS ERS [Handbook Section 440]

NOTICE

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

Hewlett-Packard Company
19447 Pruneridge Ave., Cupertino, California 95014 U.S.A.

Printed in U.S.A.

TABLE OF CONTENTS

[Letter refers to Handbook Section number: 441A, 441B, etc.]

3.11	LC	list commands
3.12	LF	list files
3.13	LIST	list statements or data
3.14	LOAD	load AID program
3.15	LOOP	set loop flag
3.16	LOOPOFF	clear loop flag
3.17	MODIFY	modify statement
3.18	PURGE	purge file
3.19	REN	renumber program
3.20	RST	reset control flags
3.21	RUN	execute program
3.22	SAVE	save AID program
3.23	SEPR	suppress error print
3.24	SEPS	suppress error pause
3.25	SET	new statement number
3.26	SNPR	suppress non-error print
3.27	SNPS	suppress non-error pauses
3.28	SO	shut off streaming
3.29	TEST	specify test section to execute

D - 4.0 AID Statements (Non I/O)

4.1	ASSIGN	store data buffer
4.2	BUMP	increment pass counter
4.3	CB	compare buffers
4.4	. (Comment)	comment entry
4.5	DB	define buffer
4.6	DELAY	suspend execution
4.7	ENABLE	allow error reporting
4.8	END	cease execution
4.9	EPAUSE	error pause
4.10	EPRINT	error print and pause
4.11	FILENAME	declare user file
4.12	FOR-STEP-UNTIL	start FOR-NEXT sequence
4.13	GOSUB	go to subroutine
4.14	GOTO	branch to statement
4.15	IF-THEN	conditional branch to statement
4.16	IFN-THEN	conditional branch to statement
4.17	INPUT	get operator input
4.18	INPUTB	get operator input
4.19	LET	assignment
4.20	LOOPTO	conditional loop branch
4.21	LPOFF/LPON	control print to line printer
4.22	NEXT	end of FOR-NEXT sequence
4.23	NOCHECKS	suppress error checking
4.24	PAGE	page to line printer
4.25	PAUSE	non-error pause
4.26	PPRINT	non-error print and pause
4.27	PRINT	non-error print
4.28	PRINTEX	error print
4.29	RANDOM	get random number
4.30	READCLOCK	read system clock count
4.31	READFILE	read data from file
4.32	RETURN	return from subroutine call
4.33	SECTION	test section execution decision

```
   4.34 SPACE                 space one line to line printer
   4.35 SPACESOFF/SPACESON    control leading spaces print
   4.36 STARTCLOCK            initiate system clock
   4.37 SUPPRESS              stop error reporting
   4.38 WRITEFILE             write data to file
   4.39 ZEROESOFF/ZEROESON    control leading zeroes print

E - 5.0  Special Characters
   5.1  .  (Comment)
   5.2  CONTROL H
   5.3  CONTROL X
   5.4  PARENTHESES
   5.5  " "
   5.6  !
   5.7  %
   5.8  PRINT SPACING
   5.9  )
   5.10 & (Ampersand)
   5.11 ;
   5.12 CONTROL Y  (ATTENTION)
   5.13 ? or ??
   5.14 ,  (Comma)
   5.15 /(Slash)
   5.16 CTRL-Shift (Hold up listing)

F - 6.0  Operators
   6.1  :=
   6.2  *
   6.3  /
   6.4  +
   6.5  -
   6.6  NOT
   6.7  =
   6.8  <>
   6.9  < or <= or > or >=
   6.10 AND
   6.11 OR
   6.12 XOR
   6.13 MOD
   6.14 LSL or LSR
   6.15 ASL or ASR
   6.16 CSL or CSR
   6.17 Special Relational Operators (NE, EQ, LT, GT, LE, GE)

G - 7.0  Reserved Variables
   7.1  BADINTP               illegal interrupt information
   7.2  CHANNEL               device channel under test
   7.3  CONCHAN               Console channel device
   7.4  DEVICE                device number under test
   7.5  FILEINFO              file information
   7.6  FILELEN               length of file
   7.7  GOPARAMn              GO Command parameters
   7.8  INDEX                 CB statement result
   7.9  INPUTLEN              last character length input
```

7.10	MAXMEMORY	memory space available
7.11	NEWTEST	TEST Command indicator
7.12	NOINPUT	non-error print indicator
7.13	NORESPONS	I/O operation condition code info
7.14	OFFSET	control RETURN point
7.15	PASSCOUNT	program pass counter
7.16	RUNPARAMn	RUN Command parameters
7.17	SECTION	currently executing Test Section
7.18	SECTIONSn	Test Section indicators
7.19	STATENUM	last executed FUNCTION-calling statement
7.20	STEP	currently executing Step
7.21	TIMEOUT	I/O timeout control
7.22	TRUE or FALSE	-1 for TRUE, 0 for FALSE

H - 8.0 AID Statements (I/O - Non-Channel-Program Type)

8.1	ADDRESSON/ADDRESSOF	control bit 4 word 4 of Read/Write
8.2	DSIO	define Channel program
8.3	COPY	duplicate Channel program
8.4	CPVA	define CPVA buffer
8.5	ESIO	end Channel program definition
8.6	HIOP	halt Channel program execution
8.7	INIT	initialize I/O Channel
8.8	IOCL	clear all I/O Channels
8.9	ION/IOFF	control interrupt system
8.10	LOCATE	find Channel program instruction
8.11	PROC	control I/O with/without wait
8.12	RDRT	read DRT word
8.13	RIOC	read I/O Channel
8.14	RMSK	read interrupt mask info
8.15	ROCL	roll call I/O Channels
8.16	RSIO	run Channel program
8.17	RSW	read Switch Register
8.18	SMSK	set interrupt mask
8.19	UPDATEON/UPDATEOFF	control bit 5 word 4 of Read/Write
8.20	WIOC	write I/O Channel

I - 9.0 AID Statements (Channel Program Type)

9.1	CHP	command HP-IB
9.2	CLEAR	clear I/O Channel
9.3	DSJ	device specified jump
9.4	IDENT	identify device
9.5	IN	interrupt halt/run
9.6	JUMP	Channel program branch
9.7	RB	read burst
9.8	RDMAB	read burst with DMA
9.9	RDMAR	read record with DMA
9.10	RMW	read-modify-write
9.11	RR	read record
9.12	RREG	read register
9.13	WAIT	suspend for device service request
9.14	WB	write burst
9.15	WDMAB	write burst with DMA
9.16	WDMAR	write record with DMA
9.17	WR	write record

LIST OF FIGURES

1.0 INTRODUCTION TO AID

AID (Advanced Interactive Diagnosis) is a stand alone program, independent of operating systems, which interprets operator statements and commands with emphasis on easy communication with I/O devices. HP AID is designed for use on an HP 300 or HP 3000/33 System containing at least 256K bytes of memory with a device to load AID and a keyboard console for operator interaction.

HP AID consists of statements for writing programs and commands for controlling program operation. It is the intent of HP AID to provide the operator with the ability to communicate with many different I/O devices in an interpretive level language while maintaining execution efficiency as if the program was written in a lower level language.

This ERS assumes the operator is familiar with the keyboard Console and terms related to the console (e.g. ENTER).

For documentation purposes, throughout this ERS, characters output by the computer are underlined to distinguish them from user input.

All references to ENTER will be considered synonymous with similar keys or controls on other consoles or specialized consoles (i.e. the ENTER key on the IDS performs the same function as return/line feed on most consoles).

This ERS makes reference to the Diagnostic/Utility System which is documented in the Diagnostic/Utility System ERS [Section 440].

1.1 Special Keys

RETURN or ENTER	Must be pressed after every command and statement. It terminates the line and causes the Console to return to the first print position.
linefeed	Advances the Console one line.
CTRL	When pressed simultaneously with another key, converts that key to a control character that is usually non-printing.
CTRL H (Bs) or BACKSPACE	Deletes the previous character in a line. The cursor is moved one space to the left.

CTRL X (Cn) or DELETE ENTRY	Cancels the line currently being typed. Three exclamation marks, a Return and Linefeed are issued to the Console (Note- May not apply to all Console types).
CTRL Y (Em) or ATTENTION	Suspends AID program execution, reports the statement number currently executing and prompts ()). See the PAUSE command for further action. CTRL Y has no significance in the entry mode except during LISTing where it causes the LISTing to terminate.

1.2 Prompt Characters

AID uses a set of prompting characters to signal to the user that certain input is expected or that certain actions are completed:

>	The prompt character for AID; an AID command or statement is expected.
?	User input is expected during execution of an INPUT(B) statement.
??	Further input is expected during execution of an INPUT statement.
!!!	A full line has been deleted with CTRL X (Note- May not apply to all Console types).

1.3 Running AID

To run AID:

(1) Bring up the Diagnostic/Utility System (DUS) from a Diagnostic/Utility Disc.

(2) Enter security code, then 'AID'

(3) AID will display its message and prompt.

1.4 AID Commands and Statements (Introduction)

1.4.1 Commands - AID Commands instruct AID to perform certain control functions. Commands differ from the statements used to write a program in that a Command instructs AID to perform some action immediately, while a statement is an instruction to perform an action only when the program is executed. A statement is always assigned a statement number; a command is not.

Commands are entered following the prompt character (>). Most commands are allowed in either the entry mode or pause mode but not both. Each command is a single word that must be typed in its entirety with no embedded blanks. Some commands have additional parameters to further define command operation.

For a complete decription of all Commands, see Section 3.0 - AID Commands.

1.4.2 Statements - Statements are used to write an AID program that will subsequently be executed. Each statement entered is limited to 80 characters and becomes part of the current program which is kept until explicitly deleted.

A statement is always preceded by a statement number. This number may be an integer between 1 and 9999 inclusive. The statement number indicates the order in which the statements will be executed. Statements are ordered by AID from the lowest to the highest statement number. Since this order is maintained by AID, it is not necessary for the user to enter statements in execution order.

Following each statement, ENTER must be pressed to inform AID that the statement is complete. AID generates a return-line feed, prints the prompt character (>) and next statement number on the next line to signal that the statement was accepted. If an error was made in the statement, AID will print an error message prior to prompting (see Error Reporting).

AID statements have a semi-free format. This means that some blanks are ignored. Imbedded blanks are not allowed in the keywords or variables, and keywords and variables must be separated by at least one blank.

```
> 30   PRINT S                    VALID
----
> 30        PRINT S               VALID
----
> 30   PRINTS                     NOT VALID
----
> 30        P R I N T S           NOT VALID
----
> 30   PRINT       S             VALID
----
```

For a complete description of all statements, see Sections 4, 8, 9 and 10 - AID Statements.

1.4.3 Changing or Deleting a Statement - If an error is made before ENTER is pressed, the error can be corrected with CTRL H (Hc) or the line may be cancelled with CRTL X (Xc) (see Special Keys). After ENTER is pressed, the error can be corrected by replacing modifying or deleting the statement.

To replace a statement, simply type the statement number followed by the correct statement.

```
To replace this statement:      > 30  PRINT X
                                ----
retype it as:                   > 40  30 PRINT S
                                ----
```

or better yet, the MODIFY command may be used:

```
                                > 30  PRINT X
                                ----
                                > 40  M30
                                ----
                                30 PRINT X
                                ----------
                                          RS
                                30 PRINT S
                                ----------
                                (enter)
                                > 40 (statement 30 is now PRINT S)
                                ----
```

To delete a statement use the following format:

```
Statement 30 is deleted by:     > 100   DELETE 30
                                -----
```

1.5 Programming

Any statement or group of statements constitutes a program.

This is an example of a program with only one statement.

```
> 100  PRINT "HELLO"
```

100 is the statement number. PRINT is the key word or instruction that tells AID the kind of action to perform. In this case, it prints the string that follows.

The statement 100 PRINT "HELLO" is a complete program since it can run with no other statements and produce a result. However, a program usually contains more than one statement.

These three statements constitute a program:

```
> 10  INPUT A,B,C,D,E
> 20  LET S:=A+B+C+D+E/5
> 30  PRINT S
```

This program, which calculates the average of five numbers, is shown in the order of its execution. It could be entered in any order if the statement numbers assigned to each statement were not changed.

This program input would execute exactly like the program above:

```
> 10  20  LET S:=A+B+C+D+E/5
> 30  10  INPUT A,B,C,D,E
> 30  PRINT S
```

1.6 Listing a Program

The LIST command can be used to produce a listing of the statements that have been accepted by AID:

```
> 40  LIST
----
  10  INPUT A,B,C,D,E
  ------------------
  20  LET S:=A+B+C+D+E/5
  ----------------------
  30  PRINT S
  -----------
> 40
----
```

Note that the prompt character (>) is not printed in the listing, but is printed when the list is complete to signal that AID is ready for the next command or statement.

Any LIST may be terminated with CTRL Y or ATTENTION.

Refer to the LIST Command (Section 3.0) for other listing functions.

1.7 Running a Program

After a program is entered it can be executed with the RUN command. RUN will be illustrated with two sample programs.

The first program contains one statement:

```
> 10  PRINT "HELLO"
----
```

When executed, the string HELLO is printed:

```
> 20  RUN
----
HELLO
-----
END OF AID USER PROGRAM
-----------------------
> 20
----
```

When the present AID program is done executing AID reports with "END OF AID USER PROGRAM" before prompting in the entry mode.

The second sample program averages a group of five numbers. The numbers must be input by the user:

```
> 10  INPUT A,B,C,D,E
> 20  LET S:=A+B+C+D+E/5
> 30  PRINT S
```

Each of the letters following the word INPUT, and separated by commas, names a variable that will contain a value input by the user from the Console. When the program is run, AID signals that an input is expected by printing a question mark. The user enters the values, separated by commas, after the question mark.

EXAMPLE:

```
> 40  RUN
? 7,5,6,8,9
```

AID prints the result:

```
7
END OF AID USER PROGRAM
> 40
```

See the RUN Command (Section 3.0) for further details.

1.8 Deleting a Program

The program that has been entered may be deleted with the EP (Erase Program) command.

On the previous page, the first program entered was 10 PRINT "HELLO". After it has run, it should be erased before entering the next program, otherwise both programs will run, as one, when RUN is commanded (i.e. they will run in the order of their statement numbers).

For example:

```
> 10  PRINT "HELLO"
> 20  INPUT A,B,C,D,E
> 30  LET A:=A+B+C+D+E/5
> 40  PRINT S
> 50  RUN
HELLO
? 7,5,6,8,9
-
7
-
END OF AID USER PROGRAM
> 50
```

To avoid confusing results, the following sequence should be used:

After entering and running:

```
> 10  PRINT "HELLO"
> 20  RUN
HELLO
END OF AID USER PROGRAM
```

The program is erased:

```
> 20  EP
Confirm you want to ERASE the
current program (Y or N)? Y
Program Erased
> 10
```

The user's resident program area is now cleared and another program can be entered:

```
> 10  INPUT A,B,C,D,E
> 20  LET S:=A+B+C+D+E/5
> 30  PRINT S
> 40  RUN
? 15,25,32,11,27
 
22
 
END OF AID USER PROGRAM
> 40
```

Unless this program is to be executed again, it can now be erased and another program entered. See the EP Command (Section 3.0) for further details.

1.9 Documenting a Program

Comments can be inserted in a program with the period(.) Special Character. Any comment typed after a period will be printed in the program listing but will not affect program execution. Comments cannot be continued on the next line, but as many comments can be entered as are needed.

The previous sample program to average 5 numbers can be documented with several comments:

```
> 40  5. THIS PROGRAM AVERAGES
> 40  7. 5 NUMBERS
> 40  10 INPUT A,B,C,D,E  .GET THE VALUES
> 40  25.S CONTAINS THE AVERAGE.
```

The statement numbers determine the position of the comments within the existing program. A list will show them in order:

List of sample program including comments:

```
) 40 LIST
 5  . THIS PROGRAM AVERAGES
 7  . 5 NUMBERS
10   INPUT A,B,C,D,E  .GET THE VALUES
20   LET S:=A+B+C+D+E/5
25   .S CONTAINS THE AVERAGE
30   PRINT S
) 40
```

When executed, the program will execute exactly as it did before the comments were entered. See the comment statement (SECTION 4.0) or the period (.) Special Character (SECTION 5.0) for further details.

1.10 AID Operator Mode State Diagram

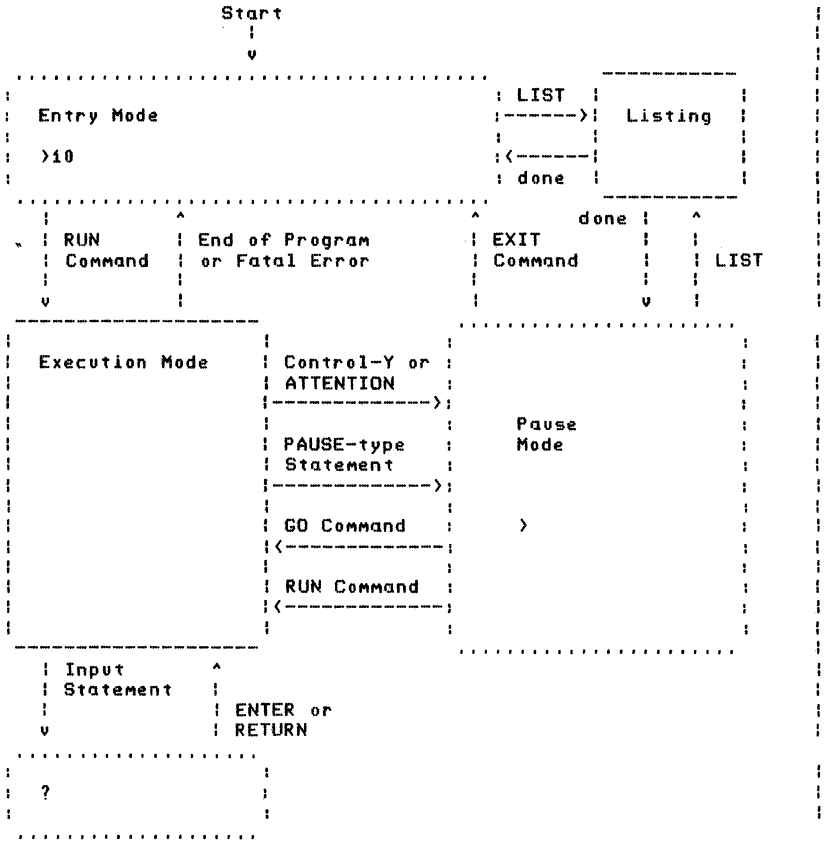


Figure 1.1 -- AID Operator Mode State Diagram

-hp-

2.0 ESSENTIALS OF AID

This section will explain some of the ground rules for handling constants, variables and strings. Also included are sections covering the basic elements of the Operators and Reserved Variables. For more precise definitions of the items covered, refer to the sections covering Special Characters, Operators, and Reserved Variables.

2.1 Expressions

An expression combines constants and variables with operators in an ordered sequence. Constants and variables represent integer values and operators tell the computer the type of operation to perform on those integer values.

Some examples of expressions are:

P + 5 / 27	P is a variable with an assigned value. 5 and 27 are decimal constants. The slash (/) is the divide operator. If P = 49, the expression will result in the value 2.
N - R + 5 - T	N, R, and T contain assigned values. If N = 20, R = 10, and T = 5, the value of the expression will be 10.

There is no operator hierarchy and evaluation of expressions is executed from left to right.

2.2 Constants

A constant is either a numeric or a byte.

Numeric Constants: A numeric constant is a positive or negative integer including zero. It may be written in any of the following three forms:

*As a decimal integer	- a series of digits with no decimal point.
*As an octal integer	- a series of digits (but not 8 or 9) preceded by a percent (%) symbol.
*As a hexadecimal integer	- a series of digits or letters (A - F only) preceded by an exclamation mark(!).

Examples of Decimal Integers:

(Range is 0 (= INTEGER (= 65536)

```
-1472   (unary negate operation)
+6732   (or 6732)
0
19
65536 (or -1)
```

Examples of Octal Integers:

(Range is 0 (= INTEGER (= %177777)

```
%1472
%6732
%17
-%20  (OR % 177760)
```

Examples of Hexadecimal Integers:

(Range is 0 (= INTEGER (= !FFFF)

```
!F
!23
!A     (NOTE:  A represents the value 10, not the
       variable A) (or !FFEA)
-!16
```

Example of a byte constant:

"A" or "5" or "!"

2.3 Variables

A variable is a name to which a value is assigned. This value may be changed during program execution*. A reference to the variable acts as a reference to its current value. Variables are represented by a single letter from A to Z.

A variable always contains a numeric value that is represented in the computer by a 16-bit word.

Variables may be manipulated as decimal, octal, or hexadecimal. However, variable type designations(i.e. ! or %) would be used in input and output (e.g. INPUT, PRINT) operations only.

A decimal variable is identified by the absence of a % or ! preceding it:

G, +G, and -G are decimal variables.
%G or !G are not decimal variables.

An octal variable is identified by a preceding percent (%) symbol:

%A and %B are octal variables.

A hexadecimal variable is identified by a preceding exclamation (!) mark:

!K, !G, !Z are hexadecimal variables.

* All variables are set to zero when a LOAD or RUN command is entered.

2.4 Data Buffers

Data Buffers are identified by duplicate letters (AA - ZZ) and are manipulated as one dimensional INTEGER arrays with the 16-bit integer row value defined within parentheses. This row value starts at 0 and may be represented by a variable A through Z, any Reserved Variable and constants only. Examples of Data Buffer elements:

AA(4), CC(400), DD(G), SS(INDEX)

Data Buffers may be declared up to the user memory available (see MAXMEMORY Reserved Variable).

Once a buffer is declared with a DB statement* it may be manipulated as a variable in the form of a decimal, octal or hexadecimal integer**:

AA(2)	is a decimal buffer element.
%BB(200)	is an octal buffer element.
!FF(1)	is a hexadecimal buffer element.

* If a buffer is not initialized with data the content of any element is indeterminate.

**The octal or hexadecimal notation would be used only in INPUT and PRINT type statements.

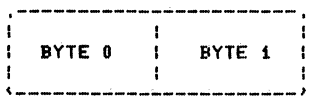
2.5 Strings and String Buffers

2.5.1 Strings - STRINGS are defined as any number of ASCII characters enclosed by quotation marks (i.e."strings"). Any ASCII character (except the quotation mark) is allowed within the string.

2.5.2 String Buffers - STRING BUFFERS are byte-oriented one-dimensional arrays used to manipulate STRINGS. These buffers are identified by duplicate letters (AA to ZZ) preceded by an ampersand (&) and are limited to the available user memory (see MAXMEMORY Reserved Variable). The element of a buffer is enclosed in parentheses and defines the byte to be manipulated. This element may be represented by a variable A through Z, a Reserved Variable or constant only. Examples of STRING BUFFER elements are:

&AA(5)	identifies byte 6 of buffer &AA (index 0 is the first element)
&CC(20)	identifies byte 21 of buffer &CC
&GG(X)	identifies byte X of the buffer &GG

Bytes are packed left-justified so that word one of a buffer contains:



STRINGS within STRING BUFFERS may be altered by using starting and ending byte indicators:

&AA(STARTING BYTE, ENDING BYTE)

The following examples will display some of the rules in manipulating STRING BUFFERS:

```
> 10  PRINT &AA(10)              .PRINT BYTE 10 OF THE &AA BUFFER
----
> 20  PRINT &AA(10, 20)          .PRINT BYTES 10 THROUGH 20 OF &AA
----
> 25  .ANY EXPRESSION RESULT MAY BE STORED INTO A BYTE
----
> 30  LET &AA(2):=B+%60
----
> 35  .ONLY SINGLE CHARACTER STRINGS ARE ALLOWED IN AN EXPRESSION
----
> 40  LET &AA(4):="B"+C
----
> 45  .ALL MULTIBYTE STRING ASSIGNMENTS MUST BE OF EQUAL LENGTH
----
> 50  LET &AA(2,5):="ABCD"
----
> 55  .THE FOLLOWING STATEMENTS WOULD GENERATE ERRORS
----
> 60  LET &AA(2,3):=B+%60    .LET &AA(2,3) MUST BE STORED WITH "XX"
----
> 60  LET &AA(4);="BC"+C          ."BC" NOT ALLOWED IN EXPRESSIONS
----
> 60  LET &AA(2,6):="ABCD"       .&AA(2,6) IS EXPECTING 5 CHARACTERS
----
> 60  LET &AA(0):=&AA(1):="B"   .MULTIPLE STRING ASSIGNMENTS
----
> 60  LET &AA(2,5):=&BB(7,10):="ABCD"    .NOT ALLOWED
----
```

2.6 Operators (Introduction)

An operator performs an arithmetic or logical operation on one or two values resulting in a single value. Generally, an operator has two operands, but there are unary operators that precede a single operand. For instance, the minus sign in A-B is a binary operator that results in subtraction of the values; the minus sign in -A is a unary operator indicating that A is to be negated.

The combination of one or two operands with an operator forms an expression. The operands that appear in an expression can be constants, variables or other expressions.

Operators may be divided into types depending on the kind of operation performed. The main types are arithmetic, relational, and logical (or Boolean) operators.

The arithmetic operators are:

+	Integer ADD (or if unary, no operation)	A + B (or +A)
-	Integer Subtract (or if unary, negate)	A - B (or -A)
*	Integer Multiply	A * B
/	Integer Divide	A / B
MOD	Modulo; remainder from division	A MOD B produces the remainder from A / B

In an expression, the arithmetic operators cause an arithmetic operation resulting in a single integer numeric value.

The relational operators are:

=	Equal	A = B
<	Less Than	A < B
>	Greater Than	A > B
<=	Less Than or Equal To	A <= B
>=	Greater Than or Equal To	A >= B
<>	Not Equal	A <> B

When relational operators are evaluated in an expression they return the value -1 if the relation is found to be true, or the value 0 if the relation is false. For instance, A = B is evaluated as -1 if A and B are equal in value, or as 0 if they are unequal.

The following examples demonstrate the difference between relational operators and special relational operators in expression evaluation:

```
10 LET B:=6
20 IF 1<B<100 THEN 500
   IS EVALUATED AS
   1<6 = TRUE (-1)
 (-1)<100 = TRUE (-1)
   RESULT "TRUE"
```

```
10 LET B:=-10
20 IF 1<B<100 THEN 500
   IS EVALUATED AS
   1<-10 = FALSE (0)
   (0)<100 = TRUE (-1)
      RESULT "TRUE"
```

Notice using relational operators doesn't work in this type application. However, consider the evaluation of special relational operators (see Special Relational Operators (SECTION 6.0) regarding the Special Operators EQ, LT, GT, LE, GE and NE.):

```
10 LET B:=6
20 IF 1 LT B LT 100 THEN 500
   IS EVALUATED AS
   1<6 = TRUE (-1)
   6<100=TRUE (-1)
   TRUE AND TRUE = TRUE
     RESULT "TRUE"
```

```
10 LET B:=-10
20 IF 1 LT B LT 100 THEN 500
   IS EVALUATED AS
  1<-10 = FALSE (0)
  -10<100=TRUE (-1)
    TRUE AND FALSE = FALSE
    RESULT "FALSE"
```

The Logical or Boolean operators are:

AND	Logical "and"	A AND B
OR	Logical "inclusive or"	A OR B
XOR	Logical "exclusive or"	A XOR B
NOT	Logical complement	NOT A

Unlike the relational operators, the evaluation of an expression using logical operators results in a numeric value which is evaluated as true (non-zero but not necessarily -1) or false (0).

The Shift Operators are:

LSL or LSR	Logical Shift	X LSL n (where n is any variable or constant)
ASL or ASR	Arithmetic Shift	X ASR n
CSL or CSR	Circular Shift	X CSL n

For further descriptions of Operators, see Section 6.0.

2.7 Reserved Variables (Introduction)

AID reserves special locations for variables that may commonly be used or accessed from a known area. These locations are assigned names which become Reserved Variables. Reserved Variables may be altered or accessed as a variable (i.e. like A thru Z), however, caution must be used since some Reserved Variables are altered by commands and statements. The following list briefly describes those Reserved Variables and the operations that change them.

NORESPONS	- If>0 then altered during bad I/O operation.
BADINTP	- altered by an illegal device interrupt.
CONCHAN	- set to the system console channel device.
FILELEN	- set to file length after FILENAME.
FILEINFO	- set to file information after FILENAME.
INPUTLEN	- set to character input length during INPUT.
MAXMEMORY	- Altered during DB and BSIO/ESIO execution
TRUE	- Stored with -1 at run time
INDEX	- During a CB statement, set to -1 if the buffers compare otherwise the element number (of the first buffer) which didn't compare
PASSCOUNT	- Optionally incremented by the BUMP statement
RUNPARAM1/3	- Set to the value of any parameters passed with the RUN command otherwise 0
GOPARAM1/3	- Set to the value of any parameters passed with the GO command otherwise 0
OFFSET	- Set to 0 after a RETURN statement
NOINPUT	- Set to true with a SNPR command or false with an ENPR command
SECTIONS1/3	- Set to the appropriate bit mask combination of up to 48 section numbers input with the TEST command otherwise set to all "ones" at run time.
STATENUM	- Set to the AID program statement number of the most recently executed FUNCTION-calling statement.
NEWTEST	- Set to true if a TEST command is entered with parameters and set to false after a TEST command without parameters
SECTION	- Set to the section number of a SECTION statement (if the SECTION is executed)

All other Reserved Variables are set to zero at run time. For a description of each Reserved Variable see Section VII.

2.8 Operator Input Modes

Three modes of operator input are available. These modes, discussed next in detail, are entry, execution and pause.

2.8.1 Entry Mode Input - Anytime a program is not executing or in a pause mode, AID is in the entry mode. Entry mode is identified by a prompt (>) and the next sequential statement number.

Example: `> 10`

In this mode, the operator may enter any valid statement or command.

2.8.2 Execution Mode Input - Anytime a program is executing, there are two inputs allowed:

(1) CONTROL Y or ATTENTION- initiates a break at the end of the currently executing statement and a message identifying that statement number.

Example:

```
Break in Statement 20
---------------------
>
_
```

At this point any pause type entry may be made (see Pause Mode below).

(2) INPUT Statement Execution - When an INPUT or INPUTB statement is executed, a question mark is prompted. Any valid numeric or alpha input(s) will be accepted. Each input must be separated by a comma if multiple inputs are requested.

Example:

```
INPUT THREE NUMBERS
-------------------
? !4F,%37,10
_
```

2.8.3 Pause Mode Input - Anytime a CONTROL Y interrupt* or pause-type statement has occurred, AID prompts with (>) and no statement number. At this point the operator may enter any valid command which affects program execution or control except EP, REN, SAVE, LOAD, SET, DELETE, INC and MODIFY. Program alteration is not allowed, but the operator may display any LIST data.

For further explanations, see the operator mode state diagram (Section 1.10) or refer to the various statements and commands for input restrictions.

* An interrupt during an I/O operation is indicated by the message:

```
Internal Break in Statement 10
------------------------------
>
_
```

(Any pause mode input except LIST, PURGE, CREATE and LF may be made when this occurs)

2.9 Program Execution

After the RUN command is issued AID must do some house cleaning before turning over control to execution of the program. This may cause a slight delay in the initial pass of the resident program, but subsequent passes will not be delayed. Also, during this house cleaning, errors may be detected that could abort the program (e.g. a referenced statement number is missing).

Assuming all goes well in the house-cleaning, execution commences. If an AID error occurs during execution, the program may abort and AID will return to the entry mode.

The programmer should be aware of statements that cause large amounts of time to execute in case time is an important consideration (e.g. DB of a predeclared buffer which causes a pack of the buffer area). And, he should be aware of statements that consume large amounts of user area in case memory is a critical factor (e.g. Comments). A list of memory allocation and approximate execution times of statements is provided in Statement Memory Allocation and Execution Time Information (in SECTION II).

If the program doesn't loop it will exit by printing "END OF AID USER PROGRAM" and a prompt to indicate AID is in the entry mode.

If the program loops or runs indefinitely the only way to abort it is to interrupt(Control Y or Attention) and, after the prompt character is printed, enter the EXIT command.

2.10 Error Reporting

Three types of errors may be reported to the operator: entry mode errors, execution mode errors and program detection errors.

2.10.1 Entry Mode Errors - If an If an error is detected in a statement or command just input AID prints a circumflex (^) under, or in the vicinity of, the character that generated the error and then prints an error message.

Example:

```
) 10  LET A:=%384
----        ^
ENTRY MODE ERROR
----------------
ARITHMETIC ERROR (OVERFLOW,DIVIDE BY
------------------------------------
0, NUMBER TOO LARGE,ETC.)
-------------------------
) 10
----
```

The error message implies the octal digit was illegal.

2.10.2 Execution Mode Errors - If a failure is detected during program execution which might cause a catastrophic failure in AID, the resident program is usually aborted and an error message is reported identifying the faulty statement.

Example:

```
> 10  LET AA(4):=B
> 20  RUN
EXECUTION MODE ERROR IN STATEMENT 10
UNINITIALIZED DB
END OF AID USER PROGRAM
> 20
```

The error indicates the buffer accessed has not been declared with a DB statement.

2.10.3 Program Detection Errors - These errors are detected by the user program and will not cause a catastrophic failure in AID. Documenting the errors would be the responsibility of the program writer.

Example:

```
INPUT A LETTER
? 4
BAD INPUT, I SAID A LETTER. TRY AGAIN!!
?
```

2.11 Statement Memory Allocation and Execution Time Information

2.11.1 Statement Memory Allocation - Every statement uses a minimum of three words of user area. In addition, any parameters entered occupy the following space:

Parameter	Word(s) Used
Operators (+,-,MOD,etc.)	1/2
Special Characters (!,%)	1/2
Constants	1-1/2
Variables (A-Z)	1-1/2
Reserved Variables (PASSCOUNT, etc.)	1-1/2
Strings ("ABC")	1+ (character length/2)*
Data Buffers (AA(x))	3-1/2
String Buffers (&AA(x))	3-1/2
String Buffers (&AA(x,y))	5-1/2
Comments	1+ (character length/2)*

* Strings or comments containing character strings with more than four repetitive characters will consume less space because the repetitive string is packed into two words (i.e., "ABCDEFGH" would require four words and "********" would require two). Note also that alternate spaces are packed into bits (i.e. " A B C D" would require two words but "ABCDEFGH" would require four).

From the table above a few helpful hints arise:

- Use variables or Reserved Variables instead of buffers when possible.

- Use strings, string buffers and comments sparingly. If strings must be used, look for a trade-off in space (i.e. if a string containing more than about six characters will be used repeatedly, it might be beneficial to assign that string to a string buffer for further manipulation or printing).

- A comment following a statement text consumes three words less than a comment statement.

  Example:
```
> 10  .SAVE XYZ VALUE
> 20  LET A:=AA(4)
```

  THE FOLLOWING SAVES THREE WORDS
```
> 10  LET A:=AA(4)  .SAVE XYZ VALUE
```

- Although it isn't obvious from the table above, chaining LET statements saves a minimum of three words for each assignment and greatly enhances execution time.

Example:

```
> 10  LET A:=4
----
> 20  LET B:=5
----
> 30  LET C:=5
----
```

THE FOLLOWING SAVES SIX WORDS

```
> 10  LET A:=4,B:=5,C:=5
----
```

THE FOLLOWING SAVES SEVEN AND A HALF WORDS

```
> 10  LET A:=4,B:=C:=5
----
```

- Savings are also derived by nesting LET statements in other statements when allowed.

Example:

```
> 10  LET A:=4,B:=5.C:=6
----
> 20  FOR A STEP B UNTIL C
----
```

THE FOLLOWING SAVES SEVEN WORDS

```
> 10  FOR A:=4 STEP B:=5 UNTIL C:=6
----
```

2.11.2 Execution Times - Each statement requires about twenty machine instructions to start executing. This overhead is required for setting up certain parameters required for all statements.

Once a statement actually starts executing it may require as few as two machine instructions (e.g., SUPPRESS,ENABLE) or thousands to execute (e.g, DB, where the buffer has been defined previously).

Since the "Time to Execute" to "Time of Execution" ratio of most statements is relatively high, it would behoove the programmer to compact multiple statements into one.

Example:

```
 > 10  .START THE XYZ TEST
 ----
 > 20  LET A:=4
 ----
 > 30  LET D:=55
 ----
 > 40  FOR A STEP 3 UNTIL D
 ----
       .
       .
       .
```

BE CONDENSED TO

```
 > 10  FOR A:=4 STEP 3 UNTIL D:=55 .START THE XYZ TEST
 ----
```

The first set of statements takes at least 96 machine instructions more to execute where:

```
Statement 10  costs    6+
Statement 20  costs   45+
Statement 30  costs   45+
                     ----
                      96+
```

Here are some more time saving hints for programming in AID:

* Comment statements cost 20 machine instructions where comments in statements cost nothing in execution (see previous example).

* FOR-NEXT loops are much faster than IF-THEN loops

  Example:

```
 > 10  FOR A:=0 UNTIL 10
 ----
 > 20  LET AA(A):=A
 ----
 > 30  NEXT 10
 ----
```

WILL EXECUTE MUCH FASTER THAN ---

```
 > 10  LET A:=-1
 ----
 > 20  LET AA(A):=A:=A+1
 ----
 > 30  IF A <= 10 THEN 20
 ----
```

* DB statements of previously defined buffers are very expensive because of the packing required for dynamic buffer allocation and should therefore be used sparingly.

Example:
```
> 10  DB AA, 20
    .
    .
    .
>100  DB AA,10  .VERY EXPENSIVE
```

HINT: If space is available use another buffer instead.

Example:
```
> 10  DB AA,20
>100  DB BB,10
```

* Chain assignments whenever possible.

Example:
```
> 10  LET A:=4
> 20  LET B:=5
> 30  LET C:=5
```

May be rewritten to save at least 70 machine instructions as:

```
> 10  LET A:=4,B:=5,C:=5
```

or even greater savings may be realized by:

```
> 10  LET A:=4,B:=C:=5
```

* Because of inter-statement overhead, transfer of control should be made to the exact destination.

Example:
```
> 10  GOTO 50
    .
    .
    .
> 50  .BEGIN XYZ TEST
> 60  SECTION 4,300
```

Although harmless in appearance, the GOTO 50 should bypass any unnecessary or non-executable comments. The most efficient code would be:

```
> 10  GOTO 60
    .
    .
> 50  .BEGIN XYZ TEST
```

3.0 AID COMMANDS

The AID Commands available to the operator are listed, in detail, in this section. The format for each command explanation is:

OPERATION NAME: General phrase of what the Command does.

MNEMONIC: The form that the Command would be called in.

DESCRIPTION: A detailed explanation of the Command's function.

ALLOWED IN: Describes whether the command is allowed in the Pause Mode, Entry Mode or both.

EXAMPLES: One or more examples using the Command.

3.1 CREATE

OPERATION NAME: Create a new file

MNEMONIC: CREATE filename, number of sectors [,revision level]

ALLOWED IN: Entry Mode or Pause Mode but not Internal Break Mode (See Pause Mode Input)

DESCRIPTION: Creates, i.e. adds to the directory of files of the Diagnostic/Utility disc, a Data file named "filename" which will be the "number of sectors" parameter long. The range on the number of sectors is 1(=sectors(=available sectors left on the disc. See the Diagnostic/Utility System ERS for further details.

EXAMPLE(S): > 10 CREATE TEST,4 (creates the Data file TEST with a length of 4 sectors).

3.2 DELETE

OPERATION NAME: Delete statement(s)

MNEMONIC: D[ELETE] first statement number[/last statement number]

ALLOWED IN: Entry Mode Only

DESCRIPTION: Removes the statement specified in first statement number from the user program. If the last statement number parameter is entered then the statements from first to last statement number are deleted.

EXAMPLE(S):

```
> 100 DELETE 20  (remove statement 20)
-----

            -or-

> 100 D30/40   (remove statements 30 through 40)
-----
```

3.3 EEPR

OPERATION NAME: Enable Error Printout

MNEMONIC: EEPR

DESCRIPTION: Enables AID to print error messages*. This is a default condition and would normally be used only after a previous SEPR Command.

NOTE: Default is error print enabled.

ALLOWED IN: Pause Mode Only

EXAMPLE(S):
```
> 110  RUN
-----
(ATTENTION)

Break in Statement 80
-----------------------
> EEPR       (ENABLE ERROR PRINTOUT)
-
```

* These messages are those contained in the EPRINT and PRINTEX Statements only.

3.4 EEPS

OPERATION NAME: Enable Error Pause

MNEMONIC: EEPS

DESCRIPTION: Enables AID to generate an error pause* after an error. This is a default condition and would normally be used only after a previous SEPS Command.

NOTE: Default is error pause enabled.

ALLOWED IN: Pause Mode Only

EXAMPLE(S):
```
> 110  RUN
-----
(ATTENTION)

Break in Statement 20
-----------------------
> EEPS       (ENABLE ERROR PAUSES)
-
```

* These pauses are those contained in the the EPRINT and EPAUSE Statements only.

3.5 ENPR

OPERATION NAME: Enable Non-Error Printout

MNEMONIC: ENPR

DESCRIPTION: Enables non-error messages* to be printed and operator response to a message to be acknowledged. This is a default condition and would normally be used only after an SNPR Command was previously entered. ENPR sets the Reserved Variable NOINPUT to false.

NOTE: Default is non-error print enabled.

ALLOWED IN: Pause Mode Only

EXAMPLE(S):

```
> 50  RUN
(ATTENTION)

Break in Statement 10
> ENPR          (Enable Non-error Print)
```

* These messages are those contained in the PPRINT and PRINT Statements only.

3.6 ENPS

OPERATION NAME: Enable Non-Error Pauses

MNEMONIC: ENPS

DESCRIPTION: Enables non-error pauses* during AID program execution. This is a default condition and would normally be used only after a SNPS command was previously entered.

NOTE: Default is non-error pause enabled.

ALLOWED IN: Pause Mode Only

```
EXAMPLE(S):       > 50  RUN
                  ----
                  (ATTENTION)

                  Break in Statement 10
                  ---------------------
                  > ENPS        (Enable Non-Error pauses again)
                  -
```

* These pauses are those contained in PPRINT and PAUSE Statements only.

3.7 EP

OPERATION NAME: Erase Program

MNEMONIC: EP

DESCRIPTION: Erases the resident AID program from memory.

ALLOWED IN: Entry Mode Only

EXAMPLE(S):

```
> 100  .LAST LINE
-----
> 110  EP
-----
CONFIRM YOU WANT TO ERASE THE CURRENT PROGRAM (Y OR N)
-----------------------------------------------------
? Y
-
PROGRAM ERASED  (If this message doesn't appear the
----------------   program is intact.)
> 10
----
```

3.8 EXIT

OPERATION NAME: Leave Program Execution

MNEMONIC: EXIT

DESCRIPTION: Stops AID program execution and returns to the entry mode. If AID is in the entry mode then EXIT returns to the Diagnostic/Utility System.

ALLOWED IN: Pause Mode or Entry Mode

EXAMPLE(S):

```
      > 50  RUN
      -----
      (ATTENTION)

      Break in Statement 30
      -------------------------
      > EXIT
      -
      END OF AID USER PROGRAM
      --------------------------
      > 50                  (READY FOR NEXT STATEMENT)
      -----


         -or-


      > 100 EXIT
      ------
      CONFIRM YOU WANT TO ERASE THE CURRENT PROGRAM (Y OR N)
      ------------------------------------------------------------
      ? Y (a N response will return the operator to the
      -                  AID entry mode)

      Enter Program Name
      !
```

3.9 GO

OPERATION NAME: Continue Execution

MNEMONIC: GO [G1][,[G2][,G3]]

DESCRIPTION: Causes the present AID program to continue from the point at which it paused. Up to three parameters (G1/G3) may be passed which are accessible by the program with the GOPARAM1/3 Reserved Variables (additional parameters are ignored). The parameters are delimited by commas and are assumed to be decimal integers unless preceded by a % or ! (see Special Characters). Default parameters are assigned the value 0.

ALLOWED IN: Pause Mode Only

EXAMPLE(S):

```
.
.
.
> 100  RUN
-----
DISC NOT READY, READY DISC AND CONTINUE
-------------------------------------
> GO      (PROGRAM EXECUTION CONTINUES GOPARAM1
-          THROUGH GOPARAM3 EQUAL 0)
           or

> GO,,2   (THE THIRD PARAMETER (GOPARAM3) IS 2
-          AND THE REST ARE 0)

           or

> GO 8    (THE FIRST PARAMETER (GOPARAM1) IS 8)
-
```

3.10 INC

OPERATION NAME: Change Statement Increment

MNEMONIC: INC X

DESCRIPTION: Allows the operator to change the statement increment value without renumbering (see REN Command). The new value X will take effect after a valid statement is entered with a number greater than or equal to the existing statement number.

ALLOWED IN: Entry Mode Only

EXAMPLE(S):
```
> 10  LET A:=4
> 20  INC 1
> 20  GOSUB 200
> 21     (Note- increment is by one and not 10)
```

3.11 LC

OPERATION NAME: List Commands

MNEMONIC: LC

DESCRIPTION: Lists the commands that are available in AID. The entry mode and pause mode commands are listed depending on the mode AID is in at the time of the LC command.

ALLOWED IN: Pause Mode or Entry Mode

EXAMPLE(S):
```
> 10 LC     (Lists the entry mode AID commands)
```
or
```
Break in Statement 50
> LC        (Lists the Pause mode AID commands)
```

3.12 LF

OPERATION NAME: List Files

MNEMONIC: LF [P[RINTER]]

DESCRIPTION: Lists the files that reside in the Diagnostic/Utility Disc directory. For further information refer to the Diagnostic/Utility System ERS.

ALLOWED IN: Entry Mode or Pause Mode but not Internal Break Mode (See Pause Mode Input)

EXAMPLE(S): `> 10 LF` (See Diagnostic/Utility System ERS for printout information)

3.13 LIST

OPERATION NAME: LIST

MNEMONIC: L[IST] [P[RINTER]] [DATA TYPE] [statement number]
[R]
[V]
[B]
[C]

ALLOWED IN: Entry Mode or Pause Mode but not Internal Break Mode (See Pause Mode Input)

DESCRIPTION: Will print the information requested to the console device. If the optional [PRINTER] is entered the LIST will be printed on the printer device. If DATA TYPE is specified the listing will be in that type (i.e. ! for hex, % for octal else decimal). Any LIST may be terminated with CTRL Y or ATTENTION.

If using the IDS as console, hold down CTRL and Shift keys simultaneously to temporarily interrupt the listing. It will resume when you release the keys. (This is an IDS firmware feature it is not provided by the AID interpreter.)

Listing formats are:

Entry	Meaning
LIST [x/y]	List the present AID program. x causes a one line list of statement x. y causes a multi-line list of statements x through y.
LIST C	List the value of PASSCOUNT.
LIST R [,x]	List the Reserved Variables. If x is entered then list only that Reserved Variable.

WARNING

The reserved variables VALUE1 to VALUE6 and NAME1 to NAME6 contain information that is pertinent only to the use of the FUNCTION statement.

Entry	Meaning
LIST V [,x]	List the variables as follows: If x is not entered then list all variables (A - Z). If x is entered then list only that variable.
LIST B [,x,y/z]	List Buffers as follows: If only B is entered, then list all buffers and their lengths in the order of the statement numbers where a DB or BSIO occurs. If x is entered, list the entire contents of buffer x (If x is a string buffer then list in ASCII with a header that designates the character numbers). With data buffers if y is entered, list only that element of buffer x. If z is entered, list all elements of buffer x from y to z.

EXAMPLE(S): SAMPLE PROGRAM LIST

```
> 60  LIST
> 10  .XYZ DIAGNOSTIC
> 20  .WHAT
> 30  .A
> 40  .FUNNY
> 50  .PROGRAM
> 60
```

SAMPLE VARIABLE LIST

```
> 110  RUN

(ATTENTION)

Break in Statement 10

> LIST!V,A

A = !F6

> LIST%V,F

F = %366

> LIST V

A = 246 B = 10 C = 43 D = 4 . . .
. . . . Z = 94
```

SAMPLE DATA BUFFER LIST

```
> 200  RUN

(ATTENTION)

Break in Statement 40

> LIST B

STATEMENT  NAME  SIZE

40          AA    20   (AA is 20 words long)
100         &BB    6   (&BB is 6 bytes long)
150         DD   *SIO* (DD is declared as BSIO DD. It's
                        length is indeterminate)

> LIST B,AA             . Will list the 20 elements of AA

AA(0) = 44  26 . . . . . . . . . . . . . . . . . . . . 13
AA(8) = 76  14 . . . . . . . . . . . . . . . . . . . . 10
AA(16) = 5  10  77  31

>LIST B,AA,1/3   . Will list elements 1-3 of AA

AA(1) = 26  14    4
```

```
      >LIST PRINTER B  (Will list all presently defined
      -                      buffers on the Printer Device)
```

SAMPLE STRING BUFFER LIST

Any character outside the range !20(=character value(!7E will be replaced with a circumflex (^) for continuity in listing (i.e. characters 20 and 21 in the following example are a carriage return and a linefeed).

```
      >LIST B,&BB       (Will list a header which identifies
                          each character position in the string
                          in increments of 70 (i.e. in the
                          following example the character D is
                          in the 70th character position) and
                          then lists the contents of the &BB
                          buffer)

    0         10         20  .....................    60        69
    +         +          +   .....................    +         +
    ------------------------------------------------------------
    JKLMNOPQRSTUV          ^^
DEF
```

3.14 LOAD

OPERATION NAME: Load Program

MNEMONIC: LOAD filename

DESCRIPTION: Allows the operator to load an AID program from disc (see the SAVE command). Any statements entered before the LOAD are erased and when the program is loaded AID responds with a normal prompt with the next sequential statement number following the loaded program.

ALLOWED IN: Entry Mode Only

EXAMPLE(S): Assume the AID program on the disc ends at statement 1270.

```
      > 110  LOAD TESTPROG (INITIATES A READ FROM THE
      -----                        DISC VIA DUS)
```

```
CONFIRM YOU WANT TO ERASE THE PROGRAM (Y OR N)
? Y              (A "Y" RESPONSE WILL ERASE THE
--                CURRENT PROGRAM AND LOAD THE NEW
                  PROGRAM, AND A "N" RESPONSE WILL
                  CAUSE NO ACTION TO OCCUR).
Program Loaded
The Next Available Statement Number is
> 1280

(LOAD SUCCESSFUL.  THE AID PROGRAM TESTPROG ON
 DISC IS NOW IN MEMORY AND ANY VALID STATEMENT
 OR COMMAND MAY BE ENTERED).
```

3.15 LOOP

OPERATION NAME: Set Loop Flag

MNEMONIC: LOOP

DESCRIPTION: Sets a LOOP flag that, during program execution, will cause a LOOPTO statement branch to occur (See the LOOPTO statement). See the LOOPOFF command for resetting this flag.

ALLOWED IN: Pause Mode Only

EXAMPLE(S):

```
> 100 SECTION 1,200
      .
      .
> 200 SECTION 2,500
      .
      .
> 500 LOOPTO 100 .Branch to Section 1 if LOOP commanded
```

3.16 LOOPOFF

OPERATION NAME: Clear Loop Flag

MNEMONIC: LOOPOFF

DESCRIPTION: Clears the LOOP flag that was set by the LOOP command. See LOOP command.

ALLOWED IN: Pause Mode only.

```
(ATTENTION)
Break in Statement 200
> LOOPOFF      (clear LOOP flag meaning exit the
                 AID program normally upon
                 completion)
```

3.17 MODIFY

OPERATION NAME: Modify Statement

MNEMONIC: M[ODIFY] Statement Number [/Statement Number]

DESCRIPTION: Provides a means of editing the ASCII text of a statement. When the MODIFY command is entered with an existent statement number AID lists the statement. Any character editing may now be done by entering a key letter under the column to be edited. This editing feature allows inserting, replacing or deleting characters. After the edit is complete the operator may delete the old statement number and add the new by simply pressing ENTER, or he may leave the old statement intact and add the new by entering "J" (meaning JOIN). If more than one edit type is entered only the first edit type is acknowledged. Any modify may be aborted by entering "A".

ALLOWED IN: Entry Mode Only

```
EXAMPLE(S):
      > 100  M10
      -------
          10 LET A:=4
                IA(0)  (INSERT A(0))

          10 LET AA(0):=4
             RFOR        (REPLACE LET WITH FOR)
                                  ---      ---

          10 FOR AA(0):=4
             DDDD        (DELETE FOR )
                                 ----

          10 AA(0):=4
      (ENTER)             (REPLACES STATEMENT 10)
      > 100
      ------

      > 100  M30
      -------
         30 .ABC
         R50
         50 .ABC
      (ENTER)       (DELETES STATEMENT 30, ADDS STATEMENT 50)
      > 100
      -------
                      -or-
      > 100  M50
      -------
         50 .ABC
        R1
        150 .ABC
      J                (PRESERVES STATEMENT 50, ADDS STATEMENT 150)
      > 160
      ------
```

3.18 PURGE

OPERATION NAME: Purge a File

MNEMONIC: PURGE filename

DESCRIPTION: Removes the file "filename" from the Diagnostic/Utility Disc directory. See the Diagnostic/Utility System ERS for details.

ALLOWED IN: Entry Mode or Pause Mode but not Internal Break Mode (See Pause Mode Input)

EXAMPLE(S):

```
) 10 PURGE TEST  (Remove the file TEST from the
                    directory)
```

3.19 REN

OPERATION NAME: Renumber Statements

MNEMONIC: REN [c]
where c=(statement multiple)=1 and default is 10)

DESCRIPTION: Renumbers the existing statements as specified by the statement multiple. If the renumbering will exceed 9999 an error is reported and a new number must be entered. All references to Statement numbers are also changed to reflect the new Statement numbers.

ALLOWED IN: Entry Mode Only

EXAMPLE(S):

```
) 10  . . .
) 20  GOTO 30
) 30  PAUSE
) 40  REN     (DEFAULTS TO STATEMENT INCREMENTS OF
               10 WHICH MEANS THE PROGRAM DOESN'T
) 40  LIST     CHANGE IN THIS EXAMPLE)
) 10  . . .
) 20  GOTO 30
) 30  PAUSE
) 40  REN3
) 12  LIST
)  3  . . .
)  6  GOTO 9
)  9  PAUSE
) 12
```

3.20 RST

OPERATION NAME: Reset

MNEMONIC: RST

DESCRIPTION: Resets all execution state flags to the default state:

- Error Pause is enabled (EEPS Command)
- Error Messages unsuppressed (EEPR Command)
- Non-Error Messages unsuppressed (ENPR Command)
- Non-Error Pauses enabled (ENPS Command)

ALLOWED IN: Pause Mode Only

3.21 RUN

OPERATION NAME: Initiate Execution

MNEMONIC: RUN [P1],[, [P2][, [P3]]]

DESCRIPTION: Causes the resident AID program to initiate execution from the lowest numbered statement regardless of the state of execution. Up to three parameters (P1/P3) may be passed into the RUNPARAM1/3 Reserved Variables for use by the program (additional parameters are ignored). The parameters are delimited by commas and are assumed to be decimal integers unless preceded by a % or ! (see Special Characters). Default parameters are assigned the value 0. AID resets all variables, buffer pointers and indicators to their default values except the LOOP and TEST flags and information.

ALLOWED IN: Pause Mode or Entry Mode

EXAMPLE(S):

```
.
.
.
> 100  RUN       .RUNPARAM1 THRU RUNPARAM3 = 0

(ATTENTION)

Break in Statement 20

> RUN
-
This sequence would restart program execution

-- or --

> RUN 1,,3   (THE FIRST PARAMETER (RUNPARAM1) IS
              ASSIGNED THE VALUE 1 AND
              THE THIRD (RUNPARAM3) THE VALUE 3)
```

3.22 SAVE

OPERATION NAME: Save Program

MNEMONIC: SAVE filename [,revision level]

DESCRIPTION: Allows the operator to save the resident AID program, in binary, on the disc via DUS (also see the LOAD command). Nothing is altered in the AID program and, after the SAVE is completed, AID returns to the entry mode. If the optional revision level is entered filename will have that revision. If no revision is entered filename will be assigned a 00.00 revision level.

NOTE: If room does not exist on the diskette for the file, the message "Insufficient disc space" is displayed. Since going to DUS will cause the current AID program to be lost, follow this recovery procedure:

(1) Insert another Diagnostic/Utility diskette which has more space

(2) SAVE the current AID program on the second diskette

(3) Re-insert the original Diagnostic/Utility diskette

(4) Use PACK command to attempt to open-up space

(5) Re-insert the second Diagnostic/Utility diskette

(6) LOAD the program

(7) Re-insert the first Diagnostic/Utility diskette

(8) SAVE the program

ALLOWED IN: Entry Mode Only

EXAMPLE(S):
```
> 1280  SAVE TEST, 01.02
------
PROGRAM SAVED  (ANY OTHER MESSAGE INDICATES
-------------    NO SAVE OCCURRED)

> 1280       (SUCCESSFUL SAVE!  ANY VALID COMMAND
------        OR STATEMENT MAY BE ENTERED)
```

3.23 SEPR

OPERATION NAME: Suppress Error Printout

MNEMONIC: SEPR

DESCRIPTION: Suppresses error messages and error pauses* until an EEPR or RST command is acknowledged.

NOTE: Default is error print enabled.

ALLOWED IN: Pause Mode Only

EXAMPLE(S):
```
> 110  RUN
-----
(ATTENTION)

Break in Statement 20
---------------------

> SEPR
_
```

* These error messages and error pauses are those contained in the EPRINT and PRINTEX Statements only.

3.24 SEPS

OPERATION NAME: Suppress Error Pause

MNEMONIC: SEPS

DESCRIPTION: Suppresses error pauses* from occurring. The RST and EEPS Commands will override this condition.

NOTE: Default is error pause enabled.

ALLOWED IN: Pause Mode Only

EXAMPLE(S):
```
> 110  RUN
-----
(Attention)
Break in Statement 50
---------------------
> SEPS
_
```

* These pauses are those contained in the EPRINT and EPAUSE statements only.

3.25 SET

OPERATION NAME: Set New Statement Number

MNEMONIC: SET Statement Number

DESCRIPTION: Allows the operator to set the current statement number to any valid statement number. If an existing statement number is encountered while sequencing because of the SET command a warning message is issued which informs the operator that a valid statement entry will delete the existing statement.

ALLOWED IN: Entry Mode Only

EXAMPLE(S):

```
> 10  LET A:=4
> 20  INC 1
> 20  SET 8
>  8  LET B:=4
>  9  GOSUB 50
**WARNING - NEXT STATEMENT ALREADY EXISTS**
> 10  SET 20  (RETURN TO ORIGINAL STATEMENT ENTRY
              STATEMENT 10 IS NOT ALTERED)
> 20
```

A typical application would be:

```
> 50  GOSUB 900
> 60  SET 900
>900  .BEGIN SUBROUTINE
  .
  .
  .
> 1010  RETURN    .END SUBROUTINE
> 1020  SET 60
>   60  (RETURN TO ORIGINAL MAIN PROGRAM ENTRIES)
```

3.26 SNPR

OPERATION NAME: Suppress Non-Error Printout

MNEMONIC: SNPR

DESCRIPTION: Suppress non-error messages* on the Console. The RST and ENPR Commands will override SNPR. SNPR sets the Reserved Variable NOINPUT to true and does not allow INPUT(B) statements to be executed.

NOTE: Default is non-error print enabled.

ALLOWED IN: Pause Mode Only

EXAMPLE(S):

```
> 110  RUN
-----
(ATTENTION)

Break in Statement 40
---------------------
> SNPR
_
```

* These messages are those contained in the PPRINT and PRINT statements only.

3.27 SNPS

OPERATION NAME: Suppress Non-Error Pauses

MNEMONIC: SNPS

DESCRIPTION: Suppresses non-error pauses* during AID program execution.

NOTE: Default is non-error pause enabled.

ALLOWED IN: Pause Mode Only

EXAMPLE(S):
```
) 110  RUN
------
(ATTENTION)

Break in Statement 40
---------------------
) SNPS
-
```

* These pauses are those found in the PPRINT and PAUSE Statements only.

3.28 SO

OPERATION NAME: Shut off streaming

MNEMONIC: SO

DESCRIPTION: Streaming is not implemented; SO acts like EXIT.

3.29 TEST

OPERATION NAME: Section Test Select

MNEMONIC:
```
TEST [+ or -][X[[/Y],Z]]
TEST ALL
```

DESCRIPTION: Allows the operator the capability of externally selecting program sections to be executed. The optional + or - adds or deletes the following test sections from the current test section bit mask; absence of the + or - deletes all existing test section bit masks before continuing. The optional slash (/) indicates inclusive sections i.e.- 3/5 means test sections 3, 4, 5. The optional comma (,) indicates separate test sections (i.e. 1,3,5 means test sections 1 and 3 and 5). Section numbers may be entered in any order but the section number must be greater

then 0 and less than 49. Whenever TEST is entered with parameters the Reserved Variables SECTIONS1/3 are set with bit masks correlating to the section numbers (see Reserved Variable SECTIONS1/3) and the Reserved Variable NEWTEST is set to true (see Reserved Variable NEWTEST). If TEST is entered without parameters the NEWTEST Reserved Variable is set to false and the bit masks in Reserved Variables SECTIONS1/3 are set to all ones. If TEST ALL is entered all Test Sections are selected (i.e. All bits in SECTIONS1,SECTIONS2 and SECTIONS3 are set).

ALLOWED IN: Pause Mode Only

EXAMPLE)S):

```
      )  TEST 1/3,5,7,9/11   (INDICATES SECTIONS 1,2,3,
      -                       5,7,9,10 AND 11 ARE SELECTED)
               or
      )  TEST 10             (INDICATES SECTION 10
      -                       IS SELECTED)
               or
      )  TEST                (SETS THE NEWTEST RESERVED
      -                       VARIABLE TO FALSE)

      )  TEST + 4            (ADD TEST 4 TO THE TEST
      -                       SECTION BIT MASK)

      )  TEST - 6            (REMOVE TEST 6 FROM THE
      -                       TEST SECTION BIT MASK)
```

See the Reserved Variables SECTIONS1/3 and NEWTEST and the statement SECTION for further examples and explanations.

4.0 AID STATEMENTS (NON I/O)

The AID statements available to the operator are listed, in detail, in this section. The format for each statement explanation is:

OPERATION NAME: General phrase of what the statement does.

MNEMONIC: The form that the statement would be called in.

DESCRIPTION: A detailed explanation of the statement's function.

EXAMPLES: One or more examples using the statement.

4.1 ASSIGN

OPERATION NAME: Assign Data to Buffer

MNEMONIC: ASSIGN data buffer(element)[,(repeat factor)],data1 [,data2].....[dataN]

DESCRIPTION: Stores data into a data buffer. The word data1 is stored into data buffer (element) and, if included, data2 is stored in data buffer (element+1), and so on through dataN, which is stored in data buffer (element+N-1). If repeat factor is included the data pattern is repeated repeat factor times. Data1 through dataN must be numeric constants.

```
> 10  DB AA,100,%55                .INITIALIZE AA TO %55
----
> 20  ASSIGN AA(50),5,10,15,20,25,30,35
----  (AA(50)=5, AA(51)=10, . . . AA(56)=35)

> 30  ASSIGN AA(10),(10),!FF
----  (AA(10) THROUGH AA(19))=!FF)

> 40  ASSIGN AA(80),(5),3,7
----  (AA(80)=3, AA(81)=7, AA(82)=3, AA(83)=7...AA(89)=7)

> 50  LET A:=80,F:=5
----
> 60  ASSIGN AA(A),(F),3,7     .IDENTICAL TO STATEMENT 40
----
```

4.2 BUMP

OPERATION NAME: Bump Pass Counter

MNEMONIC: BUMP[;][H]

DESCRIPTION: Increments the Reserved Variable PASSCOUNT (unless the H parameter is used) and then prints that pass count on the Console. The pass counter (Reserved Variable PASSCOUNT) is initialized to zero whenever a RUN Command is issued. Printing may be suppressed by a SNPR Command and, if the optional semi-colon follows BUMP, no return-line feed will be issued after the pass counter value is printed. The PASSCOUNT is limited to 32767.

EXAMPLES(2):

```
> 10  BUMP H
> 20  RUN
END OF PASS 0 (NOTE- PASSCOUNT is still 0 after the
                     print because of the H paramete
     .
     .
     .

 ---or---

> 10  BUMP;
> 20  PRINT "FOUND A BUG!!"
> 30  RUN
END OF PASS 1 FOUND A BUG!!
```

4.3 CB

OPERATION NAME: Compare Buffers

MNEMONIC: CB Buffer 1, Buffer 2, Length of Compare

DESCRIPTION: Provides a fast comparison between the contents of two buffers (two string buffers or two data buffers). If the buffer areas compare, the Reserved Variable INDEX is set to -1. Otherwise, INDEX is set to the element of Buffer 1 which didn't compare (see INDEX under Reserved

Variables).

The length of the compare is in words (limit 32,767) if comparing data buffers and bytes if comparing string buffers.

EXAMPLE(S):

```
> 5   CB AA(10), BB(10), 10     . COMPARE AA(10)-AA(19)
----
> 10                            . WITH BB(10)-BB(19).
----
> 15  IF INDEX <> -1 THEN 200   . REPORT ERROR ROUTINE AT 200
----
> 20  CB &CC(5), &DD(10), 6     . COMPARE BYTES 5-10 OF &CC
----
> 25                            . TO BYTES 10-15 OF &DD
----
> 30  IF INDEX = -1 THEN 100    . IF INDEX = -1 THEN COMPARE
----
> 35                            . WAS GOOD
----
```

NOTE: If a Compare Error occurs in statement 20, the programmer is responsible for remembering that the buffer elements are offset (i.e., &CC(5) is compared to &DD(10), not &DD(5)).

4.4 . (Comment)

OPERATION NAME: Comment String

MNEMONIC: . (period)

DESCRIPTION: Allows entry of comment strings as statements or following statements. Any entry following a period (.) will be interpreted as a comment string for the pending line (the only exception is a period inside a string). Comments should be kept short, concise and used sparingly since they can only be maintained as source data thus consuming a lot of user space.

EXAMPLE(S):

```
> 10  .THIS IS
----
> 20  .A COMMENT STRING.
----
```

```
> 30  GOTO 40          .THIS IS A COMMENT STRING
> 40  PRINT "STOP.THEN GO"
                ^
              (This does not indicate a comment string)
```

4.5 DB

OPERATION NAME: Define Buffer

MNEMONIC: DB Name, Length [,assignment data]

DESCRIPTION: Declares a buffer with a two (alpha) character name (AA, BB, ...ZZ) and a buffer length up to the allowable space available* (see MAXMEMORY under Reserved Variables). The parameter length is interpreted as a numeric (0 will delete the buffer). The only assignment data allowed at declaration is a string assignment for string buffers (see example) or numeric or variable for data buffers where the entire buffer is stored with that numeric or variable value. Dynamic allocation of buffers is allowed, but may cause large overhead in execution time since existing buffers are "packed" to allow room for a new buffer. Dynamic allocation will leave the existing element values unchanged.

EXAMPLE(S):

```
> 10  DB AA, 100          .DECLARES THE BUFFER AA AS 100 WORDS
                           LONG

> 20  DB &AA, 10          .DECLARES THE STRING BUFFER &AA AS
                          .10 BYTES LONG (NOTE AA AND &AA
                          .ARE SEPARATE BUFFERS).
> 30  DB &CC,100,"START".EACH SEQUENTIAL 5 BYTE SET OF &CC
                          .CONTAINS START
                                 -----
> 40  DB CC, 100, 0      .STORES 0 IN ALL 100 ELEMENTS OF CC.

> 50  DB CC, 110         .REALLOCATE CC TO 110 WORDS
                           (FIRST 100 ELEMENTS INTACT)

> 60  DB CC, 0           .DELETES BUFFER CC
```

*A limit of 32,767 words is set for data buffers. String buffer length is limited to 65,536.

4.6 DELAY

OPERATION NAME: Delay

MNEMONIC: DELAY increment

DESCRIPTION: Provides a delay of program execution in approximately 91.43* microsecond increments. The maximum delay increment is 65,535 (5.99 seconds).

*Based on current system clock.

EXAMPLE(S):

```
> 60  DELAY 10          (SUSPENDS PROGRAM EXECUTION FOR
----                     914.3 MICROSECONDS)

> 100 DELAY 1           (SUSPENDS PROGRAM EXECUTION
-----                    91.4 MICROSECONDS)
```

EXAMPLE(S):

```
> 120 DELAY A           (SUSPEND FOR Ax91.4 MICROSECONDS)
-----
```

4.7 ENABLE

OPERATION NAME: Enable Errors

MNEMONIC: ENABLE

DESCRIPTION: Re-enables program execution error reporting previously disabled by a SUPPRESS statement or the commands SEPR and SEPS.

EXAMPLE(S):

```
> 100  ENABLE   (ANY SUBSEQUENT ERROR WILL NOW BE
-----             REPORTED DURING PROGRAM EXECUTION)
```

4.8 END

OPERATION NAME: Stop Program

MNEMONIC: END

DESCRIPTION: Indicates the end of the existing program execution. END may be used anywhere in the program and does not have to be the last statement.

EXAMPLE(S):

```
> 10  LET A:=4
> 20  PRINT A
```

The above program is identical in execution to:

```
> 10  LET A:=4
> 20  PRINT A
> 30  END
```

END may be used anywhere to terminate the program

```
>  5  LET A:=4
> 10  GOSUB 30
> 20  END                .END PROGRAM AFTER GOSUB 30
> 30  LET A:=A + 1
> 40  PRINT A
> 50  RETURN
```

4.9 EPAUSE

OPERATION NAME: Error Pause

MNEMONIC: EPAUSE

DESCRIPTION: Creates an unconditional pause in the execution of the resident program. This statement is suppressed only by the SEPS command and SUPPRESS statement. A prompt character (>) is printed on the console, the operator may enter any valid command.

```
EXAMPLE(S):       > 10  EPAUSE
                  ----
                  > 20  RUN
                  ----
                  >  (Any valid command may be entered)
                  -
```

4.10 EPRINT

OPERATION NAME: Print Error Message to Console

MNEMONIC: EPRINT [*] [string [,(or;)] [string] etc.]

DESCRIPTION: Enables data,print spacing# or strings to be output to the Console. This statement must be used to print error messages only (see PRINT for non-error messages). This statement will only be suppressed by the SEPR command and SUPPRESS statement. The optional (*) disables the pause following the print. If the Reserved Variable STEP is greater than zero the error message is preceded by a STEP number message (See Reserved Variable STEP).

EXAMPLE(S):

```
> 10  EPRINT &BB(0,7)    .&BB PREVIOUSLY SET TO "BAD UNIT"
----
> 20  EPRINT * &BB(0,7)
----
> 30  RUN
----
BAD UNIT                  CREATED BY STATEMENT 10
--------
> GO
-
BAD UNIT                  CREATED BY STATEMENT 20
--------
END OF AID USER PROGRAM
-----------------------
```

--or--

```
> 10  EPRINT "DATA WORD ";A; "IS"; !BB(J);" SHOULD BE "; !CC(J)
----
> 20  RUN
----
DATA WORD 5 IS !F8D4 SHOULD BE !F7D4
------------------------------------
```

See Print Spacing under Special Characters.

4.11 FILENAME

OPERATION NAME: Set Filename

MNEMONIC: FILENAME string buffer [,offset]

DESCRIPTION: Specifies the filename* pointed to by the string buffer parameter be used in future file access statements. The optional offset is the sector number from the start of the file to start subsequent file accesses from (default is 0). The string pointed to in this statement must contain a valid and existent filename during execution and must terminate in a space or !FF character. Also see the CREATE command and the READFILE and WRITEFILE statements and FILEINFO and FILELEN Reserved Variables.

EXAMPLE(S):

```
> 10  DB &AA,9,"FNAME123 "
----
> 20  FILENAME &AA(0)
----  (ALL FUTURE FILE REFERENCES WILL ACCESS THE FILE
       NAMED FNAME123)

      -or-

> 100 FILENAME &AA(2),5
------ (ALL FUTURE FILE REFERENCES WILL ACCESS THE FILE
       NAME AME123 STARTING FROM THE 6TH SECTOR
       I.E.-SECTOR 5 OF THE FILE)
```

* The file "filename" must reside on the Diagnostic/Utility Disc being used and must be a valid filename as specified by the Diagnostic/Utility System ERS.

4.12 FOR-STEP-UNTIL

OPERATION NAME: For-Step-Until

MNEMONIC: F[OR] assignment exp [STEP exp] UNTIL(or TO) terminator exp

DESCRIPTION: Provides a means of repeating a group of instructions between the FOR statement and a subsequent NEXT statement using a variable as a counter (the variable cannot be a string buffer element). The STEP parameter is an optional increment of the FOR variable with a default of 1. The FOR-NEXT sequence is repeated until the terminator expression value is exceeded* by the FOR variable value. FOR statements may be nested. Note that no execution occurs in the FOR statement after the initial execution. Note also that UNTIL or TO may precede the terminator expression but UNTIL will always be listed.

EXAMPLE(S):

```
> 10  FOR I: = 5 to 50  .WILL EXECUTE THE STATEMENTS
----                     .BETWEEN 10 AND 100 (46 TIMES)
       .                 .WITH I=5 THRU I=50 STEPPING
       .                 .ONE AT A TIME
> 100 NEXT 10
-----
      -or-

> 10  FOR I:=5 STEP 8 UNTIL 50
----                     .WILL EXECUTE THE STATEMENTS
       .                 .BETWEEN 10 AND 100 (6 TIMES)
       .                 .WITH I=5,13,21,29,37,45
> 100 NEXT 10
-----
      -or-

> 10  FOR I:=5 STEP B:=8 UNTIL C:=50
----                     .THIS SEQUENCE PROVIDES
       .                 .THE SAME SEQUENCE OF
       .                 .STATEMENTS AS ABOVE
> 100 NEXT 10
-----
      -or-

> 10  FOR AA(2):= -5 TO 50
----           (AA(2) WILL STEP -5,-4,-3,-2,-1,0,1...50)
       .
> 100 NEXT 10
-----
```

*If the STEP value is negative the sequence will repeat until the FOR value is less then the UNTIL value.
(Note: The FOR loop always executes at least once.)

4.13 GOSUB

OPERATION NAME: Go to Subroutine

MNEMONIC: G[OSUB] Statement

DESCRIPTION: Allows the program to enter a subroutine and then return to the next sequential statement* after the GOSUB statement. Nesting subroutines is allowed to 20 levels.

EXAMPLE(S):

```
> 10  GOSUB 500      .GO TO THE SUBROUTINE STARTING
> 20  . . .          .AT STATEMENT 500.
      .
      .
      .
> 490 GOTO 600       .JUMP AROUND THE SUBROUTINE.
> 500 LET A:=A+1     .THIS SUBROUTINE
> 510 PRINT A;       .WILL INCREMENT A
> 520 RETURN         .PRINT IT ON THE CONSOLE AND THE
                     .RETURN CONTROL TO THE STATEMENT
                     .FOLLOWING THE GOSUB WHICH CAUSE
                     .TRANSFER OF CONTROL TO 500.
```

*See the Reserved Variable OFFSET for returning to other statements.

4.14 GOTO

OPERATION NAME: GO TO (Unconditional Branch)

MNEMONIC: GOTO Statement Number

DESCRIPTION: Allows the program to branch unconditionally to another statement number.

EXAMPLE(S):
```
> 10  GOTO 50        .TRANSFER CONTROL TO STATEMENT 5
    ----
```

4.15 IF-THEN

OPERATION NAME: If-Then Control

MNEMONIC: IF exp [[SPECIAL OPERATOR exp][SPECIAL OPERATOR exp]
THEN statement number

DESCRIPTION: Allows the executing program to evaluate "exp" and if it's true (non-zero)* to transfer control to the statement number specified. "Exp" may be a simple variable, data buffer element, assignment or expression. Expressions may be separated by a special relational operator not allowed in any other expression. The allowable special operators are:

GT (greater than)
LT (less than)
GE (greater than or equal to)
LE (less than or equal to)
NE (not equal to)
EQ (equal to)

WARNING

String buffers are handled as data buffers in this mode, i.e., &AA(0):=5 would store &AA(1) with 5.

Each expression is evaluated and then tested (left to right) with the special operator. The result(s) of the special operator evaluation(s) is logically ANDed and if the overall result is true, control is transferred to the THEN statement. Up to three expressions are allowed.

EXAMPLE(S):

```
> 10  IF AA(2) THEN 50   .IF AA(2) IS TRUE (NON-ZERO) GO
----                      TO 50
> 50  IF B:=C THEN 30    .THE ASSIGNMENT IS EXECUTED THEN
----                      .EVALUATED.

> 70  IF A OR B THEN 30 .THE EXPRESSION "A OR B" IS
----                      .EVALUATED.
> 80  IF 14 LE A:=A+1 LE 20 THEN 120
----                      .TEST IF A+1 IS BETWEEN 14 AND
                          20 INCLUSIVE.

> 90  IF A:=A+1 GE B:=B+1 GE C:=C+1 THEN 200
----                      .TEST IF (A+1))=(B+1))=(C+1)

>100  IF 1 LT B LT 100 THEN 20
----                      .TEST IF B IS BETWEEN 1 & 100**.
```

* See IFN Statement for the reverse branch condition. **Note that statement 100 would not execute the same as IF 1<B<100 THEN 20 which executes as "IF(1<B)<100 THEN 20" where the result of 1<B will equal -1 or 0.

4.16 IFN-THEN

OPERATION NAME: IF-NOT-THEN

MNEMONIC: IFN exp THEN statement

DESCRIPTION: Identical to the IF-THEN statement (see IF-THEN) except the expression "exp" is tested for falsity in determining if control is passed to the label "statement". The expression value is not altered by the NOT function.

EXAMPLE(S):

```
> 10  IF 1 LE A LE 14 THEN 20
----                      .IF A IS BETWEEN 1 AND 14 GOTO 20
> 20  IFN 1 LE A LE 14 THEN 20
----                      .IF A IS "NOT" BETWEEN 1 AND 14
                          GOTO 20

        --or--

> 10  IF A THEN 20        .IF A<>0 GOTO 20
----
> 20  IFN A THEN 20       .IF A=0 GOTO 20
----
```

4.17 INPUT

OPERATION NAME: Input Data

MNEMONIC: INPUT x,[y],...[n]
I x,[y],..[n]

DESCRIPTION: Provides capability of receiving operator input from the Console and assigning that input to a variable(s). x may be a simple variable, buffer element, string buffer or Reserved Variable. When executing, input prompts with a ? or ?? to signify an input is expected (see Special Characters). Each input value must be separated by a comma. Inputs may be an ASCII character but not ! or % alone. Also change in character type will terminate input but not necessarily report an error. Additional input beyond the expected is ignored. All ASCII characters are shifted to upper case. See the Reserved Variable INPUTLEN for determining the character length of the input.

EXAMPLE(S):

```
  10  INPUT A             .VALUE INPUT FROM THE CONSOLE IS
----                      .INTERPRETED AND THEN STORED
                          .IN A

  30  INPUT AA(2)         .AA(2) WILL BE STORED WITH THE
----                      .INPUT VALUE.

  40  INPUT &BB(2,6)      .ELEMENTS 2 THROUGH 6 OF STRING BUFFER
----                      .&BB WILL READ THE FIRST 5 CHARS INPUT
                          .FROM THE CONSOLE. STRING BUFFERS MUST
                          .BE USED IF ASCII INPUT IS REQUIRED.

  50  INPUT A,B,C         .THE OPERATOR MUST INPUT THREE
----                      .NUMERIC VALUES (SEPARATED BY COMMA
                          .DELIMITERS) TO BE ASSIGNED TO A,
                          .B AND C

  60  INPUT A
----
  70  RUN
----

? %7776                   (STATEMENT 10 EXECUTION A:=%7776)
_
? !F4                     (STATEMENT 30 EXECUTION AA(2):=!F4)
_
? HELLO                   (STATEMENT 40 EXECUTION &BB(2,6):=
-                          "HELLO")
```

```
? 2,4                    (STATEMENT 50 EXECUTION A:=2, B:=4)
-
??  8                    (STATEMENT 50 MORE INPUT REQUIRED
--                        C:=8)
? B                      (STATEMENT 60 EXECUTION A:=%102)
-
```

4.18 INPUTB

OPERATION NAME: Input for buffers

MNEMONIC: INPUTB XX(N)

DESCRIPTION: This statement allows variable length numeric input into a buffer. XX(N) is the first buffer element. Commas may replace data to suppress input into that element. String buffers are not allowed.

EXAMPLE(S):

```
> 10  DB XX,7,9          .Fill XX with nines
----
> 20  FOR I:=0 UNTIL 6   .Print initial XX contents
----
> 30  PRINT XX(I);1;
----
> 40  NEXT 20
----
> 45  PRINT
----
> 50  INPUTB XX(0)       .Get input data from operator
----
> 60  FOR I:=0 UNTIL 6   .Print XX contents with input values
----
> 70  PRINT XX(I);1;
----
> 80  NEXT 60
----
> 90  RUN
----
9 9 9 9 9 9 9
? ,,2,3,,5
9 9 2 3 9 5 9
```

Note that XX(0), XX(1), XX(4) and XX(6) are not changed by the input.

4.19 LET

OPERATION NAME: Assignment

MNEMONIC: [LET] variable:= Any variable, numeric, expression or string

DESCRIPTION: Allows assignment to a variable, data buffer or stri buffer the value of any variable, numeric, expressio or string.

EXAMPLE(S):

```
> 10  LET A:=10          .A IS ASSIGNED THE VALUE DECIMAL 10.
> 20  LET C:=D+E         .C IS ASSIGNED THE SUM OF D+E.
> 30  LET AA(2):=!F      .ELEMENT 2 OF THE BUFFER AA IS ASSIGNED
                         .THE HEXADECIMAL VALUE F.

> 45  LET A:=C:=4        .MULTIPLE VARIABLE ASSIGNMENTS ALLOWED.
> 48  LET A:=4,B:=7      .MULTIPLE EXPRESSION ASSIGNMENTS
                          ALLOWED.
> 50  LET AA(4):=B       .ELEMENT 4 OF BUFFER AA IS ASSIGNED
                         .THE VALUE OF THE B VARIABLE.

> 60  LET &AA(5,9):="HELLO"
                         .&AA(5,6)=HE, &AA(7,8)=LL, &AA(9)=O
> 70  A:=10              .IDENTICAL TO STATEMENT 10*
> 80  LET A:=B<C         .A=-1 if B<C else A=0
```

*The LET keyword may be omitted but a subsequent list will display it

4.20 LOOPTO

OPERATION NAME: Conditional Loop Branch

MNEMONIC: LOOPTO label

DESCRIPTION: Causes a branch to the statement specified in label if a LOOP Command was previously issued otherwise no action occurs.

EXAMPLE(S):

```
> 100 SECTION 1,200
```

```
                .
               > 200 SECTION 2,500
               -----
                     .
                     .
               > 500 LOOPTO 100  . Go to 100 if LOOP flag is set.
               -----
```

4.21 LPOFF/LPON

OPERATION NAME: Control offline listing

MNEMONIC: LPOFF/LPON

DESCRIPTION: Print statements normally have their output directed to the Console. LPON statements may be used to direct the print output to the line printer*. LPOFF will direct the output back to the Console.

EXAMPLE(S):

```
> 10  PRINT "This will go to the Console"
----
> 20  LPON
----
> 30  PRINT "This will go to the line printer"
----
> 40  LPOFF
----
> 50  PRINT "This will also go to the Console"
----
> 60  RUN
----
```

* If no line printer exists the print will default back to the Conso

4.22 NEXT

OPERATION NAME: End of For-Next loop

MNEMONIC: NEXT x
N x

DESCRIPTION: Specifies the end of a For-Next set of statements where x must be the statement number of a respective FOR statement.

EXAMPLE(S):

```
> 10  LET J:=5
----
> 20  FOR K:=1 UNTIL 20
----
> 30  LET BB(K):=J, J:=J+5
----
> 40  NEXT 20
----
```

This set of statements would store BB(1)=5, BB(2)=10 ...BB(20)=100.

4.23 NOCHECKS

OPERATION NAME: No Checks Enabled

MNEMONIC: NOCHECKS

DESCRIPTION: Gives the programmer the ability to disable time critical execution error checks*. This statement would typically be the first statement in a "finished known good" program so that the execution overhead of programming checks is alleviated (i.e. bounds violations, uninitialized DB, etc. needn't be checked). The "checks" condition is always enabled until this statement is encountered and then "no checks" are done until execution is complete.

EXAMPLE(S):

```
> 10  NOCHECKS
----
> 20  DB AA,100          (Buffer area overflow not checked)
----
> 30  LET BB(100):=12    (Bounds and buffer declarations
----                       not checked)
```

* If a catastrophic error occurs in the "no checks" mode
  the results are unpredictable.

4.24 PAGE

OPERATION NAME: Page Eject

MNEMONIC: PAGE

DESCRIPTION: Issues a page eject to the printer device during LISTing. During execution this statement executes as a comment.

EXAMPLE(S):

```
) 100  .END OF SECTION X
) 110  PAGE
) 120  .BEGIN SECTION Y
) 130  L PRINTER 100/120
(Listing of Line Printer looks like the following)

100  .END OF SECTION X
(Page Eject)
120  .BEGIN SECTION Y
```

4.25 PAUSE

OPERATION NAME: Non-Error Pause

MNEMONIC: PAUSE

DESCRIPTION: Creates an unconditional pause in the execution of the AID user program. This statement is suppressed only by the SNPS command. After a prompt character ()) is printed on the console the operator may enter any valid command.

EXAMPLE(S):

```
) 10  PAUSE
) 20  RUN
```

```
> (Enter any valid command)
_
```

4.26 PPRINT

OPERATION NAME: Pause Print

MNEMONIC: PP[RINT] [*] string [; (or ,)] [string] (etc.)

DESCRIPTION: PPRINT is identical to the PRINT statement except after the print a pause occurs. PPRINT may be suppressed by SNPR and the pause may be suppressed by SNPS. The optional (*) will suppress the pause which follows the print. If the Reserved Variable STEP is greater than zero the message string is preceded by a STEP number message (See Reserved Variable STEP).

EXAMPLE(S):

```
> 10  LET A:=5
----
> 20  PPRINT "BAD GUY IN";2;A
----
> 30  RUN
----
BAD GUY IN 5
------------
>           (pause mode)
_

       -or-

> 10  PPRINT * "TOO LATE NOW!!"    .SUPPRESS PAUSE
----
> 20  RUN
----
TOO LATE NOW!!
--------------
END OF AID USER PROGRAM
-----------------------
> 20
----
```

4.27 PRINT

OPERATION NAME: Print to Console without Pause

MNEMONIC: PR[INT] [string] [; (or ,)] [string] etc.

DESCRIPTION: Enables data, print spacing* or strings to be output to the list device. This statement must be used to print non- error messages only (see EPRINT or PRINTEX for error message reporting). This PRINT will only be suppressed by the SNPR command. PRINT strings may be concatenated with (;) to suppress return- line feed or (,) which generates a return-linefeed.

EXAMPLE(S):

```
> 10  PRINT "A";2;"BC","DE";3;"FGH"
> 20  RUN
A  BC
DE   FGH

      -or-

> 10  DB &AA,10,"ABCDEFG"
> 20  PRINT &AA(3,6);2;&AA(0,2)
> 30  RUN
DEFG  ABC
> 30
```

* See PRINT SPACING under Special Characters.

4.28 PRINTEX

OPERATION NAME: Print Error without Pause

MNEMONIC: PRINTEX [string] [; (or ,)] [string] etc.

DESCRIPTION: PRINTEX is identical to PRINT except that it is suppressed by SEPR like EPRINT (see PRINT for further details).

EXAMPLE(S):

```
> 10  PRINTEX "ABC";"DEF";2;"GHI"
----
> 20  RUN
----
ABCDEF  GHI
----------
> 20
----
```

4.29 RANDOM

OPERATION NAME: Generate Random Numbers

MNEMONIC: RANDOM [(argument)] variable1 [,variableN]

DESCRIPTION: Generates random integers (-37,768 to 32,767) from an argument (optional) and stores them into the variables specified (variabl1 to variableN). If the arguement is not included the random sequence continues normally, otherwise the random generator is preset to the argument. The random generator will cycle through 128,563 random numbers.

EXAMPLE(S):

```
> 10  RANDOM(10)A,B
----
> 20  RANDOM(10)C,D      (NOTE THAT A=C AND B=D SINCE
----                       THE SAME ARGUMENT WAS USED)

       -or-

> 10  RANDOM A           . NO ARGUMENT
----

       -or-
```

```
> 10  RANDOM(RUNPARAM1) A  (OPERATOR PASSED AN ARGUMENT
----                          WITH RUN X)

      -or-

> 10  RANDOM AA(0),F,TIME
----                      (GENERATE THREE SEQUENTIAL
                           RANDOM NUMBERS WITH NO
                           INITIAL ARGUMENT)
```

4.30 READCLOCK

OPERATION NAME: Read System Clock Contents

MNEMONIC: READCLOCK variable

DESCRIPTION: Reads the contents of a register which contains the amount of clock intervals as specified in the STARTCLOCK statement (see STARTCLOCK Statement). Resolution is restricted to +-95% of a clock interval therefore averaging schemes should be used for critical timing measurement. This statement also stops the system clock from further interrupts.

EXAMPLE(S):

```
> 100 STARTCLOCK 10    .START 10 MILLISECOND
                         TIMER
> 110 RS10 AA          .START CHANNEL PROGRAM
> 120 READCLOCK A      .GET 10 MILLISECOND
                         INTERVAL COUNTER VALUE
                         SINCE STATEMENT 100
   .
   .
   .
```

NOTE: The amount of overhead in executing AID statements should be accounted for by the programmer.

4.31 READFILE

OPERATION NAME: Read File

MNEMONIC: READFILE buffer element,length

DESCRIPTION: Reads data from the file "filename"* and stores it into memory starting at the location of the buffer element for length words(or characters if using a string buffer)**. Any file may be accessed by this statement.

EXAMPLE(S):

```
> 10  DB &AA,7,"HOLDIT "
> 15  DB BB,10
> 20  FILENAME &AA(0)

> 30  READFILE BB(0),10 (The first 10 words of the file
                          HOLDIT are stored into the
                          buffer BB starting at element
                          zero)
```

* A valid FILENAME statement must be executed prior to executing this statement.

**If the buffer being written is a string buffer the element is rounded down to the nearest even element to maintain even word boundaries. If a "rounding" is needed the length parameter is incremented.

Example: `> 100 READFILE &AA(3),5`

This statement would read 6 bytes from HOLDIT and put them into &AA(2).

4.32 RETURN

OPERATION NAME: Return from Subroutine

MNEMONIC: R[ETURN]

DESCRIPTION: Causes a transfer of control to the next sequential statement after the last GOSUB statement executed*. If no GOSUB occurred, program execution is aborted with an error message.

EXAMPLE(S):

```
  10  GOSUB 60      .GO TO SUBROUTINE STARTING AT 60.
----
  20  . . .
----
        .
        .
        .
  60  LET A:=A+1,B:=B+1
----
  70  RETURN        .RETURNS TO STATEMENT 20
----
```

*See the Reserved Variable OFFSET for returns to other statements.

4.33 SECTION

OPERATION NAME: Section Execute Test

MNEMONIC:: SECTION x, label

DESCRIPTION: When a program is split up into sections the SECTION statement* may be used to determine whether or not to execute a particular section. The executable sections are predefined by the TEST command and/or by assigning values to the Reserved Variables SECTIONS1/3 (see Reserved Variable section for further details). When a SECTION statement is executed the Section x bit is extracted from the appropriate bit mask for SECTIONS1/3 and if set the next sequential statements are executed normally and the Reserved Variable SECTION is set to the section number. Otherwise, control is transferred to the statement specified in LABEL.

EXAMPLE(S):

```
> 10  SECTION 1, 60
> 20
          .
          .
> 50  .End of section 1
> 60  SECTION 2, 120
> 70
          .
          .
> 120  . END OF SECTION 2
```

* Do NOT confuse the SECTION statement with the SECTION Reserved Variable.

4.34 SPACE

OPERATION NAME: Line Space

MNEMONIC: SPACE [X]

DESCRIPTION: When listing a program on a printer device, generates X line spaces before the next statement. During execution this statement is treated as a comment. Default X is 1 space.

EXAMPLE(S):

```
> 10  .END OF STEP X
> 20  SPACE 3
> 30  .BEGIN STEP Y
> 40  LIST PRINTER
(listing on the line printer looks like the following
10   .END OF STEP X

(3 Line Spaces)
```

```
30    .BEGIN STEP Y
```

4.35 SPACESOFF/SPACESON

OPERATION NAME: Control Numeric Print (with/without leading spaces)

MNEMONIC: SPACESOFF/SPACESON

DESCRIPTION: Allows the programmer to print numbers right justified with leading spaces(SPACESON). The default condition is no leading spaces until a SPACESON is executed. SPACESOFF disables leading spaces print.

Note: Hex numbers occupy 5 digits
Octal numbers occupy 7 digits
Decimal numbers occupy 6 digits

EXAMPLE(S):

```
> 10  LET A:=!FDF,B:=%7657,C:=4839
> 20  PRINT !A;%B;C          .LEFT JUSTIFIED
> 30  SPACESON
> 40  PRINT !A;%B;C          .RIGHT JUSTIFIED
> 50  SPACESOFF              .RETURN TO LEFT JUSTIFIED
> 60  RUN
!FDF%76574839
!FDF  %7657  4839
```

Note: If ZEROESON and SPACESON are both enabled then ZEROESON is dominant

4.36 STARTCLOCK

OPERATION NAME: Start System Clock

MNEMONIC: STARTCLOCK [interval in milliseconds]

DESCRIPTION: Initiates operation of the system clock and causes a counter increment every interval as specified in the optional parameter (default is 1 millisecond). The resolution of the clock is +-95% of the interval specified.

EXAMPLE(S):

```
          .
          .
)100 STARTCLOCK      .START 1 MILLISECOND TIMER
          .
          .
          .
) 100 STARTCLOCK 1   .START 1 MILLISECOND TIMER
```

4.37 SUPPRESS

OPERATION NAME: Suppress Errors

MNEMONIC: SUPPRESS

DESCRIPTION: Resets the ENABLE statement override flag thus returning to conditions set by the error printing commands. See ENABLE statement.

4.38 WRITEFILE

OPERATION NAME: Write File

MNEMONIC: WRITEFILE buffer element, length

DESCRIPTION: Writes data starting at the element of the specified buffer into the file "filename"* for length words (or characters if using a string buffer)**. Only DATA and SPLII files may be written into by this statement. (See the Diagnostic/Utility System ERS for further information)

```
EXAMPLE(S):      > 10  DB &AA,6,"HOLD1 "
                 ----
                 > 15  DB BB,200
                 ----
                 > 20  FILENAME &AA(0)
                 ----
                 > 30  WRITEFILE BB(100),20
                 ----            (Writes data starting at BB(100)
                                  into the file HOLD1 for 20 words)
```

* A valid FILENAME statement must be executed prior to executing this statement.

**If the buffer being written is a string buffer the element is rounded down to the nearest even element to maintain even word boundaries. If "rounding" is needed the length parameter is incremented.

```
  Example:  > 100 WRITEFILE &AA(3),HOLD1,5
            -----
```

This statement would write 6 bytes into HOLD1 starting at &AA(2).

4.39 ZEROESOFF/ZEROESON

OPERATION NAME: Control Numeric Print (with/without leading zeroes)

MNEMONIC: ZEROESOFF/ZEROESON

DESCRIPTION: Allows the programmer to print numbers right justified with leading zeroes (ZEROESON). The default condition is no leading zeroes until a ZEROESON is executed. ZEROESOFF disables leading zeroes print.

Note: Hex numbers occupy 5 digits
Octal numbers occupy 7 digits
Decimal numbers occupy 6 digits

```
EXAMPLE(S):      > 10  LET A:=!FDF,B:=%7657,C:=4839
                 ----
                 > 20  PRINT  !A;%B;C     .LEFT JUSTIFIED
                 ----
                 > 30  ZEROESON
                 ----
                 > 40  PRINT !A;%B;C      .RIGHT JUSTIFIED
                 ----
```

```
> 50  ZEROESOFF          .RETURN TO LEFT JUSTIFIED
----
> 60  RUN
----
!FDF%76574839
!0FDF%007657004839
```

Note: If ZEROESON and SPACESON are both enabled then ZEROESON is dominant.

5.0 SPECIAL CHARACTERS

The AID Special Characters are listed, in detail, in this section. The format for each Special Character explanation is:

OPERATION NAME: General phrase of what the Character does.

SYMBOL: The Special Character.

DESCRIPTION: A detailed explanation of the Special Character's function.

EXAMPLE(S): One or more examples using the Special Character.

5.1 . (Period)

OPERATION NAME: Comment Identifier

SYMBOL: (Period)

DESCRIPTION: See the description under Comment in the Statement Section.

5.2 CONTROL H

OPERATION NAME: Backspace (one character)

SYMBOL: CONTROL H (Bs) or BACKSPACE

DESCRIPTION: Allows the operator to backspace to the last Console character by pressing the Control and H keys simultaneously on the Console. The cursor is relocated to the last character input and that character is deleted.

EXAMPLE(S): CRT Example

```
> 10  LES
----   -
   (S is incorrect, Operator presses CONTROL H)
> 10  LE
```

```
                ----    -
```

5.3 CONTROL X

```
OPERATION NAME:  Delete Existing Line Input

SYMBOL:          CONTROL X(CN) or DELETE ENTRY

DESCRIPTION:     Allows the operator to delete the existing input
                 character string by pressing Control and X
                 simultaneously on the Console.  Three
                 exclamation marks (!!!) and a return-line feed
                 are printed* and the operator may input a new
                 string of characters.

EXAMPLE(S):      ) 10  LET Xc !!!  (No input occurs)
                 ----          ---

                 -

                      -or-

                 ?6,7Xc!!! (Deletes all inputs)
                 -     ---

                 -
```

* Note- !!! may not be displayed on some Console types.

5.4 () Parentheses

```
OPERATION NAME:  Enclose

SYMBOL:          () Parentheses

DESCRIPTION:     Used to:

                 --Enclose a buffer element
                 --Enclose a special optional parameter

EXAMPLE(S):

) 10  LET AA(2):=2       .DEFINES ELEMENT 2 OF AA
----
```

```
> 20  LET &BB(2):="H"    .DEFINES BYTE 2 OF &BB
----
> 30  PRINT  "(2)"       .PARENTHESES ARE ASCII CHARACTERS ONLY
----
> 40  RANDOM(X) A        .ENCLOSES OPTIONAL ARGUMENT
 ----
```

5.5 " " (Quotation Marks)

OPERATION NAME: Enclose a Character String

SYMBOL: " " (Quotation Marks)

DESCRIPTION: Encloses a string of characters for assignment or printing.

EXAMPLE(S):

```
> 10  LET &AA(1):="4"   (SET THE RIGHT BYTE
----                     OF WORD 1 OF &AA TO AN ASCII
                         CHARACTER 4)
> 20  LET &CC(10,14):="HELLO"
----                    (STARTING AT CHARACTER 10
                         OF &CC STORE THE ASCII
                         CHARACTERS HELLO SEQUENTIALLY)
> 30  PRINT "OK"        .PRINTS OK ON THE CONSOLE.
----
```

*Note: Quotation marks inside a string are not allowed.

5.6 ! (Exclamation Mark)

OPERATION NAME: Hexadecimal Notation

SYMBOL: ! (Exclamation Mark)

DESCRIPTION: Denotes the following variable, numeric or buffer element will be referenced or manipulated as a hexadecimal based number.

EXAMPLE(S):

```
> 10  PRINT !G       .PRINT THE VALUE OF G IN HEXADECIMAL.
----
> 20  PRINT "!A"     .DENOTES AN ASCII !A ONLY.
----
> 30  LET A:=!F      .A=HEXADECIMAL F
```

5.7 % (Per Cent Sign)

OPERATION NAME: Octal Notation

SYMBOL: % (Per Cent Sign)

DESCRIPTION: If the symbol (%) is not contained in a character string, it denotes the variable, numeric or buffer element following it is represented or manipulated as an octal based number.

EXAMPLE(S):

```
> 10  PRINT %G        .PRINT THE VALUE OF G IN OCTAL
> 20  PRINT "%A"      .DENOTES AN ASCII CHARACTER %A 0
> 30  LET A:=%37      .A=OCTAL 37
```

5.8 Print Spacing

OPERATION NAME: Print Spacing

SYMBOL: 0 through 79

DESCRIPTION: Provides print spacing when concatenating strings in print statements.

EXAMPLE(S):

```
> 10  PRINT 8; "EIGHT"  .PRINTS 8 SPACES AND THEN "EIGHT"
> 20  PRINT "BIG";15;"GAP"
                        .PRINTS BIG, 15 SPACES AND THEN
                        .GAP
```

5.9 > (Greater Than Sign)

OPERATION NAME: Prompt Character

SYMBOL: > (Greater Than Sign)

DESCRIPTION: When AID or an executing program expects a Console input, the prompt character (>) will be printed in t first line space (See the operators section for a description of the "greater than" function).

EXAMPLE(S):

```
> 100  RUN
-----

(ATTENTION)
Break in Statement 50
---------------------
>  (AID IS NOW AWAITING OPERATOR INPUT)
-
```

5.10 & (Ampersand)

OPERATION NAME: String Buffer Desigtion

SYMBOL: & (Ampersand)

DESCRIPTION: Denotes a string buffer. This Special Character is not allowed anywhere else (except inside a character string).

EXAMPLE(S):

```
> 10  DB &AA,10 .DEFINES &AA AS A 10 CHARACTER STRING BUFFER
----
> 20  INPUT &AA(2,4)     .ACCEPTS 3 ASCII CHARACTERS
----
> 30  LET  &A:="HI"   .NOT ALLOWED. VARIABLES CANNOT BE USED
----
> 40  LET &AA:="HI"     (NOT ALLOWED. STRING LENGTH
----                       MUST EQUAL ELEMENT COUNT)
> 45  LET &AA(0,1):="HI"     (ALLOWED.  ELEMENT COUNT
                                EQUALS STRING LENGTH)
> 50  PRINT "&";A     .SPECIFIES AN ASCII & WILL BE PRINTED
----
```

5.11 ; (semi-colon)

OPERATION NAME: Suppress Return-Line Feed

SYMBOL: ; (semi-colon)

DESCRIPTION: If the symbol (;) is contained in a concatenated print string, it denotes no return-line feed is desired after the print operation. A comma is used to force a return-line feed (see comma Special Character).

EXAMPLE(S):

```
> 5  LET A:=5
> 10  PRINT A;            .PRINTS 5 ON THE CONSOLE.
                           THE NEXT PRINT WILL CONTINU
                           FOLLOWING THE 5.
> 20  PRINT A;" DAYS"     .PRINT 5 DAYS ON THE CONSOLE
                           WITH A RETURN-LINE FEED
> 30  PRINT "CALL " ;A    .PRINT CALL 5 ON THE CONSOLE
                           WITH A RETURN-LINE FEED
> 40  PRINT ";"           .PRINTS A SEMI-COLON ON THE
                           CONSOLE
> 50  PRINT A;5;A;4;A,A;5;A
> 60  RUN
  55  DAYS (statement 10 and 20)
  CALL 5   (statement 30)
  ;        (statement 40)
  5     5     5     (statement 50)
  5     5
```

5.12 Control Y Attention

OPERATION NAME: Suspend Execution

SYMBOL: Control Y(Em) or ATTENTION

DESCRIPTION: During execution of a program or command, the operator may interrupt and suspend execution by pressing control and Y simultaneously(or ATTENTION). The prompt character (>) is printed to indicate AID is awaiting operator input.

EXAMPLE(S):

```
.
.
.
> 100  RUN
```

```
_____
(The AID program is now executing.)

CTRL Y                (Operator presses Control and Y)

Break in Statement 20
---------------------
)
-
```

5.13 ? or ??

OPERATION NAME: Input Expected

SYMBOL: ? or ??

DESCRIPTION: A question mark (?) indicates the executing program expects an operator input. A double question mark (??) indicates the operator did not input sufficient information (i.e. more input is expected).

EXAMPLE(S):

```
) 10  PRINT "INPUT"
----
) 20  INPUT A,B,C
----
) 30  PRINT A;2;B;2;C
----
) 40  RUN
----
INPUT
-----
? 3,6
-
?? 8
--
3  6  8
-------
```

5.14 , (Comma)

OPERATION NAME: Separation of Expressions or Force Return-Line Feed

SYMBOL: (Comma)

DESCRIPTION: Comma (,) may be used to separate expressions; to force a return-line feed in concatenated print strin (see semi-colon Special Character for suppressing return-line feed); during command and statement input to separate parameters, and during INPUT execution to delimit individual inputs.

EXAMPLE(S):

```
> 10  LET A:=4, B:=5     .COMMA SEPARATES EXPRESSIONS
----
> 20  PRINT A,B          .FORCE RETURN-LINE FEED
----
> 30  PRINT ","          .DESIGNATES AN ASCII COMMA ONLY
----
> 40  RUN
----
4
 5
 0
```

-or-

```
> 10  RUN 1,2,3          (COMMAS SEPARATE RUN PARAMETERS)
----
```

-or-

```
> 10  INPUT A,B,C
----
> 20  RUN
----
? 1,2,3               (COMMAS SEPARATE INPUT VALUES)
```

5.15 / (slash)

OPERATION NAME: Inclusion

SYMBOL: / (slash)

DESCRIPTION: Allows the operator to enter multiple numbers X/Y meaning X through Y inclusive (also see the Divide Special Character).

EXAMPLE(S):

```
> 100  LIST 10/50        (list statement 10 through 50)
-----
> 100  D20/50            (delete statement 20 through 50)
-----
>    TEST 1/3          (initialize test of Sections 1
-                       through 3)
```

5.16 CTRL-Shift

OPERATION NAME: Interrupt output on IDS

DESCRIPTION: If using the IDS as console, hold down CTRL and Shift keys simultaneously to temporarily interrupt the listing. It will resume when you release the keys. This is an IDS firware feature; it is not provided by the AID interpreter.

6.0 OPERATORS

The Operators available to the programmer are listed in detail in this section. The format for each Operator explanation is:

OPERATION NAME: General phrase of what the Operator does.

MNEMONIC: The form that the Operator would be used in.

DESCRIPTION: A detailed explanation of the Operator's function.

EXAMPLE(S): One or more examples using the Operator.

6.1 :=

OPERATION NAME: Assignment

SYMBOL: :=

DESCRIPTION: Assigns the value of an expression to a variable or buffer (see the LET statement for further examples and explanation).

EXAMPLE(S):

```
) 10  LET A:=2*B+4
) 20  LET &AA(0,5):="HELLO!"   (&AA(0)=H
                                &AA(1)=E,
                                &AA(2)=L,ETC.)
) 30  LET BB(4):=!F      .BB(4)=HEXADECIMAL F
```

6.2 *

OPERATION NAME: Single Word Integer Multiply

SYMBOL: *

DESCRIPTION: Executes an integer multiply on two values. The multiplication product is limited to the range of a single word integer (i.e. = -32,768 to 32,767). Integer overflow during execution will

cause an abort with an error message.

EXAMPLE(S):

```
> 10  LET B:=2
> 20  LET A:=B*20000     .WILL RESULT IN AN OVERFLOW.
> 30  LET A:=B*2         .A = 4
```

6.3 /

OPERATION NAME: Single Word Integer Divide

SYMBOL: /

DESCRIPTION: Executes a single word integer divide on two single integers. To access the remainder from the divide, the MOD Operator may be used. Divide by zero during execution will cause an abort and an error message (see the special inclusion character (/) also).

EXAMPLE(S):

```
> 10  LET A:=4,B:=11
> 20  LET C:=B/A          .C=2  QUOTIENT
> 30  LET D:=B MOD A     .D=3  REMAINDER
```

6.4 +

OPERATION NAME: Single Word Integer Addition

SYMBOL: +

DESCRIPTION: Adds two single word integers and provides a single word result. Overflow (Sum)32767 or Sum(-32768) during execution will result in an error message and will abort the program.

EXAMPLE(S):

```
> 10  LET A:=10, B:=30
> 20  LET C:=A + B       .C = 40
```

6.5 -

OPERATION NAME: Single word integer subtraction

SYMBOL: -

DESCRIPTION: Subtracts two single word integers and yields a single word result. Overflow (Difference)32767 or Difference(-32768) during execution will result in an error message and program abort.

EXAMPLE(S):
```
> 10 LET A:=4
----
> 20 LET B:=10
----
> 30 LET C:=A-B     .C=-6
----
```

6.6 NOT

OPERATION NAME: Ones Complement

MNEMONIC: NOT

DESCRIPTION: Executes ones complement arithmetic on a value (all zeroes to ones, all ones to zeroes).

EXAMPLE(S):
```
> 10  LET A:=-1           .A=-1 OR TRUE*
----
> 20  LET B:=NOT A        .B=0 OR FALSE*
----
```

* Any non-zero number is true and zero is false.

6.7 =

OPERATION NAME: Equal to

SYMBOL: =

DESCRIPTION: Provides a relational test between two values. No assignment is made.

EXAMPLE(S):
```
> 10  IF A = B THEN 20  (GO TO 20 IF A=B)
----
```

```
                > 20  LET A:=B=C (A IS SET TO -1  IF B IS EQUAL TO C
                ----               ELSE A IS SET TO 0)
```

6.8 <>

OPERATION NAME: Not Equal to

SYMBOL: <>

DESCRIPTION: Provides an equality test between two values.

EXAMPLE(S):

```
> 10  IF A <> B THEN 20 .GO TO 20 IF A DOESN'T EQUAL B.
----
> 15                    .A AND B ARE UNALTERED.
----
> 20  LET C:=A<>B       .C IS SET TO -1 IF A<>B OR 0 IF
----                      A=B.
```

6.9 < OR > OR <= OR >=

OPERATION NAME: Greater or Less Than

MNEMONIC: < or > or <= or >=

DESCRIPTION: Provides a relational test between two values. No assignment is made.

EXAMPLE(S):

```
> 10  IF A>B THEN 20    .IF A IS GREATER THAN BUT NOT
----                     EQUAL TO B
> 15                    .THEN 20.
----
> 20  IF A<=B THEN 40   .IF A IS LESS THAN OR EQUAL TO
----                     B THEN 40

> 30  LET A:=B<C        .A=-1 IF B IS LESS
----                     THAN C ELSE A =0
```

6.10 AND

OPERATION NAME: Logical And

MNEMONIC: AND

DESCRIPTION: Provides a Logical AND of two values.

EXAMPLE(S):

```
> 10  LET A:=!C7
> 15  LET B:=!B5
> 20  LET C:=A AND B     .C=!85
> 30  IF A AND B THEN 20
                    (A AND B ARE ANDED AS !85 THEN
                     TESTED FOR TRUTH (NON-ZERO))
```

6.11 OR

OPERATION NAME: Logical OR

MNEMONIC: OR

DESCRIPTION: Provides a Logical OR of two values.

EXAMPLE(S):

```
> 10  LET A:=!C7
> 15  LET B:=!B5
> 20  LET C:=A OR B      .C=!F7
> 30  IF A OR B THEN 20 .A AND B ARE OR-ED AS !F7 TH
                        .TESTED FOR TRUTH (NON-ZERO)
```

6.12 XOR

OPERATION NAME: Exclusive Or

MNEMONIC: XOR

DESCRIPTION: Provides a Logical Exclusive OR of two values.

EXAMPLE(S):

```
> 10  LET A:=!C7
> 20  LET B:=!B5
> 30  LET C:=A XOR B     .C=!72
> 40  IF A XOR B THEN 20.A AND B ARE XOR-ED AS !72
                         .THEN TESTED FOR TRUTH (non-zero)
```

6.13 MOD

OPERATION NAME: Modulo Operation

MNEMONIC: MOD

DESCRIPTION: Provides a means of determining the remainder of a division process.

EXAMPLE(S):

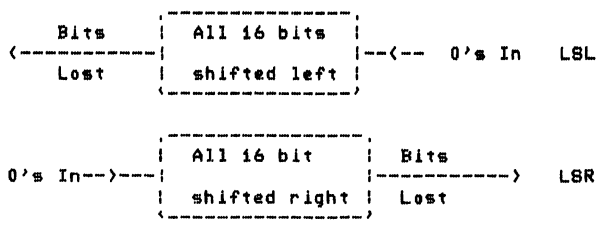
```
> 10  LET A:=10
> 20  LET B:=A MOD 3     .B=1
```

6.14 LSL or LSR

OPERATION NAME: Logical Shift

MNEMONIC: LSL x or LSR x

DESCRIPTION: Logically shifts a value x places where x may be any value. A logical shift corresponds to a logical divide(LSR) or a logical multiply(LSL).



EXAMPLE(S):

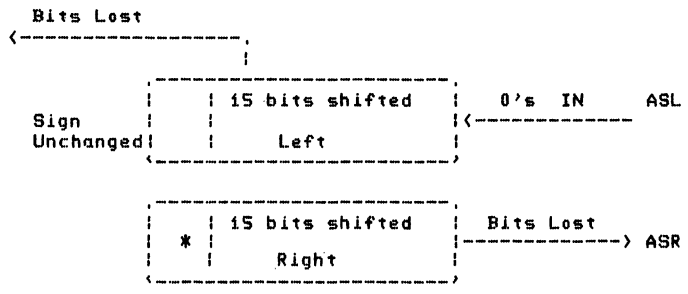
```
> 10  LET A:=A LSR 2     .Shift A logically 2 places right
----
> 20  LET B:=C LSL 1     .Shift C logically 1 place left.
----
> 30  LET C:=5 LSL A     .Shift 5 logically (A) places left
----
```

6.15 ASL or ASR

OPERATION NAME: Arithmetic Shift

MNEMONIC: ASL x or ASR x

DESCRIPTION: Arithmetically shifts an integer value x places where x may be any value. An arithmetic shift corresponds to an integer divide(ASR) or an integer multiply(ASL).



* Copy Sign bit x times.

EXAMPLE(S):

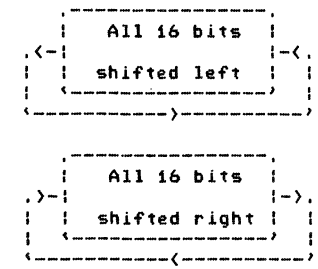
```
> 10  LET A:=A ASL 2     .Shift A arithmetically 2 places
----                       left.
> 20  LET B:=C ASR 1     .Shift C arithmetically 1 place
----                       right.
> 30  LET C:=5 ASL A     .Shift 5 arithmetically (A)
----                       places left.
```

6.16 CSL or CSR

OPERATION NAME: Circular Shift

MNEMONIC: CSL x or CSR x

DESCRIPTION: Executes a Circular Shift on an integer value x places where x may be any value.



EXAMPLE(S):

```
> 10  LET A:=A CSL 8     .Circular Shift A 8 places left.
----
> 20  LET B:=C CSR 1     .Circular shift C 1 place right.
----
> 30  LET C:=5 CSR A     .Circular shift 5 (A) places right
----
```

6.17 Special Relational Operators

OPERATION NAME: Special Relational Operators

MNEMONIC: NE (Not Equal), EQ (Equal To), LT (Less Than), GT (Greater Than), LE (Less Than or Equal To), GE (Greater Than or Equal To)

DESCRIPTION: These special operators may be used only in the IF-THEN and IFN-THEN statements. The operators NE, EQ, LT, GT, LE and GE may be used to logically AND up to three expressions which determine whether a branch should occur to the "THEN" statement. Evaluation of the "IF" expressions occurs left to right.

EXAMPLE(S):

```
> 10  IF 5 LT A LT 10 THEN 150
----                      (This statement is evaluated as:
                           IF (5<A) AND (A<10) THEN GO TO
                           STATEMENT 150)
> 50  IF A:=R MOD 200 LT 0 THEN 60
----                      (This statement says:
                           IF (A:=R MOD 200)<0
                           THEN 60).
                           Note that A is not stored with
                           a relational result (see next
                           example).

> 70  IF A:=R MOD 200<0 THEN 50
----                      (This statement would store A with
                           a True or False value R MOD 200<0)
```

FOR MORE EXAMPLES SEE THE "IF" STATEMENT.

7.0 RESERVED VARIABLES

The Reserved Variables available to the operator are listed in detail in this section. The format for each Reserved Variable explanation is:

OPERATION NAME: General phrase of what the Reserved Variable means.

MNEMONIC: The form that the Reserved Variable would be called in.

DESCRIPTION: A detailed explanation of the Reserved Variable's function.

INITIALIZED TO: Displays the value the Reserved Variable is set to at the start of program execution (i.e. at RUN time).

EXAMPLE(S): One or more examples using the Reserved Variable.

7.1 BADINTP

OPERATION NAME: Bad Interrupt

MNEMONIC: BADINTP

DESCRIPTION: Should an interrupt occur from an unexpected device or multiple interrupts occur from an expected device the erroneous channel/device is stored in BADINTP%. Some diagnostics will use this information to test interrupt operation. If BADINTP is non-zero when an RSIO statement is executed, AID will report an error.

INITIALIZED TO: Zero

EXAMPLE(S):

```
> 1000  RSIO AA          .START CHANNEL PROGRAM
> 1010  IF BADINTP <>0 THEN 2000
> 1020  .OK - TRY NEXT STEP
```

* Bits 8-12= Channel and Bits 13-15= Device

7.2 CHANNEL

OPERATION NAME: Set I/O Channel Number

MNEMONIC: CHANNEL

DESCRIPTION: Specifies the channel number of the I/O device to be used in subsequent I/O or channel progra operations.

INITIALIZED TO: Zero

EXAMPLE(S):

```
> 10  LET CHANNEL:=2,DEVICE:=0 (Following I/O operations will
                                execute on Channel 2, Device 0)
```

7.3 CONCHAN

OPERATION NAME: Console Channel Number

MNEMONIC: CONCHAN

DESCRIPTION: This Reserved Variable is initialized to the channel device number of the AID Console where bits 9-12= channel and bit 13-15=device.

INITIALIZED TO: Console Channel-Device number

EXAMPLE(S):

```
> 10  PRINT "AID CONSOLE CHANNEL=";%CONCHAN
> 20  RUN

AID CONSOLE CHANNEL=%10
```

7.4 DEVICE

OPERATION NAME: Set I/O Device Number

MNEMONIC: DEVICE

DESCRIPTION: Specifies the device number of the I/O device to be used in subsequent I/O or channel program operations.

INITIALIZED TO: Zero

EXAMPLE(S):

```
) 10  LET CHANNEL:=2,DEVICE:=4  (Following I/O operations will
----                             execute on channel 2,device 4)
```

7.5 FILEINFO

OPERATION NAME: File Information

MNEMONIC: FILEINFO

DESCRIPTION: After a FILENAME statement has executed FILEINFO contains the following information about the file:

Bit 0 =1 if file protected otherwise 0
Bit 8/11 =Class of the file
Bit 12/15 =Type of the file

(See Diagnostic/Utility System ERS)

INITIALIZED TO: Zero

EXAMPLE(S): Assume the file XYZ is protected, class 1(diagnostic), type 1(SPLII) and length is 256 words: 10 DB &AA,10,"XYZ "

```
--
20 FILENAME &AA(0)
--
30 LET A:=FILEINFO AND %100000 LSR 15
--
40 LET B:=FILEINFO AND %360 LSR 4
--
50 LET C:=FILEINFO AND %17
```

```
--
60 PRINT &AA(0,2);" file ","PROTECT BIT=";A;2;
--
70 PRINT "Class=";B;2;"Type=";C;2;"Length=";FILELEN
--
80 RUN
--
XYZ file
PROTECT BIT=1  Class=1  Type=1  Length=256
```

7.6 FILELEN

OPERATION NAME: File Length

MNEMONIC: FILELEN

DESCRIPTION: After a FILENAME statement has executed FILELEN contains the length of the specified file rounded up to the nearest 128 word sector boundary.

INITIALIZED TO: Zero

EXAMPLE(S): See FILEINFO Reserved Variable example.

7.7 GOPARAM1/GOPARAM2/GOPARAM3

OPERATION NAME: Go Parameters

MNEMONIC: GOPARAM1/GOPARAM2/GOPARAM3

DESCRIPTION: Allows the executing program to access up to three parameters that may have been passed during the last GO Command. The default value of unpassed parameters is 0.

INITIALIZED TO: Zero

EXAMPLE(S):

```
) 10  IF GOPARAM2=2 THEN 50 (IF THE SECOND PARAMETER
----                          IN THE GO COMMAND WAS 2
                                THEN GO TO 50)
```

```
        -or-

> GO 4,,6
_
> GO 4,,6      (GOPARAM1=4 GOPARAM2=0, GOPARAM3=6)
_
```

7.8 INDEX

OPERATION NAME: Buffer Compare Indicator

MNEMONIC: INDEX

DESCRIPTION: After a compare buffer (CB) statement has executed, INDEX will contain -1 if the buffers compared or it will contain the element of the first buffer in the CB statement that didn't compare.

INITIALIZED TO: Zero

EXAMPLE(S):

```
> 10  CB AA(10), BB(10),20  .ASSUME AA(11)<>BB(11)
> 20  IF INDEX=-1 THEN 80  .INDEX=11
> 30  PRINT "GOOD= ";AA(INDEX);"BAD=";BB(INDEX)
> 35  .CHECK THE REST OF THE BUFFER
> 40  FOR INDEX:= INDEX + 1 UNTIL 29
> 50  IF AA(INDEX)<>BB(INDEX) THEN 30
> 70  NEXT 40
> 80  .NEXT STATEMENT
```

7.9 INPUTLEN

OPERATION NAME: Last Input character Length

MNEMONIC: INPUTLEN

DESCRIPTION: This Reserved Variable contains the character length of the last input of the most recently executed INPUT statement.

INITIALIZED TO: Zero

EXAMPLE(S):

```
> 10  INPUT A
----
> 20  PRINT INPUTLEN
----
> 30  RUN
----
? 437
3 (INPUTLEN=3)
-
     -or-

> 10  INPUT A,B
----
> 20  PRINT INPUTLEN
----
> 30  RUN
----
? 437,26
2 (LAST INPUT WAS 2 CHARACTER,I.E.-ASCII 26)
-
     -or-

> 10  INPUT &AA(4,10)
----
> 20  PRINT INPUTLEN
----
> 30  RUN
----
? HELLO
-
5
-
- (INPUTLEN=5 EVEN THOUGH 7 CHARACTERS WERE EXPECTED
```

7.10 MAXMEMORY

OPERATION NAME: Maximum Buffer Area

MNEMONIC: MAXMEMORY

DESCRIPTION: Dynamically indicates the amount of unused buffer space available to the executing program.

INITIALIZED TO: Memory space available prior to RUN time

EXAMPLE(S):

```
> 20  IF MAXMEMORY < 4000 THEN 50
> 30  DB AA, 4000
> 40  GOTO 60
> 50  DB AA, 2000
```

(IF THE DB AT 30 WAS EXECUTED THEN MAXMEMORY WOULD THEN EQUAL MAXMEMORY - 4000)

7.11 NEWTEST

OPERATION NAME: Test Command Indicator

MNEMONIC: NEWTEST

DESCRIPTION: This Reserved Variable may be used to determine if a test section sequence has been specified externally. NEWTEST is set to false when a TEST command is entered with no parameters and stays false until a TEST Command with parameters is entered.

INITIALIZED TO: Not altered at RUN time

EXAMPLE(S): The XYZ Program has ten sections that are executed as a standard test and section 11 which is optional. A typical entry sequence would be:

```
> 10  IF NEWTEST THEN 30
> 20  LET SECTIONS 1:=!FFDF .CLEAR SECTION 11 INDICA
```

```
                > 30   .continue
                ----
```

(See Reserved Variables SECTIONS 1/3 and Command TEST for further explanations)

7.12 NOINPUT

OPERATION NAME: Non-Error Print Indicator

MNEMONIC: NOINPUT

DESCRIPTION: NOINPUT is true if non-error print is suppressed (i.e. the SNPR Command was executed). This allows the executing program to determine if a PRINT, INPUT statement sequence should be executed (i.e., if non-error print is suppressed then no INPUT statement will be executed therefore rendering any test of the input data invalid). Setting NOINPUT to false will override the SNPR command but should be used with caution.

INITIALIZED TO: Zero

EXAMPLE(S):

```
                > 10   IF NOINPUT THEN 50
                ----
                > 20   PRINT "DO YOU WANT TO CONTINUE?"
                ----
                > 30   INPUT & AA(0)
                ----
                > 40   IF &AA(0) = "Y" THEN 400
                ----
                > 50   END
                ----
                > 60   .NEXT STATEMENT
                ----
                       .
                       .
```

If an SNPR command has been previously entered, then the program will skip past the INPUT sequence of statements 20 to 40.

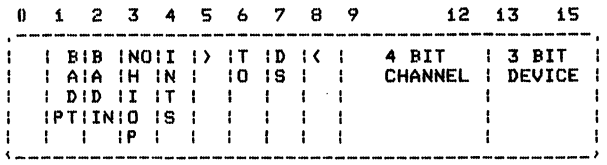
7.13 NORESPONS

OPERATION NAME: No Response to I/O Flag

MNEMONIC: NORESPONS

DESCRIPTION: If an I/O instruction or channel program execution returns an error condition and this Reserved Variable is still equal to 0 then AID will handle the error. However, if the user program has changed the value of NORESPONS to non-zero then AID will set NORESPONS (see table below) and not report an error. By setting NORESPONS to a value other than 0 the user program can handle the no response error.

NORESPONS Reserved Variable Format



If NORESPONS<>0 when a channel error occurs then:

Bit	Meaning (if set)
0	reserved
1	DRT0 not pointing to channel program
2	Illegal interrupt from device in Bits 9/15
3	HIOP did not halt channel program
4	too many device interrupts
5	CCG returned after I/O command
6	channel program time out (approx. 5 secs)
7	channel program did not start
8	CCL returned after I/O command
9-15	channel-device number when error occurred (bits 9-12=channel number, bit 13-15=device)

INITIALIZED TO: Zero

EXAMPLE(S):

```
> 10  LET NORESPONS:=2
> 20  LET CHANNEL:=2, DEVICE:=7
> 30  INIT
> 40  IF NORESPONS=2 THEN 60 .CHECK IF INIT WAS OK?
> 50  GOSUB 1000          .NO! PROCESS NORESPONS ERROR
> 60  .ADDITIONAL CODE
```

7.14 OFFSET

OPERATION NAME: Vary Return Point

MNEMONIC: OFFSET

DESCRIPTION: OFFSET may be used to vary the statement number returned to when executing a RETURN statement. OFFSET is set to zero when starting execution and after a RETURN statement execution. OFFSET, if used, may be set to any integer value indicating the number of statements after (if positive) or before (if negative) the normal return statement to return to.

INITIALIZED TO: Zero

EXAMPLE(S):

```
> 10  PRINT "Input yes or no"
> 20  INPUT &AA(0)
> 30  GOSUB 500           .GO CHECK FOR YES OR NO
> 40  GOTO 100            .GO TO "YES" ROUTINE
> 50  .START NO ROUTINE

>500  IF &AA(0)="Y" THEN 540  .RETURN NORMALLY
>510  LET OFFSET:=1       .FORCE RETURN TO 50
>520  IF &AA(0)="N" THEN 540
>530  LET OFFSET:=-3      .FORCE RETURN TO 10
>540  RETURN
```

```
n+4+(2*s)       (If c)7)
                              .
                              .
                              .

XAMPLE(S):      ) 10  LET CHANNEL:=5            .Define Disc
                ----
                ) 20  DB AA,3                   .Create Buffer
                ----
                ) 30  LET AA(0):=!303           .Disc Status Command
                ----
                ) 40                            .To Unit 3
                ----
                ) 50  GOSUB 200                 .Get Disc Status
                ----
                ) 60  PRINT "DISC STATUS = ";AA(1);AA(2)
                ----
                ) 65                            .Output Result
                ----
                ) 70  END
                ----
                )200  BSIO BB                   .Build Channel Program to
                ----
                )210                            .Get Status from the Disc
                ----
                )220  WR 8,AA(0),2              .Output Status Command
                ----
                )230  RR 8,AA(1),4              .Input Two Status Words
                ----
                )240  IN H                      .End of Channel Program
                ----
                )250  RSIO                      .End of Definition of
                ----
                )260                            .Channel Program -- Start
                ----
                )270                            .Execution
                ----
                )280  RETURN
                ----
```

8.3 COPY

OPERATION NAME: Copy Channel Program

MNEMONIC: COPY XX [*N]

DESCRIPTION: Duplicates the master channel program in XX into all copies of XX. If the optional *N is added then only the Nth copy of XX will be duplicated. Since the RSIO instruction automatically duplicates copies COPY would be needed if modification to a channel program is needed before execution (See example). Note: Copy number 0 is the first channel program copy.

EXAMPLE(S):

```
> 10  LET CHANNEL:=2,DEVICE:=4
----
> 20  BSIO AA,3  .CREATE 3 COPIES OF CHANNEL PROGRAI
----
> 30  IN H,1,5
----
> 40  ESIO
----
> 50  LOCATE 30,A  .GET IN H POINTER TO COPY 0
----
> 60  LET AA(A):=6 .CHANGE HALT CODE TO 6 IN COPY 0
----
> 70  RSIO AA,0     .RUN FIRST COPY
----
> 80  COPY AA*0     .DUPLICATE FIRST COPY ONLY
----
> 90  GOTO 60       .LOOP ON CHANNEL PROGRAM
----
```

8.4 CPVA

OPERATION NAME: Set User CPVA

MNEMONIC: CPVA XX(N)

DESCRIPTION: Sets a pointer to the data buffer XX(N) as the CPVA during subsequent channel program executions. The data buffer XX must be declared at least 7 words long. If this statement is not used the CPVA pointer defaults to absolute memory and is not accessible by the user.

XAMPLE(S):

```
> 10  DB AA,7,0
> 20  LET CHANNEL:=3,DEVICE:=4
> 30  CPVA AA(0)  .SET CPVA POINTER TO AA(0)
```

3.5 ESIO

PERATION NAME: End Channel Program Definition

NEMONIC: ESIO

ESCRIPTION: This statement is used to mark the end of the definition of a Channel program.

EXAMPLE(S): See BSIO

3.6 HIOP

PERATION NAME: Halt Channel Program

NEMONIC: HIOP

DESCRIPTION: This statement, when executed, will terminate the channel program executing on the currently selected device.

EXAMPLE(S):

```
> 10  LET CHANNEL:=5
> 20  PROC   .SET PROCEED MODE
> 30  BSIO AA
> 40  JUMP 50
> 50  JUMP 40
> 60  RSIO          .Start Program Which Never Ends
> 70  HIOP          .Stop Channel Program
```

8.7 INIT

OPERATION NAME: Initialize I/O Channel

MNEMONIC: INIT

DESCRIPTION: This statement will initialize the currently selected channel. The following actions take place.

(1) Operations in progress on the channel are terminated.
(2) The channel interrupt enable bit is cleared.
(3) Channel registers are set to initial values.
(4) HP-IB is set to idle state.
(5) The fourth word of each DRT for this channel is cleared.
(6) The mask bit for this channel is cleared (memory location 7).

8.8 IOCL

OPERATION NAME: I/O Clear

MNEMONIC: IOCL

DESCRIPTION: This statement will clear all I/O channels. The following actions take place:

(1) Operations in progress on each channel are terminated.
(2) All channel interrupt enable bits are cleared.
(3) Channel registers are set to initial values.
(4) All HP-IBs are set to the idle state.
(5) The fourth word of each DRT is cleared.
(6) All mask bits are cleared (memory location 7).

8.9 ION/IOFF

OPERATION NAME: Enable/Disable External Interrupts

MNEMONIC: ION/IOFF

DESCRIPTION: IOFF will disable the external interrupt system by clearing the interrupt bit in the status register. Use ION to enable external interrupts.

8.10 LOCATE

OPERATION NAME: Locate a Channel Program Element

MNEMONIC: LOCATE [(copy),] label [(offset)],variable

DESCRIPTION: Finds the element within a channel program buffer correlating to the second word of a channel program instruction (specified in label) and stores that word in the parameter variable. If the optional copy is used (where 0<=copy<=31 and default is 0) then that copy of the channel program is used. If the optional offset is added (default is 0 offset from the second word of the channel instruction) then that many words are added (or subtracted) to the result stored in the parameter variable.

Note: Copy number 0 is the first channel program copy.

EXAMPLE(S):

```
> 10  LET CHANNEL:=2
> 20  BSIO AA
> 30  IN H,1,3
> 40  ESIO
> 50  LOCATE 30,A  .GET POINTER TO 2ND WORD OF IN H
> 60  LET AA(A):=5 .CHANGE HALT CODE TO 5.
```

8.11 PROC

OPERATION NAME: Proceed

MNEMONIC: PROC [N]

DESCRIPTION: This statement is used to enable(or disable when the N is added) the proceed mode. AID normally waits for each Channel program to interrupt before continuing to the statement following the RSIO. This normal mode of having I/O with wait maybe changed to the proceed mode(i.e. I/O

without wait) by using this statement.

EXAMPLE(S): (Assume AA and BB are predefined Channel program buffers)

```
> 990   PROC              .PERFORM I/O WITHOUT WAIT
> 1010  LET CHANNEL:=2
> 1020  RSIO AA           .START CHANNEL PROGRAM AA
> 1030  LET CHANNEL:=3
> 1040  RSIO BB           .START CHANNEL PROGRAM BB
> 1050  PROC N            .WAIT HERE FOR I/O TO FINISH
```

8.12 RDRT

OPERATION NAME: Read DRT Word

MNEMONIC:
```
RDRT Z,X
RDRT Z,XX(N)
```

DESCRIPTION: The DRT(device reference table) entry is selected by the currently selected channel device. Z is the DRT word to read (0 <= Z <= 3). The word read is stored in X or XX(N).

EXAMPLE(S):
```
> 10  LET CHANNEL:=2
> 20  RDRT 3,A          .PLACE DRT WORD 3 IN A
```

8.13 RIOC

OPERATION NAME: Read I/O Channel

MNEMONIC:
```
RIOC K, XX(N) [,C]
RIOC K, X [,C]
```

DESCRIPTION: This statement will issue a command C (where 0<=C<=!F and the default is 0) to register K (0 <= K <= !F) on the currently selected channel. The result is placed in X or XX(N).

EXAMPLE(S):
```
) 10 LET CHANNEL:=2,DEVICE:=5
) 20 RIOC 3,A  .Read I/O Register 3 into A
) 30 PRINT "REG 3=";!A
) 40 RUN

REG 3=!4014
End of AID user program
```

8.14 RMSK

OPERATION NAME: Read Interrupt Mask

MNEMONIC:
```
RMSK X
RMSK XX(N)
```

DESCRIPTION: This statement will read the mask word (memory location 7), and place it in X or XX(N).

EXAMPLE(S):
```
) 10  RMSK A              .A = MASK WORD
) 20  RUN
```

8.15 ROCL

OPERATION NAME: Channel Roll Call

MNEMONIC:
```
ROCL XX(N)
ROCL X
```

DESCRIPTION: This statement will place an interrupt mask in XX(N) or X. Each bit of XX(N) or X is set to one if the corresponding channel is present.

EXAMPLE(S):
```
> 10  ROCL A
> 20  PRINT "Channels present=";
> 30  FOR Q:=R:=1 UNTIL 15 .See if Channel is presen
> 40  IFN A LSL Q AND !8000 EQ !8000 THEN 70 .Is it?
> 50  PRINT Q;1;    .Yes! Print it's number
> 60  LET R:=R+1
> 70  NEXT 30
> 80  IF R<>1 THEN 100  .Any Channels present?
> 90  PRINT "NONE";      .No! Tell operator
>100  PRINT
>110  RUN
```

8.16 RSIO

OPERATION NAME: Run Channel Program

MNEMONIC: RSIO [XX [,[C][,SN]]]

DESCRIPTION: This statement may be used instead of ESIO to terminate Channel program definition. XX (a buffer) may only be added when outside Channel program definition. See BSIO for more information. This statement differs from ESIO in that it initiates the Channel program execution. C is the copy number (0 <= C <= 31). Default for C is 0. SN, if added, is the

statement number to execute next if an error is detected during execution of the RSIO. Note: Copy number 0 is the first channel program copy.

EXAMPLE(S):

```
> 10  LET CHANNEL:=5          .Define Device
> 20  BSIO AA                .Create First Program
> 30  IN H
> 40  RSIO                   .Run First Program
> 50  BSIO BB                .Create Second Program
> 60  IN H
> 70  ESIO
> 80  RSIO AA                .Run First Program
> 90  RSIO BB                .Run Second Program
>100  RUN
```

8.17 RSW

OPERATION NAME: Read Switch Register

MNEMONIC:
```
RSW X
RSW XX(N)
```

DESCRIPTION: This statement when executed will place the value of the switch register in X or XX(N). Bits 13-15 hold the device number, and bits 9-12 hold the channel number.

EXAMPLE(S):

```
> 10  RSW A
> 20  PRINT "Switch Register=";!A
> 30  RUN
Switch Register=!20

End of AID user program
```

8.18 SMSK

OPERATION NAME: Set Interrupt Mask
MNEMONIC: SMSK X

DESCRIPTION: Sends the mask word X to all channels and a copy is stored in memory location 7.

EXAMPLE(S):
```
> 10  LET A:=!4000
> 20  SMSK A    .ENABLE CHANNEL ONE INTERRUPTS.
```

8.19 UPDATEOFF/UPDATEON

OPERATION NAME: Prevent channel programs from being updated

MNEMONIC: UPDATEOFF/UPDATEON

DESCRIPTION: UPDATEOFF prevents words 2,4 and 5 of read and write portions of channel programs from being updated by the channel program microcode. UPDATEON (the default condition) restores updating. Updating is indicated by the state of bit 5 of word 4 of Read/Write channel instructions.

8.20 WIOC

OPERATION NAME: Write I/O Channel

MNEMONIC:
```
WIOC K, XX(N), [C]
WIOC K, X, [C]
```

DESCRIPTION: This statement will write X or XX(N) into register K (0<=K<=!F) on the currently selected channel. The parameters are the same as those for RIOC.

9.0 AID STATEMENTS (CHANNEL PROGRAM TYPE)

The following Channel Program Type AID Statements must be located between the BSIO and ESIO Statements. The format of each statement explanation is:

OPERATION NAME: General phrase of what the Statement does.

MNEMONIC: The form that the Statement would be called in. X is used to indicate the variables A to Z or a number. XX is used to indicate the buffers AA to ZZ. N is the same as X but is used as an index (XX(n)).

DESCRIPTION: A detailed explanation of the Statement's function.

EXAMPLE(S): One or more examples using the Statement.

9.1 CHP

OPERATION NAME: Command HP-IB

MNEMONIC: CHP V0,[V1, . . VN]

DESCRIPTION: This statement executes the Command HP-IB channel instruction. VN is the Nth HP-IB command (0<=N<=7) and is a reference to a variable or buffer element which contains the command or is the command in numeric form.

EXAMPLE(S):

```
> 10  LET CHANNEL:=5, DEVICE:=1
> 20  BSIO AA
> 30  CHP !3F,!5E,!25,!6F
> 40  .UNLISTEN, TALK 30, IDS-LISTEN, ENABLE DOWNLOA
> 50  RSIO
> 60  RUN
```

NOTE: VN (a 16-bit quantity) is converted to a byte and stored in the CHP portion of the channel program.

9.2 CLEAR

OPERATION NAME: Control Clear

MNEMONIC: CLEAR [X]

DESCRIPTION: This statement executes the Clear channel instruction. Commands the currently selected device to clear itself. If the optional X is added it forms the control byte(where 0<=X<=!FF and the default is 0) in the channel instruction.

EXAMPLE(S):

```
> 10  LET CHANNEL:=5
> 20  BSIO AA
> 30  CLEAR      .CLEAR CHANNEL 5, DEVICE 0
> 40  RSIO
```

9.3 DSJ

OPERATION NAME: Device Specified Jump

MNEMONIC:
```
DSJ S0[*R0][,S1[*R1]...[,SM[*RM]]...]][;XX(N)]
DSJ S0[*R0][,S1[*R1]...[,SM[*RM]]...]][;X]
```

DESCRIPTION: This statement executes the DSJ channel program instruction. A jump occurs as a result of the byte returned from the device. If XX(N) or X is added, then the byte returned (last byte should the DSJ execute more than once) or !FF (if the DSJ never executes) is placed in the right byte of XX(N) or X. The left byte of XX(N) or X will be set to 0. SM is the statement to execute when the returned byte of the DSJ is equal to M. SM must be in the same Channel program. *RM is the total number of jump address copies of SM to build into the DSJ instruction.

EXAMPLE(S):

```
> 5   DB BB,7,0
> 7   CPVA BB(0)          .Define CPVA
> 10  LET CHANNEL:=5      .Define Disc
> 20  BSIO  AA            .Begin Channel Program
```

```
> 30  DSJ 40,60;A          .Stuff return byte into A
> 40  IN H, 0, 7           .Error--Store halt code 7
> 50                       .In CPVA0
> 60  IN H                 .OK--Clear CPVA0
> 70  RSIO                 .Start Execution
> 80  PRINT "DSJ=;A;2;"CPVA0=";BB(0)
                           .Output Results
```

9.4 IDENT

OPERATION NAME: Identify

MNEMONIC:
```
IDENT XX(N)
IDENT X
```

DESCRIPTION: This statement executes the IDENT channel program instruction. The word returned from the device (last word should it execute more than once) or !FFFF (if it never executes) is placed in XX(N) or X.

EXAMPLE(S):
```
> 10  LET CHANNEL:=5          .Define Disc
> 20  DB BB,8                .Create Buffer
> 30  BSIO AA                .Begin Channel Program
> 40  IDENT BB(7)            .Stuff ID into BB(7)
> 50  IN H                   .Stop Execution
> 60  RSIO                   .Start Channel Program
> 70  PRINT "IDENTIFY CODE =";BB(7)
```

9.5 IN

OPERATION NAME: Interrupt Halt or Run

MNEMONIC:
```
IN H [, [X][,C]]
IN R [, [X][,C]]
```

DESCRIPTION: Executes the INTERRUPT channel program instruction. R, if used, will allow the Channel program to continue to run when this instruction is reached. H, if used, will cause the Channel program to halt when this instruction is reached. X is the CPVA offset (0 <= X <= 3). C is the code to store at CPVAX on interrupt(0<=C<=255). Default for both X and C is 0.

EXAMPLE(S):

```
> 4  DB BB,4
> 5  CPVA BB(0)     .DEFINE CPVA
> 6  LET CHANNEL:=5
> 10 BSIO AA            .Define the following Channel Program
> 20 IN R,3,1           .CPVA3 : = 1
> 30 IN R,2,2           .CPVA2 : = 2
> 40 IN R,1,3           .CPVA1 : = 3
> 50 IN H,,4            .Stop Program Set CPVA0 : = 4
> 60 RSIO               .Execute the Above Program
> 70 PRINT "CPVA0=";BB(0);2;"CPVA1=!BB(1)
> 80 PRINT "CPVA2=";BB(2);2;"CPVA3=";BB(3)
```

9.6 JUMP

OPERATION NAME: Direct Jump

MNEMONIC: JUMP SN

DESCRIPTION: This statement executes the JUMP channel program instruction. SN is an AID statement number. The statement number must be within the same Channel program.

EXAMPLE(S):

```
> 10  LET CHANNEL:=5          .Define Disc
> 20  BSIO AA
> 30  DSJ 40,50;A             .Does Disc respond?
> 40  JUMP 30                 .No! Wait some more.
> 50  IN H                    .Yes! Exit Channel program.
> 60  ESIO
> 70  RSIO AA
```

9.7 RB

OPERATION NAME: Read Burst

MNEMONIC: RB MOD, XX(N), BC [,[BL][,[DC=X][,[R][,[TD]]]]

DESCRIPTION: This statement executes the Read Burst channel program instruction. MOD is the device dependent modifier(0<=MOD<=!1F). If MOD)!F then Read Control is used instead of Read. XX(N) defines the initial buffer location where the data is to be stored. BC is the total number of bytes to be read. BL is the burst length (default is 1) 1<=BL<=256. Burst length is the number of bytes to read this time through the RB. DC, if added, will allow separate data buffers to be linked (chained) by using sequential RB statements. X is equal to number of links to follow. R, if added, will cause the data to be stored starting in the right byte of XX(N) (default is the left byte). TD, if added, is the statement number to which channel program execution is transferred upon successful completion of the RB.

EXAMPLE(S):

```
> 10  LET CHANNEL:=7
> 20  BSIO BB              .Begin Channel Program
> 30  RB 0,AA(0),1         .Read One Byte Into
> 40                       .Left Byte of AA(0)
> 50  IN H                 .Done
> 60  RSIO                 .Execute Channel Program
```

-or-

```
> 10  LET CHANNEL:=2
> 20  DB AA,1
> 30  BSIO BB
> 40  RB 31,AA(0),1        .Read self test results
> 50  IN H
> 60  RSIO
```

9.8 RDMAB

OPERATION NAME: READ DMA Burst

MNEMONIC: RDMAB XX(N), BC[,[BL][,R][,TD]]]

DESCRIPTION: This statement executes the Read DMA Burst channel program instruction. The parameters are the same as those for RB except the modifier and DC are deleted. See HP-300 I/O ERS for definition.

9.9 RDMAR

OPERATION NAME: READ DMA Record

MNEMONIC: RDMAR XX(N),BC [,[R][,TD]]

DESCRIPTION: This statement executes the Read DMA Record channel program instruction. The parameters are the same as those for RR except the modifier and DC are deleted. See HP-300 I/O ERS for definition.

9.10 RMW

OPERATION NAME: Read Modify Write

MNEMONIC: RMW K, BN, C
RMW K, BN, S

DESCRIPTION: This statement executes the Read Modify Write channel program instruction. K is the register to be modified (0<=K<=!F). BN is the bit number of register K to modify (0<=BN<=!F). C will clear the bit and S will set it. REGISTER K is read, bit number BN is modified, then register K is written. For some registers BN has special meaning. See HP-300 I/O System ERS for further register definition.

9.11 RR

OPERATION NAME: Read Record

MNEMONIC: RR MOD, XX(N), BC[, [DC=X][, [R][, TD]]]

DESCRIPTION: This statement executes the Read Record channel instruction. MOD is the device dependent modifier (0<=MOD<=!1F). If MOD is greater than !F then Read Control is used instead of Read. XX(N) defines the initial buffer location where the data is to be stored. BC is the number of bytes to be read. If R is added will cause the data to be stored starting in the right byte of XX(N) (default is the left byte). DC(data chain), if added, will allow separate data

buffers to be linked (chained) by using sequential RR statements. X is equal to number of links to follow. TD, if added is the statement number to which channel program execution is transferred upon successful completion of the RR.

EXAMPLE(S):

```
) 100  RR 0,JJ(0),256,DC=2  .READ 4 SECTORS. PLACE THE
-----
) 110  RR 0,BB(0),512,DC=1  . FIRST ONE IN JJ AND THE LAST
-----
) 120  RR 0,FF(128),256     .  ONE AT FF(128)
-----
```

9.12 RREG

OPERATION NAME: Read Register

MNEMONIC: RREG K, XX(N)
RREG K, X

DESCRIPTION: This statement executes the Read Register Channel instruction. K is the Channel Register to be read (0<=K<=!F). XX(N) or X is where the data is placed. If this statement doesn't execute then !FFFF is placed in X or XX(N). Should this statement execute more than once, the last value read will be placed in X or XX(N).

9.13 WAIT

OPERATION NAME: Wait

MNEMONIC: WAIT [S]

DESCRIPTION: This statement executes the WAIT channel program instruction. The channel program is suspended until the device requests service. If S is used then bit 15 of the first word of the wait instruction is set. The special mode is described in the HP-300 I/O ERS.

EXAMPLE(S):

```
> 10  LET CHANNEL:=5
> 20  DB AA,3
> 30  LET AA(0):=!200    .Seek Command
> 40  LET AA(1):=100     .Cylinder 100
> 50  LET AA(2):=!105    .Head 1,Sector 5
> 60  BSIO BB
> 70  WR 8, AA(0), 3     .Issued Seek
> 80  WAIT               .Wait for Completion
> 90  IN H               .Done
>100  RSIO               .Start Channel Program
```

9.14 WB

OPERATION NAME: Write Burst

MNEMONIC: WB MOD, XX(N), BC[,[BL] [,[DC=X][,[R][, [E]]]]]

DESCRIPTION: This statement executes the Write Burst channel program instruction. The parameters are the same as those for RB except the TD is not valid and E is added to flag the end of each burst with the HP-IB END message.

EXAMPLE(S):

```
> 10  LET CHANNEL:=7
> 15  DB AA,6
> 20  BSIO BB             .Begin Channel Program
> 30  WB 0,AA(5),1,,,R    .Write One Byte
> 40                      .From the Right
> 50                      .Byte of AA(5)
> 60  IN H                .Done
> 70  RSIO
```

```
   -or-

> 10  LET CHANNEL:=2
----
> 20  DB AA,1,0          .Control byte is 0
----
> 30  BSIO BB
----
> 40  WB 31,AA(0),1      .Initiate Self test
----
> 50  IN H
----
> 60  RSIO
----
```

9.15 WDMAB

OPERATION NAME: Write DMA Burst

MNEMONIC: WDMAB XX(N), BC [,[BL][,[R][,E]]]

DESCRIPTION: This statement executes the Write DMA Burst channel instruction. The parameters are the same as those for WB except the modifier and DC are deleted. See HP-300 I/O ERS for definition.

9.16 WDMAR

OPERATION NAME: Write DMA Record

MNEMONIC: WDMAR XX(N), BC[,R]

DESCRIPTION: This statement executes the Write DMA Record channel program instruction. The parameters are the same as WR except the modifier and DC are deleted. SEE HP-300 I/O ERS for definition.

9.17 WR

OPERATION NAME: Write Record

MNEMONIC: WR MOD, XX(N), BC[, [DC=N][, R]]

DESCRIPTION: This statement executes the Write Record channel program instruction. The parameters are the same as those for RR except the TD is not valid.

EXAMPLE(S):

```
> 10  WR 0,JJ (0),256,DC=2 .WRITE 4 SECTORS. GET FIRST
----
> 20  WR 0,BB(0),512,DC=1   . FROM JJ, THE NEXT TWO FROM BB
----
> 30  WR 0,FF(128),256      . AND THE LAST ONE FROM FF(128).
----
```

9.18 WREG

OPERATION NAME: Write Register

MNEMONIC: WREG K, XX(N)
WREG K, X

DESCRIPTION: The parameters are the same as those for RREG.

9.19 WRIM

OPERATION NAME: Write Relative Immediate

MNEMONIC: WRIM Z,[X]

DESCRIPTION: This statement executes the Write Relative Immediate channel program instruction. Z is the displacement from the next instruction of the channel program (-128(=Z(=127). X is the data to write into the channel program at that location. If Z is negative then X is not used. The constant used is what is already in the word at WRIM execution time (See HP-300 I/O ERS for further details).

EXAMPLE(S):

```
) 100  JUMP 110          .Jump to 130 Second Time
) 110  WRIM -3,4         .Change 100 to JUMP 130
) 120  JUMP 100
) 130  IN H
```

10.0 FUNCTION STATEMENTS

This section defines the statements used in creating programmed functions.

10.1 ENDF

OPERATION NAME: End Function Definition

MNEMONIC: ENDF

DESCRIPTION: This statement terminates a Function definition.

EXAMPLE(S): See FUNCTION statement.

10.2 GETNAMEDATA

OPERATION NAME: Get data found offset from NAME parameter

MNEMONIC: GETNAMEDATA NAMEx, offset, variable

DESCRIPTION: Provides access to the memory location offset from the pointer found in NAMEx. If a buffer was passed as the NAME parameter then the element of the buffer plus offset is stored into variable. If a buffer was not passed then an AID execution error is reported.

EXAMPLE(S):

```
 10 DB AA,100
    .
    .
100 FUNCTION DOIT NAME1
110 GETNAMEDATA NAME1,5,A .Store contents of AA(15) int
120 GETNAMEDATA NAME1,-3,B  .Store contents of AA(7) in
    .
    .
200 ENDF
    .
    .
500 DOIT AA(10)
```

10.3 GETNAMEINFO

OPERATION NAME: Get NAME parameter information

MNEMONIC: GETNAMEINFO NAMEx [,X][,Y][,Z]

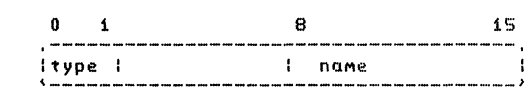
DESCRIPTION: Provides the identity of the NAME1/6 parameter including:

Type- simple variable,reserved variable,data or string buffer.
Name- A through Z or position of reserved variable i AID Reserved Variable Table.

Element- number of the buffer element passed.

Length- Size of the buffer in words.

X, if included, is stored with the following information:



type=0 for data buffers (AA-ZZ)
1 for string buffers (&AA-&ZZ)
2 for reserved variables (MAXMEMORY-FILELEN)
3 for simple variables (A-Z)

name=%101 for A,AA or &AA through %132 for Z,ZZ or &ZZ.
If type is a reserved variable then name equals the offset from the first reserved variable in memory (See AID LIST R Command for their order).

Note: if a NAME parameter is not passed then X is defaulted to that name parameters Reserved Variable.

Y, if included, is stored with the element passed if the NAME parameter was a buffer else -1.

Z, if included, is stored with the length of the buffer passed in NAMEx. If a buffer wasn't passed then Z is stored with -1.

EXAMPLE(S):

```
 10 DB AA,100


100 FUNCTION EXAMPLE NAME1,NAME2,NAME3,NAME4
110 GETNAMEINFO NAME1,A,B,C   .A=%101(ID),B=5(element),C=100(length)
120 GETNAMEINFO NAME2,D,E,F   .D=0(default parameter),E=F=-1
130 GETNAMEINFO NAME3,G,H,I   .G=%140132(ID),H=I=-1
140 GETNAMEINFO NAME4,J,K,L  .J=%10000S(5th Reserved Variable),K=L=-1
     .
     .
     .
500 EXAMPLE AA(5),,Z,STEP   .See FUNCTION EXAMPLE
```

10.4 FUNCTION

OPERATION NAME: Function Declaration

MNEMONIC: FUNCTION name [parameters]

DESCRIPTION: Defines the entry point and parameter format of subsequent function calls. The function capability enables the user to create quasi-statements with an unique name and parameters where:

name= maximum of 8 alpha characters.

parameters= Pn [,Pn.....,Pn]

where:

P= NAME for a variable or buffer passed by name.
VALUE for a constant, variable or buffer passed by value.

n= ordinal number* of P where 1 is the first parameter of the NAME or VALUE type and 1<=n<=6.

The following rules** govern FUNCTION use:

(1) Calls to the FUNCTION Statement must insure all parameter types are matched. Any parameter may be defaulted i.e. excluded, except the NAME type when it's used as a read/write buffer(e.g. RR 0,NAME1,5). Defaulted VALUE parameters are assigned the quantity 0 and defaulted NAME parameters are assigned to the Reserved Variable bearing their name.

* Example: VALUE1,VALUE2,NAME1,VALUE3,NAME2,VALUE4,NAME3,NAME4

** See the respective examples on the following pages which display rule usage.

(2) Function calls may not be input unless the appropriate FUNCTION Statement is already in the program. If a FUNCTION Statement is deleted any calls to it render the program unexecutable and a LISTing of the function calls will yield a warning message.

(3) A FUNCTION calling a FUNCTION is allowed but limited to the amount of space available to the user program (i.e. every FUNCTION call places a 13 word information block into the user area and each ENDF Statement removes just one information block).

(4) The FUNCTION Statement may never be executed in line, i.e. it must be called, and a branch into a FUNCTION-ENDF Statement sequence during execution will produce an error.

(5) All AID Statement,Command, Reserved Variable keywords (e.g. LET,TEST,etc.) and the buffer names AA to ZZ are reserved and an attempt to input a FUNCTION statement name using such a keyword will result in an error.

Limitations using functions:

(a) Use of name buffers, i.e. NAME1-NAME6, is not allowed in AID Statements that use buffers without elements, e.g. BSIO, RSIO, DB, etc.

(b) Indexing of name buffers is not allowed, i.e. NAME1(X).

Example of RULE 1 (correct way)

```
> 10 FUNCTION ADDEM NAME1,VALUE1,VALUE2
> 20 LET NAME1:=VALUE1+VALUE2
> 30 ENDF
      .
      .
>100 ADDEM A,7,2     .A:=7+2
```

Example of RULE 1 (incorrect way)

```
> 10 FUNCTION ADDEM NAME1,VALUE1,VALUE2
> 20 LET NAME1:=VALUE1+VALUE2
> 30 ENDF
      .
      .
>100 ADDEM 4,7,2
>110 RUN
** AID ERROR in Statement 40  **
FUNCTION Parameter invalid or in wrong order
```

Example of RULE 2 (correct way)

```
> 10 FUNCTION GETSR NAME1
> 20 RSW NAME1
> 30 LET NAME1:=NAME1 AND !7F
> 40 ENDF
      .
      .
      .
>100 GETSR AA(0)
>110
```

Example of RULE 2 (incorrect way)

(Assume this is the first Statement input)

```
> 10 GETSR AA(0)
         ^
** AID Entry Mode Error **
Illegal parameter, type or input
```

-or-

```
                    > 10 FUNCTION GOING NAME1,NAME2
                    > 20 ENDF
                    > 30 GOING A,B
                    > 40 DELETE 10
                    > 40 LIST

                      20 ENDF
                      30 **Undefined FUNCTION call to Statement 10

                    > 40
                   (Note- Statement 30 is supposed to be GOING A,B
                          but has no significance since Statement
                           10 was deleted. Statement 10 must be
                            restored with a FUNCTION Statement to
                              LIST or execute normally)
```

Example of RULE 3 (correct way)

```
   (Demonstrates a FUNCTION calling a FUNCTION)
       > 10 FUNCTION ADDEM NAME1,VALUE1,VALUE2
       > 20 LET NAME1:=VALUE1+VALUE2
       > 30 ENDF
       > 40 FUNCTION GETSR NAME1
       > 50 RSW NAME1
       > 60 ADDEM NAME1,NAME1,4    . Add 4 to sw. reg.
       > 70 ENDF

       >200 GETSR A    .Get sw.reg. and add 4 to it

    (Demonstrates a recursive function call)

       > 10 FUNCTION POWER NAME1,VALUE1,VALUE2,NAME2
       > 20 IF VALUE1<1 THEN 50
```

```
        ) 30 LET NAME2:=VALUE2:=NAME1*VALUE2, VALUE1:=VALUE1-1
        ) 40 POWER NAME1,VALUE1,VALUE2,NAME2
        ) 50 ENDF
             .
             .
         )200 POWER A,7,1,B  .Get A to 7th power and put in B
```

Example of RULE 3 (incorrect way)

```
        ) 10 FUNCTION FOREVER NAME1
        ) 20 FOREVER NAME1
        ) 30 ENDF
             .
        )100 FOREVER A
        )110 RUN
         ** AID ERROR in Statement 20  **
         Data buffer area overflow
```

(Statement 20 will build 13 word blocks until no more user space is available at which time the program will abort)

Example of RULE 4 (correct way)

```
        ) 10 GOTO 300  . Branch around Functions
        ) 20 FUNCTION POWER NAME1,VALUE1
             .
             .
             .
        )290 ENDF
        )300 .Start of normal program
```

Example of RULE 4 (incorrect way)

```
        ) 10 FUNCTION POWER NAME1,VALUE1
```

```
        > 20 LET NAME1:=NAME1*NAME1
        ----
        > 30 ENDF
        ----
        > 40 RUN
        ----

        ** AID Execution Mode Error in Statement 10 **
        FUNCTION Statement cannot be executed in-line
```

Example of RULE 5 (correct way)

```
        > 10 FUNCTION TESTX NAME1  .TESTX is valid
             .
             .
```

Example of RULE 5 (incorrect way)

```
        > 10 FUNCTION TEST NAME1
        ----
                             ^
        ** AID Entry Mode Error **
        Invalid FUNCTION name or reserved keyword
```

Practical I/O application

```
    >100 FUNCTION READDATA VALUE1,NAME1,VALUE2,NAME2
    ----
    >110 .Reads data into buffer NAME1 with modifier VALUE1
    ----
    >120 . and length VALUE2 and compares the read
    ----
    >130 .  data to buffer NAME2
    ----
    >140 INIT  .Intialize Device
    ----
    >150 BSIO AA  . Build Channel Program
    ----
    >160 RR VALUE1,NAME1,VALUE2  .Read record
    ----
    >170 RSIO     . Execute Channel Program
    ----
    >180 CB NAME1,NAME2,VALUE2  .Compare buffers
    ----
    >190 ENDF   .End of READDATA
    ----
         .
         .
         .
    >500 READDATA 0,AA(0),256,BB(0) .Get and test data
```

```
>510 IF INDEX=-1 THEN 550
>520 EPRINT* "Compare Error! Bad Data=";AA(INDEX);
>530 PRINTEX " Good Data=";BB(INDEX)
>540 EPAUSE
>550 .Continue Program
```

10.5 SETNAMEDATA

OPERATION NAME: Store data into a NAME buffer element

MNEMONIC: SETNAMEDATA NAMEx, offset, variable

DESCRIPTION: Stores the data in variable into the buffer element plus offset passed as a NAME parameter. If a buffer was not passed an AID execution error will occur.

EXAMPLE(S):

```
10 DB AA,100
   .
   .
100 FUNCTION DOIT NAME1
110 SETNAMEDATA NAME1,5,A .Store contents of A into AA(15)
120 SETNAMEDATA NAME1,-3,B .Store contents of B into AA(7)
   .
   .
200 ENDF
   .
   .
300 DOIT AA(10)
```

9.0 AID STATEMENTS (CHANNEL PROGRAM TYPE)

The following Channel Program Type AID Statements must be located between the BSIO and ESIO Statements. The format of each statement explanation is:

OPERATION NAME: General phrase of what the Statement does.

MNEMONIC: The form that the Statement would be called in. X is used to indicate the variables A to Z or a number. XX is used to indicate the buffers AA to ZZ. N is the same as X but is used as an index (XX(n)).

DESCRIPTION: A detailed explanation of the Statement's function.

EXAMPLE(S): One or more examples using the Statement.

9.1 CHP

OPERATION NAME: Command HP-IB

MNEMONIC: CHP V0,[V1, . . VN]

DESCRIPTION: This statement executes the Command HP-IB channel instruction. VN is the Nth HP-IB command (0<=N<=7) and is a reference to a variable or buffer element which contains the command or is the command in numeric form.

EXAMPLE(S):

```
> 10  LET CHANNEL:=5, DEVICE:=1
> 20  BSIO AA
> 30  CHP !3F,!5E,!25,!6F
> 40  .UNLISTEN, TALK 30, IDS-LISTEN, ENABLE DOWNLOA
> 50  RSIO
> 60  RUN
```

NOTE: VN (a 16-bit quantity) is converted to a byte and stored in the CHP portion of the channel program.

9.2 CLEAR

OPERATION NAME: Control Clear

MNEMONIC: CLEAR [X]

DESCRIPTION: This statement executes the Clear channel instruction. Commands the currently selected device to clear itself. If the optional X is added it forms the control byte(where 0<=X<=!FF and the default is 0) in the channel instruction.

EXAMPLE(S):

```
> 10  LET CHANNEL:=5
> 20  BSIO AA
> 30  CLEAR      .CLEAR CHANNEL 5, DEVICE 0
> 40  RSIO
```

9.3 DSJ

OPERATION NAME: Device Specified Jump

MNEMONIC:
DSJ S0[*R0][,S1[*R1]...[,SM[*RM]]...]][;XX(N)]
DSJ S0[*R0][,S1[*R1]...[,SM[*RM]]...]][;X]

DESCRIPTION: This statement executes the DSJ channel program instruction. A jump occurs as a result of the byte returned from the device. If XX(N) or X is added, then the byte returned (last byte should the DSJ execute more than once) or !FF (if the DSJ never executes) is placed in the right byte of XX(N) or X. The left byte of XX(N) or X will be set to 0. SM is the statement to execute when the returned byte of the DSJ is equal to M. SM must be in the same Channel program. *RM is the total number of jump address copies of SM to build into the DSJ instruction.

EXAMPLE(S):

```
>  5  DB BB,7,0
>  7  CPVA BB(0)          .Define CPVA
> 10  LET CHANNEL:=5      .Define Disc
> 20  BSIO  AA            .Begin Channel Program
```

```
> 30  DSJ 40,60;A          .Stuff return byte into A
> 40  IN H, 0, 7           .Error--Store halt code 7
> 50                       .In CPVA0
> 60  IN H                 .OK--Clear CPVA0
> 70  RSIO                 .Start Execution
> 80  PRINT "DSJ=;A;2;"CPVA0=";BB(0)
                           .Output Results
```

9.4 IDENT

OPERATION NAME: Identify

MNEMONIC:
```
IDENT XX(N)
IDENT X
```

DESCRIPTION: This statement executes the IDENT channel program instruction. The word returned from the device (last word should it execute more than once) or !FFFF (if it never executes) is placed in XX(N) or X.

EXAMPLE(S):
```
> 10  LET CHANNEL:=5          .Define Disc
> 20  DB BB,8              .Create Buffer
> 30  BSIO AA              .Begin Channel Program
> 40  IDENT BB(7)          .Stuff ID into BB(7)
> 50  IN H                 .Stop Execution
> 60  RSIO                 .Start Channel Program
> 70  PRINT "IDENTIFY CODE =";BB(7)
```

9.5 IN

OPERATION NAME: Interrupt Halt or Run

MNEMONIC:
```
IN H [, [X][,C]]
IN R [, [X][,C]]
```

DESCRIPTION: Executes the INTERRUPT channel program instruction. R, if used, will allow the Channel program to continue to run when this instruction is reached. H, if used, will cause the Channel program to halt when this instruction is reached. X is the CPVA offset (0 <= X <= 3). C is the code to store at CPVAX on interrupt(0<=C<=255). Default for both X and C is 0.

EXAMPLE(S):

```
>  4  DB BB,4
----
>  5  CPVA BB(0)     .DEFINE CPVA
----
>  6  LET CHANNEL:=5
----
> 10  BSIO AA             .Define the following Channel Program
----
> 20  IN R,3,1            .CPVA3 : = 1
----
> 30  IN R,2,2            .CPVA2 : = 2
----
> 40  IN R,1,3            .CPVA1 : = 3
----
> 50  IN H,,4             .Stop Program Set CPVA0 : = 4
----
> 60  RSIO                .Execute the Above Program
----
> 70  PRINT "CPVA0=";BB(0);2;"CPVA1=!BB(1)
----
> 80  PRINT "CPVA2=";BB(2);2;"CPVA3=";BB(3)
----
```

9.6 JUMP

OPERATION NAME: Direct Jump

MNEMONIC: JUMP SN

DESCRIPTION: This statement executes the JUMP channel program instruction. SN is an AID statement number. The statement number must be within the same Channel program.

EXAMPLE(S):

```
> 10  LET CHANNEL:=5          .Define Disc
> 20  BSIO AA
> 30  DSJ 40,50;A             .Does Disc respond?
> 40  JUMP 30                 .No! Wait some more.
> 50  IN H                    .Yes! Exit Channel program.
> 60  ESIO
> 70  RSIO AA
```

9.7 RB

OPERATION NAME: Read Burst

MNEMONIC: RB MOD, XX(N), BC [,[BL][,[DC=X][,[R][,[TD]]]]

DESCRIPTION: This statement executes the Read Burst channel program instruction. MOD is the device dependent modifier(0<=MOD<=!1F). If MOD>!F then Read Control is used instead of Read. XX(N) defines the initial buffer location where the data is to be stored. BC is the total number of bytes to be read. BL is the burst length (default is 1) 1<=BL<=256. Burst length is the number of bytes to read this time through the RB. DC, if added, will allow separate data buffers to be linked (chained) by using sequential RB statements. X is equal to number of links to follow. R, if added, will cause the data to be stored starting in the right byte of XX(N) (default is the left byte). TD, if added, is the statement number to which channel program execution is transferred upon successful completion of the RB.

EXAMPLE(S):

```
> 10  LET CHANNEL:=7
> 20  BSIO BB              .Begin Channel Program
> 30  RB 0,AA(0),1         .Read One Byte Into
> 40                       .Left Byte of AA(0)
> 50  IN H                 .Done
> 60  RSIO                 .Execute Channel Program
```

-or-

```
> 10  LET CHANNEL:=2
> 20  DB AA,1
> 30  BSIO BB
> 40  RB 31,AA(0),1        .Read self test results
> 50  IN H
> 60  RSIO
```

9.8 RDMAB

OPERATION NAME: READ DMA Burst

MNEMONIC: RDMAB XX(N), BC[,[BL][,R][,TD]]]

DESCRIPTION: This statement executes the Read DMA Burst channel program instruction. The parameters are the same as those for RB except the modifier and DC are deleted. See HP-300 I/O ERS for definition.

9.9 RDMAR

OPERATION NAME: READ DMA Record

MNEMONIC: RDMAR XX(N),BC [,[R][,TD]]

DESCRIPTION: This statement executes the Read DMA Record channel program instruction. The parameters are the same as those for RR except the modifier and DC are deleted. See HP-300 I/O ERS for definition.

9.10 RMW

OPERATION NAME: Read Modify Write

MNEMONIC: RMW K, BN, C
RMW K, BN, S

DESCRIPTION: This statement executes the Read Modify Write channel program instruction. K is the register to be modified (0<=K<=!F). BN is the bit number of register K to modify (0<=BN<=!F). C will clear the bit and S will set it. REGISTER K is read, bit number BN is modified, then register K is written. For some registers BN has special meaning. See HP-300 I/O System ERS for further register definition.

9.11 RR

OPERATION NAME: Read Record

MNEMONIC: RR MOD, XX(N), BC[, [DC=X][, [R][, TD]]]

DESCRIPTION: This statement executes the Read Record channel instruction. MOD is the device dependent modifier (0<=MOD<=!1F). If MOD is greater than !F then Read Control is used instead of Read. XX(N) defines the initial buffer location where the data is to be stored. BC is the number of bytes to be read. If R is added will cause the data to be stored starting in the right byte of XX(N) (default is the left byte). DC(data chain), if added, will allow separate data

buffers to be linked (chained) by using sequential RR statements. X is equal to number of links to follow. TD, if added is the statement number to which channel program execution is transferred upon successful completion of the RR.

EXAMPLE(S):

```
> 100  RR 0,JJ(0),256,DC=2  .READ 4 SECTORS. PLACE THE
-----
> 110  RR 0,BB(0),512,DC=1  . FIRST ONE IN JJ AND THE LAST
-----
> 120  RR 0,FF(128),256     .  ONE AT FF(128)
-----
```

9.12 RREG

OPERATION NAME: Read Register

MNEMONIC: RREG K, XX(N)
RREG K, X

DESCRIPTION: This statement executes the Read Register Channel instruction. K is the Channel Register to be read (0<=K<=!F). XX(N) or X is where the data is placed. If this statement doesn't execute then !FFFF is placed in X or XX(N). Should this statement execute more than once, the last value read will be placed in X or XX(N).

9.13 WAIT

OPERATION NAME: Wait

MNEMONIC: WAIT [S]

DESCRIPTION: This statement executes the WAIT channel program instruction. The channel program is suspended until the device requests service. If S is used then bit 15 of the first word of the wait instruction is set. The special mode is described in the HP-300 I/O ERS.

```
EXAMPLE(S):     > 10  LET CHANNEL:=5
                ----
                > 20  DB AA,3
                ----
                > 30  LET AA(0):=!200    .Seek Command
                ----
                > 40  LET AA(1):=100     .Cylinder 100
                ----
                > 50  LET AA(2):=!105    .Head 1,Sector 5
                ----
                > 60  BSIO BB
                ----
                > 70  WR 8, AA(0), 3     .Issued Seek
                ----
                > 80  WAIT               .Wait for Completion
                ----
                > 90  IN H               .Done
                ----
                >100  RSIO               .Start Channel Program
                ----
```

9.14 WB

OPERATION NAME: Write Burst

MNEMONIC: WB MOD, XX(N), BC[,[BL] [,[DC=X][,[R][, [E]]]]]

DESCRIPTION: This statement executes the Write Burst channel program instruction. The parameters are the same as those for RB except the TD is not valid and E is added to flag the end of each burst with the HP-IB END message.

```
EXAMPLE(S):     > 10  LET CHANNEL:=7
                ----
                > 15  DB AA,6
                ----
                > 20  BSIO BB            .Begin Channel Program
                ----
                > 30  WB 0,AA(5),1,,,R   .Write One Byte
                ----
                > 40                     .From the Right
                ----
                > 50                     .Byte of AA(5)
                ----
                > 60  IN H               .Done
                ----
                > 70  RSIO
                ----
```

```
   -or-

> 10  LET CHANNEL:=2
----
> 20  DB AA,1,0          .Control byte is 0
-----
> 30  BSIO BB
-----
> 40  WB 31,AA(0),1      .Initiate Self test
----
> 50  IN H
----
> 60  RSIO
----
```

9.15 WDMAB

OPERATION NAME: Write DMA Burst

MNEMONIC: WDMAB XX(N), BC [,[BL][,[R][,E]]]

DESCRIPTION: This statement executes the Write DMA Burst channel instruction. The parameters are the same as those for WB except the modifier and DC are deleted. See HP-300 I/O ERS for definition.

9.16 WDMAR

OPERATION NAME: Write DMA Record

MNEMONIC: WDMAR XX(N), BC[,R]

DESCRIPTION: This statement executes the Write DMA Record channel program instruction. The parameters are the same as WR except the modifier and DC are deleted. SEE HP-300 I/O ERS for definition.

9.17 WR

OPERATION NAME: Write Record

MNEMONIC: WR MOD, XX(N), BC[, [DC=N][, R]]

DESCRIPTION: This statement executes the Write Record channel program instruction. The parameters are the same as those for RR except the TD is not valid.

EXAMPLE(S):

```
> 10  WR 0,JJ (0),256,DC=2 .WRITE 4 SECTORS. GET FIRST
----
> 20  WR 0,BB(0),512,DC=1  . FROM JJ, THE NEXT TWO FROM BB
----
> 30  WR 0,FF(128),256     . AND THE LAST ONE FROM FF(128).
----
```

9.18 WREG

OPERATION NAME: Write Register

MNEMONIC: WREG K, XX(N)
WREG K, X

DESCRIPTION: The parameters are the same as those for RREG.

9.19 WRIM

OPERATION NAME: Write Relative Immediate

MNEMONIC: WRIM Z,[X]

DESCRIPTION: This statement executes the Write Relative Immediate channel program instruction. Z is the displacement from the next instruction of the channel program (-128<=Z<=127). X is the data to write into the channel program at that location. If Z is negative then X is not used. The constant used is what is already in the word at WRIM execution time (See HP-300 I/O ERS for further details).

EXAMPLE(S):

```
> 100  JUMP 110          .Jump to 130 Second Time
> 110  WRIM -3,4         .Change 100 to JUMP 130
> 120  JUMP 100
> 130  IN H
```

10.0 FUNCTION STATEMENTS

This section defines the statements used in creating programmed functions.

10.1 ENDF

OPERATION NAME: End Function Definition

MNEMONIC: ENDF

DESCRIPTION: This statement terminates a Function definition.

EXAMPLE(S): See FUNCTION statement.

10.2 GETNAMEDATA

OPERATION NAME: Get data found offset from NAME parameter

MNEMONIC: GETNAMEDATA NAMEx, offset, variable

DESCRIPTION: Provides access to the memory location offset from the pointer found in NAMEx. If a buffer was passed as the NAME parameter then the element of the buffer plus offset is stored into variable. If a buffer was not passed then an AID execution error is reported.

EXAMPLE(S):

```
 10 DB AA,100
    .
    .
100 FUNCTION DOIT NAME1
110 GETNAMEDATA NAME1,5,A .Store contents of AA(15) int
120 GETNAMEDATA NAME1,-3,B  .Store contents of AA(7) in
    .
    .
200 ENDF
    .
    .
500 DOIT AA(10)
```

10.3 GETNAMEINFO

OPERATION NAME: Get NAME parameter information

MNEMONIC: GETNAMEINFO NAMEx [,X][,Y][,Z]

DESCRIPTION: Provides the identity of the NAME1/6 parameter including:

Type- simple variable,reserved variable,data or string buffer.
Name- A through Z or position of reserved variable i AID Reserved Variable Table.

Element- number of the buffer element passed.

Length- Size of the buffer in words.

X, if included, is stored with the following information:

type=0 for data buffers (AA-ZZ)
1 for string buffers (&AA-&ZZ)
2 for reserved variables (MAXMEMORY-FILELEN)
3 for simple variables (A-Z)

name=%101 for A,AA or &AA through %132 for Z,ZZ or &ZZ.
If type is a reserved variable then name equals the offset from the first reserved variable in memory (See AID LIST R Command for their order).

Note: if a NAME parameter is not passed then X is defaulted to that name parameters Reserved Variable.

Y, if included, is stored with the element passed if the NAME parameter was a buffer else -1.

Z, if included, is stored with the length of the buffer passed in NAMEx. If a buffer wasn't passed then Z is stored with -1.

EXAMPLE(S):

```
 10 DB AA,100
                .
                .
100 FUNCTION EXAMPLE NAME1,NAME2,NAME3,NAME4
110 GETNAMEINFO NAME1,A,B,C   .A=%101(ID),B=5(element),C=100(length)
120 GETNAMEINFO NAME2,D,E,F   .D=0(default parameter),E=F=-1
130 GETNAMEINFO NAME3,G,H,I   .G=%140132(ID),H=I=-1
140 GETNAMEINFO NAME4,J,K,L  .J=%100005(5th Reserved Variable),K=L=-1
     .
     .
     .
500 EXAMPLE AA(5),,Z,STEP   .See FUNCTION EXAMPLE
```

10.4 FUNCTION

OPERATION NAME: Function Declaration

MNEMONIC: FUNCTION name [parameters]

DESCRIPTION: Defines the entry point and parameter format of subsequent function calls. The function capability enables the user to create quasi-statements with an unique name and parameters where:

name= maximum of 8 alpha characters.

parameters= Pn [,Pn.....,Pn]

where:

P= NAME for a variable or buffer passed by name.
VALUE for a constant, variable or buffer passed by value.

n= ordinal number* of P where 1 is the first parameter of the NAME or VALUE type and 1<=n<=6.

The following rules** govern FUNCTION use:

(1) Calls to the FUNCTION Statement must insure all parameter types are matched. Any parameter may be defaulted i.e. excluded, except the NAME type when it's used as a read/write buffer(e.g. RR 0,NAME1,5). Defaulted VALUE parameters are assigned the quantity 0 and defaulted NAME parameters are assigned to the Reserved Variable bearing their name.

* Example: VALUE1,VALUE2,NAME1,VALUE3,NAME2,VALUE4,NAME3,NAME4

** See the respective examples on the following pages which display rule usage.

(2) Function calls may not be input unless the appropriate FUNCTION Statement is already in the program. If a FUNCTION Statement is deleted any calls to it render the program unexecutable and a LISTing of the function calls will yield a warning message.

(3) A FUNCTION calling a FUNCTION is allowed but limited to the amount of space available to the user program (i.e. every FUNCTION call places a 13 word information block into the user area and each ENDF Statement removes just one information block).

(4) The FUNCTION Statement may never be executed in line, i.e. it must be called, and a branch into a FUNCTION-ENDF Statement sequence during execution will produce an error.

(5) All AID Statement,Command, Reserved Variable keywords (e.g. LET,TEST,etc.) and the buffer names AA to ZZ are reserved and an attempt to input a FUNCTION statement name using such a keyword will result in an error.

Limitations using functions:

(a) Use of name buffers, i.e. NAME1-NAME6, is not allowed in AID Statements that use buffers without elements, e.g. BSIO, RSIO, DB, etc.

(b) Indexing of name buffers is not allowed, i.e. NAME1(X).

Example of RULE 1 (correct way)

```
) 10 FUNCTION ADDEM NAME1,VALUE1,VALUE2
) 20 LET NAME1:=VALUE1+VALUE2
) 30 ENDF
     .
     .
)100 ADDEM A,7,2     .A:=7+2
```

Example of RULE 1 (incorrect way)

```
) 10 FUNCTION ADDEM NAME1,VALUE1,VALUE2
) 20 LET NAME1:=VALUE1+VALUE2
) 30 ENDF
     .
     .
)100 ADDEM 4,7,2
)110 RUN
** AID ERROR in Statement 40  **
FUNCTION Parameter invalid or in wrong order
```

Example of RULE 2 (correct way)

```
) 10 FUNCTION GETSR NAME1
) 20 RSW NAME1
) 30 LET NAME1:=NAME1 AND !7F
) 40 ENDF
     .
     .
     .
)100 GETSR AA(0)
)110
```

Example of RULE 2 (incorrect way)

(Assume this is the first Statement input)

```
) 10 GETSR AA(0)
          ^
** AID Entry Mode Error **
Illegal parameter, type or input
```

-or-

```
                    > 10 FUNCTION GOING NAME1,NAME2
                    > 20 ENDF
                    > 30 GOING A,B
                    > 40 DELETE 10
                    > 40 LIST

                      20 ENDF
                      30 **Undefined FUNCTION call to Statement 10

                    > 40
                   (Note- Statement 30 is supposed to be GOING A,B
                          but has no significance since Statement
                           10 was deleted. Statement 10 must be
                            restored with a FUNCTION Statement to
                             LIST or execute normally)
```

Example of RULE 3 (correct way)

```
   (Demonstrates a FUNCTION calling a FUNCTION)
        > 10 FUNCTION ADDEM NAME1,VALUE1,VALUE2
        > 20 LET NAME1:=VALUE1+VALUE2
        > 30 ENDF
        > 40 FUNCTION GETSR NAME1
        > 50 RSW NAME1
        > 60 ADDEM NAME1,NAME1,4    . Add 4 to sw. reg.
        > 70 ENDF

        >200 GETSR A    .Get sw.reg. and add 4 to it

    (Demonstrates a recursive function call)

        > 10 FUNCTION POWER NAME1,VALUE1,VALUE2,NAME2
        > 20 IF VALUE1<1 THEN 50
```

```
        > 30 LET NAME2:=VALUE2:=NAME1*VALUE2, VALUE1:=VALUE1-1
        > 40 POWER NAME1,VALUE1,VALUE2,NAME2
        > 50 ENDF
             .
             .
         >200 POWER A,7,1,B  .Get A to 7th power and put in B
```

Example of RULE 3 (incorrect way)

```
        > 10 FUNCTION FOREVER NAME1
        > 20 FOREVER NAME1
        > 30 ENDF
             .
        >100 FOREVER A
        >110 RUN
         ** AID ERROR in Statement 20  **
         Data buffer area overflow
```

(Statement 20 will build 13 word blocks until no more user space is available at which time the program will abort)

Example of RULE 4 (correct way)

```
        > 10 GOTO 300  . Branch around Functions
        > 20 FUNCTION POWER NAME1,VALUE1
             .
             .
             .
        >290 ENDF
        >300 .Start of normal program
```

Example of RULE 4 (incorrect way)

```
        > 10 FUNCTION POWER NAME1,VALUE1
```

```
        > 20 LET NAME1:=NAME1*NAME1
        ----
        > 30 ENDF
        ----
        > 40 RUN
        ----

        ** AID Execution Mode Error in Statement 10 **
        FUNCTION Statement cannot be executed in-line
```

Example of RULE 5 (correct way)

```
        > 10 FUNCTION TESTX NAME1  .TESTX is valid
             .
             .
```

Example of RULE 5 (incorrect way)

```
        > 10 FUNCTION TEST NAME1
        ----
                           ^
        ** AID Entry Mode Error **
        Invalid FUNCTION name or reserved keyword
```

Practical I/O application

```
    >100 FUNCTION READDATA VALUE1,NAME1,VALUE2,NAME2
    ----
    >110 .Reads data into buffer NAME1 with modifier VALUE1
    ----
    >120 . and length VALUE2 and compares the read
    ----
    >130 .  data to buffer NAME2
    ----
    >140 INIT  .Intialize Device
    ----
    >150 BSIO AA  . Build Channel Program
    ----
    >160 RR VALUE1,NAME1,VALUE2  .Read record
    ----
    >170 RSIO     . Execute Channel Program
    ----
    >180 CB NAME1,NAME2,VALUE2  .Compare buffers
    ----
    >190 ENDF   .End of READDATA
    ----
         .
         .
         .
    >500 READDATA 0,AA(0),256,BB(0) .Get and test data
```

```
>510 IF INDEX=-1 THEN 550
>520 EPRINT* "Compare Error! Bad Data=";AA(INDEX);
>530 PRINTEX " Good Data=";BB(INDEX)
>540 EPAUSE
>550 .Continue Program
```

10.5 SETNAMEDATA

OPERATION NAME: Store data into a NAME buffer element

MNEMONIC: SETNAMEDATA NAMEx, offset, variable

DESCRIPTION: Stores the data in variable into the buffer element plus offset passed as a NAME parameter. If a buffer was not passed an AID execution error will occur.

EXAMPLE(S):

```
10 DB AA,100
    .
    .
100 FUNCTION DOIT NAME1
110 SETNAMEDATA NAME1,5,A .Store contents of A into AA(15)
120 SETNAMEDATA NAME1,-3,B .Store contents of B into AA(7)
    .
    .
200 ENDF
    .
    .
300 DOIT AA(10)
```

QUICK REFERENCE FOR AID USERS

****
STATEMENTS
****

MNEMONIC	PARAMETERS	OPERATION
A[SSIGN]	buf(e)[,(rpt)],data [,....,data]	BUFFER ASSIGNMENT
ADDRESSOFF	none	SET R/W WORD4 BIT4
ADDRESSON	none	CLEAR R/W WORD4 BIT4
BUMP	[;suppress cr/lf][Hold passcount inc]	INCREMENT PASSCOUNT
B[GIO]	buf[,copy number]	BUILD CHANNEL PROGRAM
CB	buf(element),buf(element),length	COMPARE BUFFERS
CHP	data byte1 [,.....,data byte8]	COMMAND HPIB
CLEAR	[control byte]	CONTROL-CLEAR
.	any ascii string	COMMENT ONLY
CPVA	data buffer(element)	SET CPVA BUFFER
COPY	buf[*copy number]	COPY CHANNEL PROGRAM
DB	buf,length[,initialize data]	DEFINE BUFFER
DELAY	delay count in 86 us increments	DELAY
DSJ	label[*rpt][,..,label[*rpt]][;ret, byte, dest]	DEVICE SPECIFIED JUMP
ENABLE	none	ENABLE ERROR PRINT
END	none	TERMINATE PROGRAM
ENDF	none	END OF FUNCTION
EPAUSE	none	ERROR PAUSE
EPRINT	[*no pause] string data,variable or space ct[;or,][..]	ERROR PRINT WITH PAUSE
E[GIO]	none	END CHANNEL PROGRAM
FILENAME	string buffer(ele)[,sector offset]	SET FILENAME POINTER
F[OR]	assignment expr [STEP expr] UNTIL[orTO) expr	START FOR-NEXT LOOP
GETNAMEDATA	NAMEx, offset variable	GET NAME DATA
GETNAMEINFO	NAMEx, variable, variable, variable	GET NAME PARAMETER INFO
G[OSUB]	label	GO TO SUBROUTINE
GOTO	label	UNCONDITIONAL BRANCH
HIOP	none	HALT CHANNEL PROGRAM
IDENT	returned word destination	DEVICE IDENTIFY
IF	expr [sp.op. expr[sp.op. expr]] THEN label	CONDITIONAL BRANCH
IFN	same as IF	CONDITIONAL BRANCH/NOT
IN	Run[or Halt][,cpva word pntr][,interrupt code]	INTERRUPT
INIT	none	INITIALIZE CHANNEL
I[NPUT]	variable[,.....,variable]	INPUT DATA
INPUTB	data buffer(element)	VARIABLE LENGTH INPUT
IOCL	none	I/O CLEAR
IOFF	none	INTERRUPT SYSTEM OFF
ION	none	INTERRUPT SYSTEM ON
JUMP	label	CHANNEL PROGRAM JUMP
[LET]	variable:=[...variable:=]expr[,.....]	ASSIGNMENT
LOCATE	[(copy number),] label[(offset)],destination	FIND INSTRUCTION
LOOPTO	statement	END OF LOOP
LPOFF	none	PRINTER OFF
LPON	none	PRINTER ON

```
N[EXT]      label                                                  END OF FOR-NEXT
NOCHECKS    none                                                   STOP CHECKS
PAGE        none                                                   PRINTER PAGE
PAUSE       none                                                   NON-ERROR PAUSE
PP[RINT]    same as EPRINT                                         NON-ERROR PR W/ PAUSE
PR[INT]     string data,variable or space count[;or,][..]          NON ERROR PRINT
PRINTEX     same as PRINT                                          ERROR PRINT
PROC        [No proceed]                                           I/O WAIT CONTROL
RANDOM      [(arguement),] variable[,...,variable]                 GENERATE RANDOM NUMBER
RB          modifier,buf(e),byte count[,[burst length]]))
              [,[DC=data chain link][,[Right byte])))))))
               [,[label[td]][,[Pause after burst]]]]]              READ BURST
RDMAB       buf(e),byte count[,[burst length][,[Right byte]))
              [,[label[td]]]]                                     READ DMA BURST
RDMAR       buf(e),byte count[,[Right byte][,label[td]]]           READ DMA RECORD
RDRT        drtword number,destination                             READ DRT DATA
READCLOCK   variable                                               READ SYSTEM CLOCK
READFILE    buf(element),length                                    READ FILE
R[ETURN]    none                                                   RETURN TO GOSUB
RIOC        register number,destination[,control byte]             READ I/O CHANNEL
RMSK        destination(variable)                                  READ MASK
RMW         register number,bit number,Clear(or Set)               READ-MODIFY-WRITE
ROCL        destination(variable)                                  CHANNEL ROLL CALL
RR          modifier,buf(e),byte count[,[DC=data chain link])
              [,[Right byte][,label[td]]]]                        READ RECORD
RREG        register number,destination(variable)                  READ REGISTER
RSIO        [buf[,[copy number]][,label]]                          RUN CHANNEL PROGRAM
RSW         destination(variable)                                  READ SWITCH REGISTER
SECTION     section number,label                                   START OF SECTION
SETNAMEDATA  NAMEx, offset, data                                   STORE NAME DATA
SMSK        mask value                                             SET MASK
SPACE       line count                                             PRINTER SPACE
SPACESOFF   none                                                   NORMAL NUMBER PRINT
SPACESON    none                                                   LEADING SPACES ON # PR
STARTCLOCK   clock increment                                       START SYSTEM CLOCK
SUPPRESS    none                                                   SUPPRESS ERRORS
UPDATEOFF   none                                                   SET R/W BITS WORD 4
UPDATEON    none                                                   CLEAR R/W BITS WORD4
WAIT        [Set bit 15]                                           CHANNEL PROGRAM WAIT
WB          modifier,buf(e),byte count[,[burst length]]))))
              [,[DC=data chain link][,[Right byte])))))))
               [,[Eot flag][,[Pause after burst]]]]               WRITE BURST
WDMAB       buf(e),byte count[,[burst length])))))))))))))
              [,[Right byte][,[Eot flag]]]                        WRITE DMA BURST
WDMAR       buf(e),byte count[,Right byte]                         WRITE DMA RECORD
WIOC        register number, data                                  WRITE I/O CHANNEL
WR          modifier,buf(e),byte count[,[DC=data chain link]
              [,Right byte]]                                      WRITE RECORD
WRIM        offset[,data]                                          WRITE RELATIVE IMMED.
WRITEFILE   buf(ele),length                                        WRITE FILE
ZEROESOFF   none                                                   NORMAL NUMERIC PRINT
ZEROESON    none                                                   LEADING 0'S ON # PRINT
```

****
COMMANDS
****

MNEMONIC	PARAMETERS	OPERATION
CREATE	filename,length in sectors	CREATE FILE
DELETE	label[/label]	DELETE LINES
EEPR	none	ENABLE ERROR PRINT
EEPS	none	ENABLE ERROR PAUSE
ENPR	none	ENABLE NON-ERROR PRINT
ENPS	none	ENABLE NON-ERROR PAUSE
EP	none	ERASE PROGRAM
EXIT	none	LEAVE PROGRAM
GO	[parameter1][,..,parameter3]	CONTINUE PROGRAM
INC	increment value	CHANGE STMT INCREMENT
LC	none	LIST COMMANDS
LF	[PRINTER]	LIST FILES
L[IST]	[PRINTER][data type(!or%)]))) [Reserved var(or C(passcount) or Variable or))) Buffer)][,parameter1[/(or,)][parameter2])))] [/parameter3]	LIST PROGRAM/DATA
LOAD	filename	GET PROGRAM FROM DISC
LOOP		SET LOOP FLAG
LOOPOFF		CLEAR LOOP FLAG
M[ODIFY]	same as LOOP	MODIFY STATEMENT
PURGE	filename	PURGE FILE
REN	[renumber value]	RENUMBER PROGRAM
RST	none	RESET FLAGS
RUN	[parameter1][,..,parameter3]	RUN PROGRAM
SAVE	filename [revision (00.00)]	SAVE PROGRAM ON DISC
SEPR	none	SUPPRESS ERROR PRINT
SEPS	none	SUPPRESS ERROR PAUSE
SET	statement number	NEW STATEMENT NUMBER
SNPR	none	SUPPRESS NON-ERR PRINT
SNPS	none	SUPPRESS NON-ERR PAUSE
SO	none	STOP STREAMING
TEST	[+ or -][[section number][/(or,)][...[/(or,)]))) section number]][or ALL]	SELECT TEST

****
RESERVED VARIABLES
****

MNEMONIC	CONTENTS
BADINTP	channel/device number of device that interrupted illegally
CHANNEL	channel number of device under test
CONCHAN	channel/device number of console device
DEVICE	device number of device under test
FALSE	0 to designate a false value
FILEINFO	Bit 0=protected(if 1), Bit 8/11=class, Bit 12/15=type.
FILELEN	length of file declared in FILENAME
GOPARAM1/3	first,second and third GO parameters
INDEX	CB statement result (-1=good CB else element where CB failed)
INPUTLEN	length (in characters) of last operator input during INPUT
MAXMEMORY	remaining memory left to user
NAME1/6	FUNCTION name parameters.
NEWTEST	true if TEST X entered by operator else false
NOINPUT	true if non-error print is not allowed
NORESPONS	0=AID will handle bad response from I/O else store it with the value of I/O error
OFFSET	number of stmts (+or-) from GOSUB to return to on RETURN
PASSCOUNT	number of program passes (determined by BUMP executions)
RUNPARAM1/3	first,second and third RUN parameters
SECTION	current test section in execution (or last executed)
SECTIONS1/3	bit mask for test sections to execute (1=ok to execute): (SECTIONS1=tests 1-16) (SECTIONS2=tests 17-32) (SECTIONS3=tests 33-48)
STATENUM	statement # of most recently executed FUNCTION-calling statement.
STEP	current step number (determined by LET STEP:=number)
TIMEOUT	during channel program timeout allows ~5 secs*(TIMEOUT-1) delay before error (if negative then no timeout is desired)
TRUE	-1 to designate true value
VALUE1/6	FUNCTION value parameters.

****
OPERATORS
****

OPERATOR	MEANING
<>	not equal to
=	equal to (equivalance test only)
<	less than
>	greater than
>=	greater than or equal to
*	multiply
+	addition
-	subtraction (or two's complement addition)
/	divide
:=	assignment (LET var:=expression)
OR	logical OR
NOT	one's complement
MOD	remainder of divide (5 MOD 3 equals 2)
AND	logical AND
XOR	logical XOR
LSL	logical shift left (bit 0 lost, 0 into bit 15)
LSR	logical shift right (bit 15 lost, 0 into bit 0)
ASL	arithmetic shift left (bit 0 saved,bit 1 lost, 0 into bit 15)
ASR	arithmetic shift right (bit 0 saved,bit 0 to bit 1,bit 15 lost)
CSL	circular shift left (bit 0 to bit 15, nothing lost)
CSR	circular shift right (bit 15 to bit 0, nothing lost)
EQ	special operator (equal to)
NE	special operator (not equal to)
LT	special operator (less than)
GT	special operator (greater than)
LE	special operator (less than or equal to)
GE	special operator (greater than or equal to)

****
MODIFY INFO
****

CONTROL CHARACTER	MEANING
R	Replace with following characters
I	Insert following characters
D	Delete this character
E	Exit (delete old statement, add new statement)
J	Join (save old statement, add new statement)
A	Abort (current edit statement unchanged)

SECTION 442

IOMAP

(INPUT/OUTPUT MAP)

PROGRAM

EXTERNAL REFERENCE SPECIFICATION

Program Revision: 01.XX

References:
- Subsystem Summaries [Handbook Sections 200-299]
- HP-IB Summary [Handbook Section 161]
- HP-IB Tutorial [Handbook Section 761]

NOTICE

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

Hewlett-Packard Company
19447 Pruneridge Ave., Cupertino, California 95014 U.S.A.

Printed in U.S.A.

TABLE OF CONTENTS

LIST OF FIGURES

1.0 INTRODUCTION

1.1 General

The IOMAP utility has three purposes:

(1) It provides a display of the system physical I/O configuration

(2) It checks out the basic hardware I/O system

(3) It provides Identify , Remote Self-Test, and HP-IB Loopback device tests.

All channels on the IMB are identified. The HP-IB Identify feature is then used to obtain the ID codes for the devices connected to each GIC. A brief description of the channels and devices is displayed in a table.

This identification process tells you that:

(1) the I/O system (IMB and HP-IBs) is fundamentally working

(2) no two channels and devices have the same address

(3) the expected channels and devices are present and correctly configured.

In addition to the I/O configuration display, IOMAP has three selectable test sections available to the operator when in the optional mode. This optional operating mode allows you to perform Identify, Remote Self-Test, and HP-IB Loopback tests on selected devices. For intermittent problems, any one of these functions can be looped.

This program is written in the AID language.

1.2 Required Hardware

Hardware required to run the Diagnostic/Utility System. (DUS)

1.3 Required Software

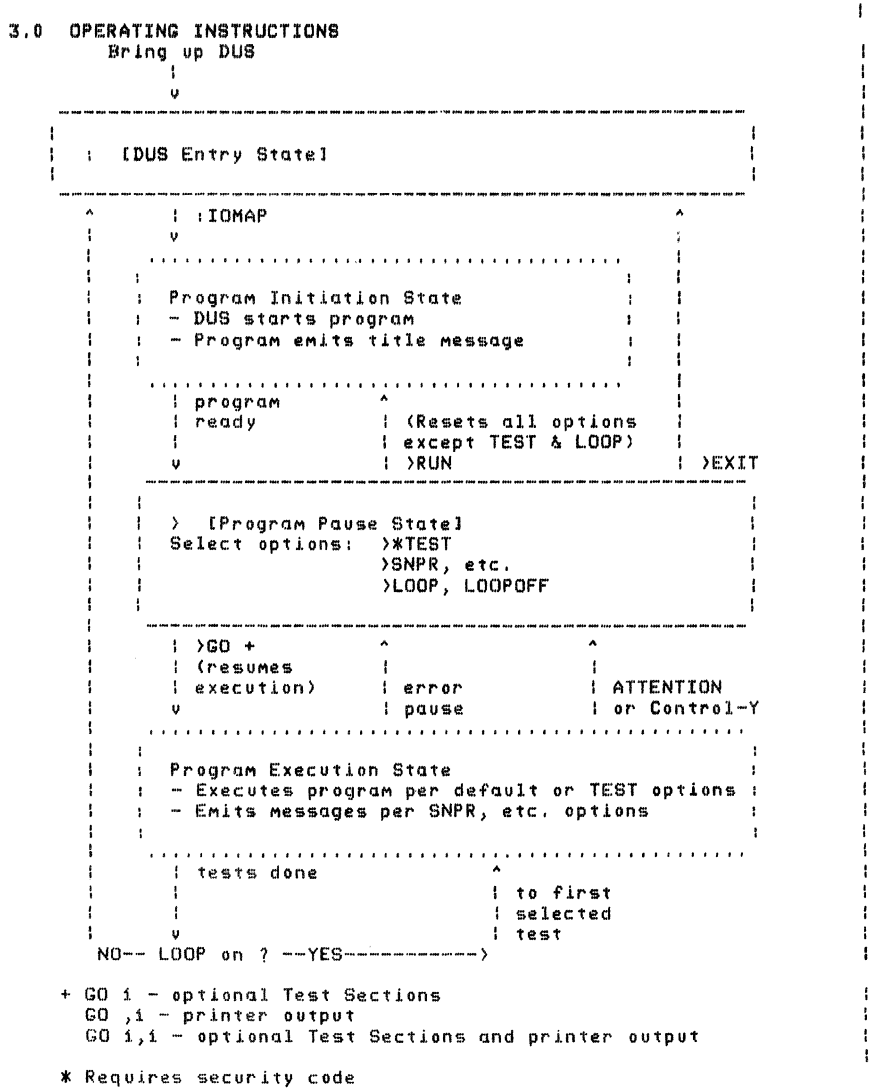
Diagnostic/Utility Package flexible disc

2.0 TEST LIMITATIONS

Channel identification depends on correct operation of the IMB and the CPU's ability to read the Configuration Registers on channels. This involves circuitry on every board connected to the IMB -- not just those explicitly involved in the transaction.

Device identification depends on correct operation of the HP-IB -- this involves circuitry in every device connected to the bus, not - just the controller and device being identified.

3.0 OPERATING INSTRUCTIONS



\+ GO i - optional Test Sections
GO ,i - printer output
GO i,i - optional Test Sections and printer output

* Requires security code

Figure 3.1 - Program State Diagram

Before running this program be sure that the physical configuration (switches on the frontplane) matches the logical configuration required by the operating system.

There are two modes of operation available for this diagnostic program -- the standard (default) mode and the optional mode.

To operate in either mode, follow these steps:

(1) Bring up the Diagnostic/Utility System (DUS)

(2) The DUS prompt character (:) is displayed. (DUS Entry State in the State Diagram, Figure 3.1)

(3) Respond 'IOMAP', to load the IOMAP Program.

(4) The program displays its title message and prompt character ')' (Program Pause State in the State Diagram.)

```
     "IOMAP   REVISION XX.XX"

     Enter 'GO' to continue
           'GO,1' to continue with printer output

           'GO 1' for Optional Test Sections
           'GO 1,1' to run Optional Sections with printer output

           ('LC' to list Commands)
)
```

The next step (step 5) depends on whether you want the Standard Mode or not.

3.1 Standard Mode

(5) If the standard mode is to be executed, enter 'GO' (or 'GO,1' to get printer output).

The standard mode is defined as follows:

(a) Execute only the I/O Map (Test Section 1)

(b) Display error, information, and prompt messages.

(c) Pause on errors and prompts.

IOMAP will display the system I/O configuration table then terminate.

3.2 Optional Mode

In addition to the I/O configuration display, IOMAP has three selectable test sections available to the operator. Entry is done when in the optional mode or entry into these test section can be accomplished via the AID 'TEST' command. The AID 'TEST' command requires knowledge of the security code. Each of the optional test sections will request the operator to enter a channel and the device number. After the operator enters a legal channel and device number and execution of the selected test section completes, the operator will be returned to the entry point of the selected test section. The request for a channel and device number only occurs on the first pass through the optional test sections. Therefore, entering the AID 'LOOP' command, does not force the operator to re-enter the desired channel and device number as each pass is made.

(S) To enter the optional mode, enter 'GO 1' or 'GO 1,1'. In response to the welcome message, the program displays:

```
IOMAP                          Optional Test Sections

Test Section 2                       Identify
Test Section 3                       Self-Test
Test Section 4                       Loopback

Enter Number for Desired Test Section

?
```

At this point, select the appropriate Test Section by typing its number.

3.3 Messages

Several kinds of messages may be displayed by IOMAP:

General messages: Request action from the operator or report results of commands.

Error messages: Response to the operator's requested function was not expected or operator entered incorrect information in a command.

I/O table: All responding channels and devices are displayed in ascending order according to their respective channel address (CHAN ADDR switch position) and device addresses (DEVICE ADDR switch position).

switch position) and device addresses
(DEVICE ADDR switch position).

Sample I/O table:

```
 IOMAP            SYSTEM I/O CONFIGURATION
-------------------------------------------------------------------
)Control panel switch settings: Channel=7  Device=4
)System console is device 4 on channel 7
-------------------------------------------------------------------
Channel 7  ID=!0       General I/O Channel  (GIC)
  Device 3 ID=!1234    XXX Device     (CODE=2)
  Device 4 ID=!4321    YYY Device
-------------------------------------------------------------------
Channel 4   ID=!1       Async. Data Comm. Channel (ADCC)
  Devices 0-3  ID=!4080  Devices on ADCC MAIN   (CODE=1,2)
  Devices 4-7  ID=!4080  Devices on ADCC EXTEND (CODE=1,2)
-------------------------------------------------------------------
Explanation of '(CODE=    )'
          1 implies:   NO LOOPBACK Capability
          2 implies:   NO SELFTEST Capability
-------------------------------------------------------------------

End of pass n
```

Note: The devices on the ADCC MAIN and ADCC EXTEND are not individually identifiable. The ADCC responds to the IDENTIFY command with !4080.

4.0 TEST SUMMARY

Std.	Sect	Description	Time	Notes
*	1	I/O Map	5 sec	
	2	Identify a specific device	ms	+
	3	Self-Test a specific device	ms	+
	4	Loopback to a specific device	ms	+

Notes: * Part of standard set of sections

+ Keeps prompting user for channel and device addresses until '-2' is entered.

Figure 4.1 - Table of Test Sections

5.0 TEST DESCRIPTIONS

Section 1 - I/O Table

The program performs the following sequence for each channel.

(1) Perform Roll Call on the IMB for specific channel type

(2) Read Register of the channel

(3) Perform ID sequence on the channel's HP-IB.

Section 2 - Identify

This test section will display the channel and device ID code and type when executed. The following is displayed upon entry of Test Section 2:

```
          Test Section 2---IDENTIFY

Function:  To describe the device on a specific Channel
           Enter a channel and device number separated by
           a comma or, -2 to EXIT this test section.

    ?
```

The AID prompt character '?', awaits the operator's response:

a. Upon entry of a legal channel and device number the test executes.

b. Upon entry of '-2' the utility returns the operator to the main prompt.

```
          IOMAP   REVISION XX.XX

    Enter 'GO' to continue
          'GO , 1' to continue with Line Printer

          'GO 1' for Optional Test Sections
          'GO 1,1' to run Optional Test Sections with Line Printer

          ('LC' to list Commands)
    >
```

At this time the operator may enter the AID 'EXIT' command to EXIT the IOMAP utility, select another test section, or enter 'GO' to continue.

Section 3 - Self-Test

In this test section, the following sequence is sent to a selected channel/device:

Initiate Self-Test, read self-test results, and display self-test results. Test Section 3 begins with the following title and question:

```
        Test Section 3----SELF TEST

Function:  To Invoke SELF TEST.
       Enter a channel and device number separated by a comma or,
       -2 to EXIT this test section.

    ?
```

The AID prompt character, '?', awaits the operator's response. Upon entering a legal channel/device number execution of this test section begins. (Entering a -2 causes same reaction as described in Test Section 2). The Self-Test results are displayed upon completion of the test section. The results displayed on the first pass are used as the basis for comparing subsequent pass results. On the first pass through this section the following message is displayed:

```
    Initial SELF TEST Results = !XXXX
```

All supported devices have a good Self-Test result of 0 except the 31213 IDS, which responds !1D.

On subsequent passes the following message is displayed:

```
    New Self-Test Result Equivalent to Initial Result of !XXXX
```

or,

```
    Self-Test Results changed on Pass X, expected !XXXX Received
    !XXXX.
```

NOTE: Not all devices will have Self-Test Capability.

Section 4 - Loopback

This test section attempts the HP-IB loopback function on the selected channel/device. This test section begins as follows:

```
        Test Section 4----LOOPBACK

Function:  To perform the LOOPBACK test -

               Enter a channel and device number separated
               by a comma or,
               -2 to EXIT this Test section.

    ?
```

The AID prompt character, '?', awaits the operator's response.

Upon entering a legal channel and device number this test section begins execution. (Entering a -2 causes the same reaction as described in Test Section 2.) Upon completion of this test, the following is displayed:

```
    LOOPBACK test completed
```

or

```
    LOOPBACK ERROR:  PASS X BYTE A
    Received !D sent !C
    LOOPBACK Test has completed.
```

Note 1: Not all devices have loopback capability.

Note 2: Loopback is not allowed for the device acting as system console.

6.0 ERROR INTERPRETATION

6.1 I/O Table Errors

If a device responds with an ID code not recognized, the following message is displayed in the description field of the I/O configuration list.

"Device responds but ID code undefined".

Possible causes:

(1) Device is not responding with correct ID code. Look up correct code in Section 7.0; check for stuck bits.

(2) Device is not supported on system. Check current Price/Configuration Guide brochure.

(3) Device is newly supported on system and IOMAP program copy is not up-to-date.

A non-recognized channel type will cause the following message:

"ID=!XXXX **Undefined Channel ID code."

XXXX= ID code of channel.

Possible causes:

(1) Operator error

(2) Channel is identifying incorrectly. Look up correct code in Section 7.0; check for stuck bits.

(3) Channel is newly supported on system and IOMAP copy is not up-to-date.

6.2 Optional Mode Errors

"Device X does not exist on Channel Y.
Enter an existing channel and device number separated by a comma or,
Enter '100' to run IOMAP again."

?

The AID prompt character, '?', awaits the operator's response. Each test section will stay in this loop (requesting entry of a channel and device number) until a legal channel and device

number has been entered. The three Optional Test Sections are described in detail in Test Descriptions (Section 5.0).

Possible causes:

(1) Operator Error

(2) Channel or Device is intermittently failing to identify.

7.0 REFERENCE TABLES

7.1 Supported Channels

IOMAP currently recognizes the following channels:

```
GIC (31262)
ADCC MAIN (31264)
ADCC MAIN with EXTENDER  (31264, 31265)
```

Figure 7.1 - Channel ID Codes Recognized

7.2 Supported Devices

IOMAP currently recognizes the following devices:

ID code	Device
!0081	7902 Flexible Disc Unit (FDU)
!0001	7910 Disc Storage Unit (DSU)
!0002	12745 HP-IB Adapter for 13037 Disc Controller
!2001	2608A Line Printer
!2002	2631A Serial Printer
!2009	2631B Serial Printer
!2080	Information Display System (or 31030A Workstation)
!4080	ADCC Devices
!6000	GIC as device
!8000	PMPI

Figure 7.2 - Device ID Codes Recognized

-hp-

SECTION 446

HP 300

SYSTEM I/O EXERCISER

EXTERNAL REFERENCE SPECIFICATION

Product No. 31409A

Program Revision 00.02 and later

References:
- I/O Overview [Handbook Section 7621
- Control Program IMS

Hewlett-Packard Company
19447 Pruneridge Ave., Cupertino, California 95014 U.S.A.

Printed in U.S.A.

TABLE OF CONTENTS

LIST OF FIGURES

1.0 INTRODUCTION

The System I/O Exerciser is a stand-alone program which tests the I/O system by generating the I/O activity of a heavily loaded system. It can be used to determine to what extent the hardware and low-level system software are working.

The program resides on its own cold-load flexible disc.

The program is written in SPL-II, and uses the low-level system software to do I/O. See Figure 1.1 for a description of the system software used. The program is self-configuring; i.e., on start-up it tries to identify each I/O device present, and sets up default attributes for each device that it finds. It then starts doing I/O to the various devices. If the program runs with no errors, you can assume that the software and hardware are interacting correctly, and any problems manifested at the operating system level probably exist in higher level software.

The program is most useful for generating random I/O tests which should catch time-dependent bugs; for intensive testing of specific devices or specific areas of discs; and for testing various I/O configurations (e.g. two ADCC Mains vs one ADCC Main and one ADCC Extender).

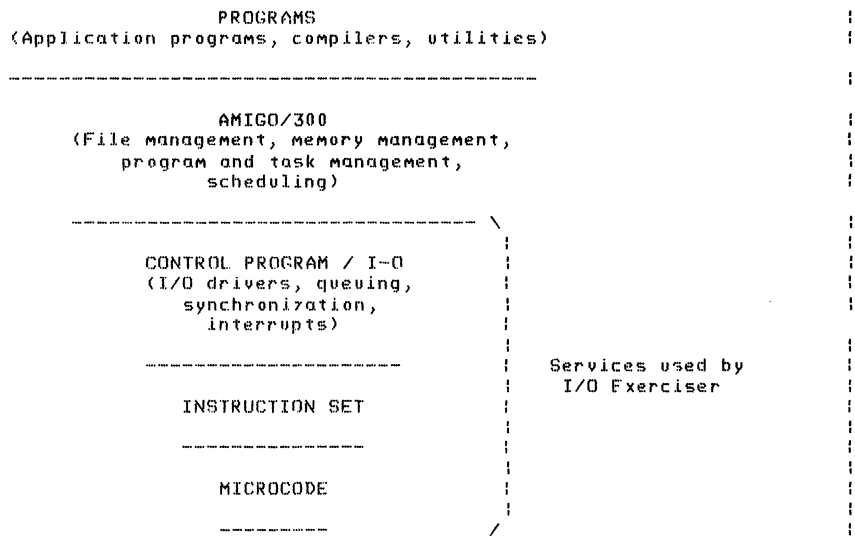


Figure 1.1 - Level of Exerciser in System

2.0 LIMITATIONS

The System I/O Exerciser is really a mini-operating system, and therefore requires a minimum set of system modules to be present and working.

(1) You must be able to cold load from a flexible disc, which means the CPU, IMB, GIC, and 7902A must be working to some extent.

(2) The IDS must be working.

(3) Other devices and channel hardware must be working well enough to allow channel programs to execute.

The program performs only reads and writes to the various devices, allowing you no control over the data actually transferred. Thus the program cannot be used to check the effect of various control sequences to the IDS or the printer.

The program also sets limitations on the length of the data that can be written to the various devices. These limitations currently are:

1) 1280 words to a disc (10 sectors)
2) 132 bytes to a line printer
3) 82 bytes to a terminal, the last 2 of which are always a carriage return/line feed
4) 255 bytes to an IDS, the first 4 of which are always "esc*@0" (write data)

You can specify that you want to write fewer bytes or words to a device, but you cannot specify more. Also note that the program will always write at least two bytes to a terminal, and at least 4 bytes to an IDS.

The program handles a maximum of 24 devices. The devices currently supported include

IDS
7902 disc (single-density, single- and double- sided)
7910 disc
13037 controller, with any mix of up to 8 7906 and 7920 drives
2631 printer
terminals connected via the ADCC

The mix of devices that can be handled is limited by memory space. There are 38000 bytes of data storage space available to the devices; individual space requirements for each device are

IDS	800 bytes
disc	3200 bytes
ADCC main or extender	2400 bytes
printer	800 bytes

Examples to explain this limitation:

A configuration of 16 terminals, 1 IDS, 5 disc drives, and 2 line printers is supportable by the program, because it totals 24 devices using

4*2400 + 1*800 + 5*3200 + 2*800 = 28000 bytes

A configuration of 1 IDS, 9 discs, and 12 terminals is also supportable by the program, because it totals 22 devices using

1*800 + 9*3200 + 3*2400 = 36800 bytes

But you could not add another disc to this configuration, because even though you would have only 23 devices, you would exceed the data space limitation.

3.0 OPERATING INSTRUCTIONS

3.1 Bringing up the Program

(You load it just like a Diagnostic/Utility flexible disc.)

(1) Insert the flexible disc. Set the Control Panel to indicate the channel and device number of the floppy drive.

(2) Press HALT, SYSTEM RESET, and LOAD in that order. The RUN light should blink quickly, indicating a soft halt. Now set the Control Panel to indicate the channel and device numbers of the IDS.

(3) Press RUN.

Wait about 20 seconds. If nothing appears on the IDS, check the CHANNEL and DEVICE switches, and try repeating from step 1 again. The IDS should display a table similar to that shown in Figure 3.1. This table contains an entry for each device that the program has identified as being on the system (identification is done using a channel program which does an Identify to each device location).

Device Table

CHAN/ DEV #	UNIT #	DEVICE TYPE	DELAY (MSECS)	DATA LEN	SECTOR ADDRESS RANGE	(DISC ONLY) INCREMENT	
4 0		TERM	0	0-80 S	9600BAUD		
4 1		TERM	0	0-80 S	9600BAUD		
4 2		TERM	0	0-80 S	9600BAUD		
4 3		TERM	0	0-80 S	9600BAUD		
7 0	0	7906	0	128-1280 R	0-76799	RANDOM	READ
7 1		7910	0	128-1280 R	0-47039	RANDOM	READ
7 2		7902D	0	128-1280 R	0-4498	RANDOM	READ
7 4		IDS	0	0-80 S			

Do you want to change any parameters?

Figure 3.1 -- Example of Device Table and Parameters

Discs attached to the 13037 controller must be powered on and ready (i.e., DRIVE READY light on) in order to be included in the exercise. Discs which come ready after the program has been loaded are ignored.

3.2 Interpreting the Device Table

Each line in the table describes one device. The hardware channel and device numbers that the device is connected to appear first. Next, if the device is a disc connected to a 13037 controller, you will see a unit number. Then you will see the device type (e.g., 7902, 7910, 2631). Devices which are attached to the system, but that are not supported by the program, are ignored. See Section 2, Limitations, for a list of supported devices.

You cannot add to the initial list of devices, nor can you change the configuration, once the program is loaded. If you wish to add to or otherwise modify the configuration, you must halt the system, make your hardware change, and then reload the program.

The initial device table tells you that you have a double-sided (7902D) flexible disc drive on your system (if the drive is connected to the HP-IB). You can use either a single-sided or a double-sided flexible disc in the drive; before the exerciser begins accessing the 7902, it checks which type of flexible disc is in the drive and accesses the drive according to the flexible disc type. It is up to you to ensure that the address field (see below) contains the appropriate range. If there is no flexible disc in the drive, the program deletes the device from the exercise and you must reactivate it as described in steps 4 and 5a below.

In addition to the fields mentioned above, you will see several other entries in the table. These entries are labelled across the top of the table, and specify various parameters that control the I/O.

The DELAY field is used to specify the time to pause after accessing the device. If this field contains a 0, no pause will occur. If there is one number in the DELAY field, then accesses will occur at fixed intervals. If there are two numbers, separated by a '-', then accesses will occur at random intervals bracketed by the two numbers. Values are in milliseconds; maximum value is 32767.

The DATA LEN field is used to specify the length of the data to be accessed on each I/O operation. If there is one number in the DATA LEN field, then fixed length buffers will be accessed. If there are two numbers, separated by a '-', then the data length will vary between the two numbers. For discs, values are in words; for all other devices values are in bytes. Maximum values are listed in section 2.

> For some devices, you will see an 'S' following the second number. In this case, the data length will vary sequentially between the two numbers, incrementing by one until it reaches the high end, and then decrementing by 1

until it reaches the low end. If an 'R' appears, then the data length will vary randomly.

For discs, each word of data in the output buffer is the same, with the left byte containing a random ASCII digit, and the right byte containing a random letter. For all other devices, the data buffer contains the bytes 'ABCDEFGHIJ" preceded by enough blanks to fill out the length.

If the device is a terminal, then the next field in the table contains the baud rate at which the terminal will be accessed. If the baud rate listed in the table does not agree with the baud rate that the corresponding terminal is set to, you will see garbage or nothing on the terminal screen.

If the device is a disc, then there are three more fields:

The first field contains two numbers, separated by a '-'. These two numbers specify the low end and high end sector addresses to be accessed. Sector addresses are logical sector addresses and range from 0 to n, where n is listed below for each disc.

7902 (single-sided)	n=2249
7902 (double-sided)	n=4499
7910	n=47039
7906	n=76799
7920	n=195599

These values do not include spare tracks. The program will allow you to specify values greater than n for any disc, but you will get an error when the exerciser attempts to access a non-existent sector. Note that when you put a different flexible disc into the 7902 drive, it is up to you to change the address field if necessary.

The second field specifies an address increment. If the field contains the word RANDOM, then addresses are chosen randomly in the range specified. If the increment is non-zero, then addresses will increment sequentially by this amount up to the high end, and then restart at the low end. Maximum value is 32767.

The third field can contain the word 'READ', 'WRITE', or 'R/W', specifying whether to do input, output, or both to the device. If R/W is specified, the device is first written to, the same location is read back into a second buffer, and the two buffers are compared to see if they are equal. If 'WRITE' is specified, you are asked to mount a scratch volume; press ENTER when you are ready to continue.

WARNING

Selecting WRITE or R/W will destroy the customer's data base. READ is safe to use.

You may modify the parameters of the I/O exercise by changing any of these fields. The next steps describe how to do this.

3.3 Modifying Test Parameters

Below the table you will see the question 'Do you want to change any parameters?' followed by the cursor. If you are satisfied with the default parameters, answer 'N' and press ENTER. Go to step 6. If you want to change some of the parameters, or delete a device completely, answer 'Y' and press ENTER.

(4) If you answered 'Y' to the question, then the phrase 'Enter chan/device/unit #:' will appear on the IDS. Enter the channel and device number (and unit number, if applicable), separated by commas, of the device you want to change (from the first columns of the table).

The line from the table corresponding to the device will be rewritten on the IDS. Below that, you will see the cursor positioned in the fifth field.

(5a) If you wish to delete the device from the exercise, type 'X' and press ENTER. Then go back to step 4. This step will also reactivate a device which has been deleted.

(5b) To modify an I/O parameter, move the cursor to the desired field (you can use the TAB key), and type the new value(s). If you make a mistake, just use the back space key or the cursor positioning keys to back up. Press ENTER when you are satisfied with your changes.

The line will be rewritten, with the cursor in the last column. If the data appears as you desired, then press the ENTER key. If you want to make more modifications, press any other key, then ENTER. The cursor will reappear in the fifth field, so that you can make further modifications. When everything looks OK, press ENTER.

When you finish editing a device's parameters, the phrase 'Enter chan/device/unit #:' will reappear. If you wish to modify another device, then repeat steps 4 and 5. If not, or if you want to see the entire table again, enter a '0' (zero). The table will be rewritten with your changes incorporated, and the question asking for changes will again appear. If you want to make more changes, answer 'Y' and go back to step 4. If not,

answer 'N'.

(6) When you answer 'N' to the question, the program will ask you some questions concerning program control. If there is a printer on the system, you will be asked 'Do you want output on the printer?' If you answer 'Y', then the device table will be written out on the printer, and error messages and data concerning accesses will also be echoed there. This output does not affect the performance of the I/O exerciser. If there are two printers on the system, the printer with the highest channel/device number is the one written to. The default answer to the question is 'Y'.

The last question asked is 'Do you want to stop on an error?'. If you answer 'Y' to this question, the program will pause after reporting an error. If you answer 'N', then, when an error occurs, the error will be reported, data concerning accesses will be dumped to the IDS and printer, and the program will restart. This mode is useful for overnight operation. You can set up the program, specify output to the printer, answer 'N' to this question, and leave. When you come back you will have a record of any errors that occurred. The default answer is 'Y'.

3.4 Program Operation

After the last question, the program begins doing the specified I/O operations to the devices. You will see output on the IDS, and hear the discs being accessed. You can interrupt the program at this time by hitting the ATTN key (Step 7, below).

As the program is doing the I/O, it logs information about its accesses. For each device, it logs the length of the data buffer written, the address written to, and the wall-clock time it took to do the access. This information is stored in a circular buffer with room for 10 entries.

3.5 Interrupting the Program

(7) When you hit ATTN, the question 'Do you want to see the access data for any device?' will appear on the IDS. If you reply 'Y' to this question, then you will be prompted for a channel and device number. Enter the numbers for the device you desire information about. The information will then be written out, and you will be reprompted. Enter another device number, or '0' (zero) when you are ready to go on.

(8) The question 'Continue or Restart?' will now appear on the IDS. If you want to continue from the present state of the exerciser, answer 'C' and go to step (6). If you want to modify some more parameters, or simply restart with the same parameters, answer 'R' and go to step (4).

Note: If a disc goes offline, and then comes back online (e.g., if you take out the flexible disc and insert a new one), you must answer 'R' (restart) to this question. Otherwise, you will get back an error code of !3800 each time the exerciser tries to access that disc. Restarting the exerciser will clear this error.

4.0 ERROR INTERPRETATION

There are two kinds of errors reported by the program: I/O errors and Exerciser program errors.

I/O errors are reported directly on the IDS (and on the printer, if it is configured). Information reported includes:

a. the channel and device number of the device that reported the error

b. the error code (see Section 5 for an explanation of error codes)

c. if the I/O access was a read or write, the program will display the length of the data buffer, and, if requested, its contents. The data is displayed both in hex and ASCII. If the device is a disc, the sector address that was accessed will also be reported.

d. the other parameters to the I/O request which include the request code (REQ'CODE; 1=READ, 2=WRITE, 3=WRITE'READ), other values depend on the device); the request priority (PRIORITY; should always be 100); and two control parameters (CONTROL1 and CONTROL2; these are device dependent)

e. the transmission log, which is the number of words or bytes actually transferred on a read or write

f. the TCB# of the task controlling the device, and ECB# of the I/O request, which may be useful for debugging.

g. the phrase 'END OF MESSAGE'

After an error is reported, the program continues as described in step 7 above.

Exerciser program errors are reported in three ways:

(1) Crash message. Ignore the parameters and reload the program. If it crashes again, there is something drastically wrong. Check your configuration; if nothing is amiss, call your System Specialist.

(2) RUN light blinks (heartbeat). Check the NIR lights (on BUS INTFC board) for an error code. Current errors:

3 - couldn't initialize GIC

You can try to continue by pressing RUN, but you probably won't get very far. Try reloading the program and checking your configuration. If that doesn't help, call your System Specialist.

(3) Call to system debug (SYSDEBUG). You can try to continue by typing "EX" and pressing the ENTER key. The program will probably do funny things though. Try reloading the program and checking your configuration.

5.0 ERROR CODES

The error code reported by the program is the one returned by the physical device driver, and is dependent upon device type.

To get further information about any errors, you will have to look at the software listings or examine the HP-IB transactions.

5.1 IDS Error Codes

!0011 - channel program error
!0021 - channel input overrun
!0081 - self-test error

5.2 Disc Error Codes

!0104 - illegal opcode
!0201 - invalid parameter
!0204 - unit available
!0704 - cylinder compare error
!0804 - uncorrectable data error
!0904 - head/sector compare error
!0A04 - I/O program error
!0C04 - end of cylinder
!0E05 - data overrun
!0F04 - possibly correctable error
!1004 - illegal space track
!1104 - defective track
!1204 - head motion detected
!1304 - status 2 error
!1604 - protected track
!1704 - unit unavailable
!1F04 - drive attention
!2006 - busy (7910=offtrack)
!2100 - offline
!2107 - powerfail
!2200 - online
!2300 - power off
!2406 - drive fault
!2506 - HP-IB error
!2606 - device was cleared
!2704 - FIFO abort
!2806 - serial poll error
!2904 - illegal CSRQ
!2A04 - protect switch on
!2B02 - timeout - IMB
!2C02 - timeout - HP-IB
!2D06 - data chain error
!2E06 - bad channel instruction
!2F06 - SIOP failure
!3006 - address rollover (non-DMA)
!3106 - HP-IB parity error
!3202 - watchdog timer

```
!3404 - seek check
!3704 - DSJ error
!3800 - blocked driver
!3906 - DMA address rollover
!3A06 - DMA non-reponding memory module
```

5.3 Terminal Error Codes (ADCC)

```
!0007 - driver busy
!000B - request not cancelled
!0103 - invalid parameter
!xx09 - request cancelled
!xx11 - parity error
!xx21 - overrun error
!xx41 - channel program abort
!8x01 - TX line bad -- disconnected
!9x01 - TX line bad -- disconnected

x - don't care
```

5.4 Printer Error Codes

```
!000B - volume monitor online status
!0201 - illegal parameter
!0804 - paper out
!0904 - offline
!0A04 - not ready
!0B04 - HP-IB command parity error
!0C00 - power on (reset)
!1102 - watchdog timer
!1204 - SIOP failure
!2506 - HP-IB error
!2606 - device was cleared
!2704 - FIFO abort
!2806 - serial poll error
!2904 - illegal CSRQ
!2A04 - protect switch on
!2B02 - timeout - IMB
!2C02 - timeout - HP-IB
!2D06 - data chain error
!2E06 - bad channel instruction
!2F06 - non-responding device
!3006 - address rollover (non-DMA)
!3106 - HP-IB parity
!3906 - DMA address rollover
!3A06 - DMA non-responding memory module
```

-hp-

SECTION 447

HP 300 FILE MANAGEMENT
STAND ALONE UTILITIES
EXTERNAL REFERENCE SPECIFICATION

NOTICE

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

Hewlett-Packard Company
19447 Pruneridge Ave., Cupertino, California 95014 U.S.A.

Printed in U.S.A.

SECTION 447B

HP 300 FILE MANAGEMENT

STAND ALONE DISC VOLUME UTILITY

(FMDISCUT)

EXTERNAL REFERENCE SPECIFICATION

(Program Revision 01.XX)

NOTICE

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

Hewlett-Packard Company
19447 Pruneridge Ave., Cupertino, California 95014 U.S.A.

Printed in U.S.A.

TABLE OF CONTENTS

1.0 INTRODUCTION

1.1 Product Abstract

This document details the external reference specifications for FMDISCUT, one of the File Management standalone utilities on the HP 300 system.

1.2 Overview

File Management provides the user several basic utilities which, because of the functions they perform, must be executed in the stand alone environment. These utilities are presented to the user as options within the FMUTILIT (File Management Utilities), FMDISCUT (File Management Volume Utilites), and FMPATCHV (File Management Patch Volume) programs which are provided to the user on the cold loadable Diagnostic and Utility Package (DUP) flexible discs. The intended uses of each program are listed below:

UTILITY	USES
FMDISCUT:	detect and spare bad tracks indicate the condition of a surface prepare a surface for use [covered in this Handbook Section]
FMUTILIT:	copy non-standard volumes and cartridges back up systems Store or Restore information replace the primary with the back-up operating system [covered in Handbook Section 447C]
FMPATCHV: (DUPCE only)	inspect or modify the the contents of a volume output the contents of a volume in hex, octal or decimal [covered in Handbook Section 447D]

The details of how to load a DUP flexible disc and the environment it initially provides the user are described in the DUS handbook section (440). This document describes only the FMDISCUT programs.

1.3 Required Hardware

The HP 300 configuration sufficient to run DUS is required. This includes 256 Kbytes of memory, a GIC, CPU, 7902 and an IDS or ADCC with a terminal as a console.

1.4 Required Software

The Diagnostic and Utility Package containing FMDISCUT (DUP or DUPCE).

2.0 OPERATION

Once the DUP flexible disc has been successfully cold loaded and the prompt character ":" is present the user must enter 'FMDISCUT' to enter the File Management Volume Utilities program.

The FMDISCUT program is made up of all of the subprograms listed in the menu below. The program initially outputs:

```
 Welcome to the Volume Utility Program Revision xx.xx

 Please select one of the following options:

    Analyze Volume          AV
  * Assign Spares           AS
    Exit                    EX
  * Format Volume           FV
    List Track Status       LS
  * Prepare Volume          PV

  *Please consult with your SE before using these functions

 Enter option:
```

The user must now enter the two-character code of the desired option. A response of 'EX' terminates the program and returns to the environment that was initially cold loaded. If a code is entered that is not in the menu list then the message:

```
  Illegal option selected
```

is output and the user is reprompted for the option. Once a legal option, other than 'EX', has been entered the function is invoked. Upon completion, successful or unsuccessful, of the function the previously detailed menu is output again and the user is prompted to enter the next option.

Any I/O errors that are encountered during the execution of a function are communicated to the user through the messages detailed in Section 3.0.

2.1 Analyze Volume

This function is invoked whenever the 'AV' function is selected. Analyze Volume reads the data field of each sector in each track using the read verify command. Read verify computes the CRC word and compares it with the existing CRC word to determine if there were any data errors. Tighter timing constraints and the fact that READ VERIFY does not transmit data to memory distinguishes READ VERIFY from READ. If a defective track is found its location will be output on the selected peripheral (i.e., the IDS, a terminal used as a console, or Line Printer). If track number zero, head number zero is found to be defective then the volume

is defective and must not be used.

The function initially outputs the message:

    ANALYZE VOLUME function now in operation

  followed by:

    Enter volume's channel number:

  Valid Response:

    Expecting 1 to 15

  Inquiry:

    Enter volume's device number:

  Valid Response:

    Expecting 0 to 7

  Inquiry:

    Enter volume's unit number:

  Valid Response:

    Expecting 0 to 7

Once a valid channel, device, and unit have been specified the following interaction occurs if the volume to be analyzed must be mounted on a 7902.

  Inquiry:

    Will you please mount the volume?

  Valid Response:

    Expecting YES or NO

A response of 'NO' terminates the function. A response of 'YES' should be entered only after the volume has been mounted. If the volume is a flexible disc and it is not properly formatted the message:

    Flexible Disc needs to be RE-FORMATTED

is output and the menu is again presented. Regardless of the media to be analyzed the system will then output the message:

The option you have selected will READ/VERIFY all data on the chosen <dev name> <vol type> volume.

<dev name> will be '7902', '7910', '7906' etc;
<vol type> will be one of the following volume types:

STANDARD
NON-STANDARD
SYSSTORE
BACK-UP
DUS
INTERCHANGE FORMAT

The following menu is then output:

Clear print echo	CPE
Execute option	EXC
Exit	EXI
Set printer echo	SPE
Show commands	SHW

followed by:

Enter command:

If an illegal command is entered the message:

Illegal command entered

is output followed by a prompt for another command. The EXI command exits the function and results in the user being prompted for another FMDISCUT function. The EXC command results in the function being executed. After it completes the user is prompted for another FMDISCUT function. The SHW command outputs the above menu. The CPE, and SPE commands result in the output of the above menu upon their completion.

If the channel, device, and unit numbers specify a 7906 then the user is given the option of analyzing the upper platter, the lower platter, or both platters through the addition of the following command to the menu:

Select platter(s) to analyze SPA (UPPER/LOWER/ALL)

If EXC is selected before SPA when analyzing a 7906 the message:

Platter(s) must be selected with SPA command

is output and the user is prompted for another command. "UPPER" specifies that only the upper platter is to be analyzed, "LOWER" specifies that only the lower platter is to be analyzed, and "ALL" specifies that all platters are to be analyzed. If illegal values are entered for SPA then the message:

Illegal value(s) entered

is output and the user is prompted for another command.

SPE enables echoing of any output, resulting from the execution of the function, to a printer and CPE disables the echoing of any output. Echoing is initially disabled. The SPE command interacts with the user to obtain the channel and device number of the printer as follows:

Inquiry:

Enter printer channel number:

Valid Response:

Expecting 1 to 15

Inquiry:

Enter printer's device number:

Valid Response:

Expecting 0 to 7

If the device does not identify itself as a 2631A or a 2608A the message:

Device is not a printer.

is output.

Once the channel and device number of a printer has been input printer echoing is enabled. When the EXC command is entered the volume is analyzed.

For each defective track which has not had an alternate track assigned to it a specific I/O error message (detailed in Section 3.0) will be output along with the message:

Defective track ttt, head h

where 'ttt' and 'h' are the track and head numbers involved. If the volume is found to be defective (i.e., 0/0/0 has been found to be bad in some way or there are more defective tracks than spare tracks available) the message:

The volume on channel 'c', device 'd', unit 'u' is defective.

is output. Upon completion the message:

ANALYZE VOLUME completed successfully

is output and the FMDISCUT menu is presented.

2.2 Assign Spares

This function is invoked whenever the option 'AS' is selected and results in the assignment of an alternate track (if one exists) to the specified defective track. A track located in the spare area can also be spared, however, neither the defective nor the spare track will be usable by the system. (flexible disc defective tracks are handled differently and will be addressed separately in a later paragraph).

If a 7906, 7920, or 7925 disc drive indicates to the disc driver (software internally used by the File Management Utilites) that a possibly correctable data error has been read, then the disc driver will attempt to correct the error. If it is successful, the data will be written to the spare track without any error indication being made to the Assign Spares function. If the error cannot be corrected a READ FULL SECTOR command is issued by AS. This will effectively bypass the error checking mechanism of the disc and write the data out to the spare track. Since the error did occur, this data may be corrupt so use it with caution!!! If the READ FULL SECTOR cannot complete successfully, the track will be spared but the data will not be written out to the spare track.

If the data cannot be read from the track to be spared without an error, the volume's label (if it is labelled) will be altered. This will prevent the volume from being mistakenly used as initially intended (example: as a good, cold loadable labelled volume).

The function initially outputs the message:

```
  ASSIGN SPARE TRACK function now in operation
```

and then interacts with the user in the identical manner detailed in Analyze Volume (Paragraph 2.1) to obtain the channel, device, and unit number of the volume, and get the volume mounted. If the volume is a flexible disc and it is not properly formatted the message:

```
  Flexible Disc needs to be RE-FORMATTED
```

is output and the menu is again presented. Once the system recognizes the device and type of volume which is to be spared the following message is output:

```
   Track(s) will be marked defective on the chosen
   <dev name> <volume type> volume.

   Do you wish to continue?
```

If NO is answered the FMDISCUT menu is presented, otherwise the function continues by listing the menu for AS.

The menu for Assign Spares will contain the following command in addition to those already discussed in Analyze Volume.

Enter DEFECTIVE track address DTA <Cyl no.>,<Head no.>

When the DTA command is selected the user is specifying with <Cyl no.> and <Head no.> the track which is to be marked defective.

When the EXC command is entered and a volume attached to a 13037 is having a track spared the message:

Set protect switch OFF and format switch ON

is output and then the following interaction occurs:

Inquiry:

Is disc drive ready?

Valid Response:

Expecting YES or NO

A response of 'NO' terminates the function. A response of 'YES' should be made only after the protect and format switches have been set as requested. Once the track to be spared has been identified and the EXC command has been entered the track is read. If no errors occur during the read the data will then be written out to the spare track. If errors do occur the process described above will be initiated. Whenever the disc driver has been notified that an error has occurred the message:

Data lost from sector ss, track ttt

will be output.

If the volume was a standard volume (ie. a volume whose label was created by the CREATE LABEL command) or an Amigo "labelled volume" (such as a SYSSTORE volume created by the STORE VOLUME command) data recovery will be attempted in the same way as the unlabelled volume above. However, if a read error does occur the label will be marked to indicate a bad volume. Along with the above message the following message will be output:

Since data was lost on the chosen <dev name> <volume type> volume please consult with your SE before an attempt is made to use this volume.

If the volume is a 7906 with a nonmatching label and an error occurs during the data read the message:

Volume label mismatch. Label is <volume name>

Do you want the cartridge/platter to be used as a NON-STANDARD volume?

will be output where <volume name> is the label (or ... if no label exists) on the upper or lower platter. If YES is answered the track will be spared, the data will be lost and the upper label will be modified if the track getting spared is on surface 0 or 1, or the lower label will be modified if surface 2 or 3 has the track to be spared. If NO is answered the track will not be spared. Instead the FMDISCUT menu will be presented.

When AS is used with flexible discs, tracks are not assigned alternates. Instead, they are made invisible so that they can't be written to. No data recovery at all is attempted when flexible disc tracks are spared. Also instead of the track being spared after the DTA and EXC commands are invoked, the sparing doesn't occur until after the EXI command is entered. Only when EXI is used with flexible disc does it also cause the entire media to be formatted (thus destroying all existing data)! This reformatting is necessary because of the way AS changes the logical/physical track relationship on a flexible disc. (See Service Handbook Section 860 for more details.)

WARNING: the address of all bad tracks to be spared must be input on the same pass of AS (i.e., before EXI is entered).

If illegal values are entered for (cyl no.) or (head no.) the message:

Illegal value(s) entered

is output and the user is prompted for another command. If EXC is selected before DTA(i.e., before a defective track is defined) the message:

Defective track must be selected with DTA command

is output and the user is prompted for another command. The message:

Defective track ttt, head h, assigned alternate track ttt, head h

is output for each defective track that is assigned an alternate (no message is output for a flexible disc).

The input of a cylinder and head number of zero will result in the output of the message:

A defective track on cylinder 0, head 0 cannot be assigned a spare

The volume on channel 'c', device 'd', unit 'u' is defective

For media mounted on a 13037 type drive the message:

Turn format switch OFF. Disc ready? (YES/NO)

is output. The message:

The specified track is already marked defective

is output if the specified track already has an alternate assigned to it. The message:

An assignable spare track was not found

is output when no spare track can be found to assign as an alternate. Upon completion the message:

ASSIGN SPARES successfully completed

is output.

2.3 Exit

The selection of the 'EX' option results in the FMDISCUT program being terminated and control returned to DUS.

2.4 Format Volume

This function is invoked whenever the option 'FV' is selected. It results in the initialization of the preamble of and the data field in every sector of the volume (or specific surface) which is being formatted. When a volume is formatted the physical address of each sector within each track being formatted is rewritten in that sectors preamble. Also the SPD bits (which are the bits in the preamble that indicate, when set to one, that the track is a spare(S), protected(P) or defective(D)) are set to zero thus requiring all tracks previously spared to be spared again after the formatting has completed.

Note: this function ignores data errors and thus will not report any bad tracks encountered. If a surface needs analyzing use Prepare or Analyze Volume functions.

The function initially outputs the message:

FORMAT VOLUME function now in operation

and then interact with the user in the identical manner detailed in Analyze Volume (Paragraph 2.1) to obtain the channel, device and unit number of the volume, get the volume mounted, warn of the destruction of data on the volume and to get the protect and

format switches set (only when 13037 drives have been identified).

If the channel, device, and unit numbers specify a 7906 then the user is given the option of formatting the upper platter, the lower platter, or both platters through the addition of the following command to the menu described in Section 2.1:

```
  Select platter(s) to format   SPF   (UPPER/LOWER/ALL)
```

If EXC is selected before SPF when formatting a 7906 the message:

```
  Platter(s) must be selected with the SPF command
```

is output and the user is prompted for another command. "UPPER" specifies that only the upper platter is to be formatted, "LOWER" specifies that only the lower platter is to be formatted, and "ALL" specifies that all platters are to be formatted. Currently, however, an error message will be output if either the upper or lower protect switches are set, regardless of the platter selected. Therefore make sure that both protect switches are off. If illegal values are entered for SPF the message:

```
  Illegal value(s) entered
```

is output and the user is prompted for another command. When the EXC command is entered the volume is formatted.

If the volume being formatted is a flexible disc that is unformatted or there are less than 74 usable cylinders the function will output the message:

```
  The UNFORMATTED volume on device <dev name> will be FORMATTED.
```

after EXC has been entered. The above message will also be output if the label in sector zero cannot be read.
Upon completion of the formatting
the message:

```
  FORMAT VOLUME successfully completed
```

is output.

2.5 List Track Status

This function is invoked whenever the option 'LS' is selected. LS determines the status of all tracks on a volume by reading the preamble of each sector within the track. It then outputs the status of each track if the track is in an abnormal state (such as spared or defective in some way). The function initially outputs the message:

LIST TRACK STATUS function now in operation

and then interacts with the user in the identical manner detailed in Analyze Volume (Paragraph 2.1) to obtain the channel, device and unit number of the volume and get the volume mounted. If the channel, device, and unit numbers specify a 7906 then the user is given the option of listing the track status of the upper platter, the lower platter, or both platters through the addition of the following command to the menu described in Section 2.1:

List trk status of platter(s) LSP (UPPER/LOWER/ALL)

If EXC is selected before LSP when listing track status on a 7906 the message:

Platter(s) must be selected with the LSP command

is output and the user is prompted for another command. "UPPER" specifies that only the track status of the upper platter is to be listed, "LOWER" specifies that only the track status of the lower platter is to be listed, and "ALL" specifies that the track status of all platters is to be listed. If illegal values are entered for LSP then the message:

Illegal value(s) entered

is output and the user prompted for another command. The selection of the EXC command results in the outputting of one or more messages which describe the status of the volumes tracks. If the volume is a flexible disc and it is not properly formatted the message:

Flexible Disc needs to be RE-FORMATTED

is output and the menu is again presented.

If the volume is a flexible disc then the message:

The volume on channel 'c', device 'd', unit 'u' does not have any defective tracks

is output if no tracks are flagged defective.

If defective tracks are found then the message:

The volume on channel 'c', device 'd', unit 'u' has 'n' defective tracks

(where n is the number of defective tracks) is output.

If the volume is not a flexible disc then the message:

Defective track 'ttt', head 'h', assigned alternate track 'ttt', head 'h'

is output for each defective track found that has been assigned an alternate track and the message:

Defective track 'ttt', head 'h', no alternate track assigned

is output for each defective track found that has not been assigned an alternate. The HP 300 system requires all defective tracks to be assigned spares or for flexible disc made invisible. AS can be executed to perform those tasks. The message:

List track status completed successfully

is output upon successful completion.

2.6 Prepare Volume

This function is invoked whenever the option 'PV' is selected. It prepares a volume in one step by automatically calling List Track Status, Format Volume and a volume verification function which spares any track it determines to be defective (except c/h/s 0/0/0). It will also spare those tracks which were listed as defective by the List Track Status pass. The function initially outputs the message:

PREPARE VOLUME function now in operation

and then interacts with the user in the identical manner detailed in Analyze Volume (Paragraph 2.1) to obtain the channel, device and unit number of the volume, get the volume mounted, warn that all data on the volume will be lost and to get the protect and format switches set (only when a 13037 drive has been identified).

If the channel, device, and unit numbers specify a 7906, then the user is given the option of preparing the upper platter, the lower platter, or both platters through the addition of the following command to the menu described in Section 2.1:

Select platter(s) to prepare SPP (UPPER/LOWER/ALL)

If EXC is selected before SPP when preparing a volume on a 7906, the message:

Platter(s) must be selected with the SPP command

is output and the user is prompted for another command. "UPPER" specifies that only the upper platter is to be prepared, "LOWER" specifies that only the lower platter is to be prepared, and "ALL" specifies that both platters are to be prepared. Currently, however, an error message will be output if either the upper or lower protect switches are set, regardless of the platter selected. Therefore make sure that both protect switches

are off. If illegal values are entered for SPP, the message:

Illegal value(s) entered

is output and the user is prompted for another command. The selection of the EXC command results in the outputting of the message:

List Track Status pass started

and the outputting of one or more messages that describe the status of the volume's tracks as discussed under the section List Track Status. If the volume being prepared is found to have defective or defective and spared tracks, one of the two following additional messages will be output. When preparing a flexible disc, the message:

The above track(s) will be kept invisible

is output. When preparing volumes other than a flexible disc, the message:

The above track(s) will be marked defective and spared

is output after the message indicating the defective track address.

If a bad preamble is encountered the above message will be followed by the messages;

EXCEPT TRACK(s) with bad pre-ambles.

Preambles are RE-WRITTEN during the FORMAT pass

Upon completion of List Track Status, Format Volume is invoked and the message:

Format pass started

is output. If the volume has an improper HP format the message:

All DEFECTIVE tracks, if any, were lost

is output. DEFECTIVE will be replaced by INVISIBLE in the above message if the volume is a flexible disc.

If the volume being prepared is a flexible disc that is unformatted or there are less than 74 usable cylinders the function will output the message:

The UNFORMATTED volume on device <dev name> will be PREPARED.

The above message will also be output if the label in sector zero cannot be read. After Format Volume completes successfully,

the message:

  Volume verification pass started

is output. At this time a non-exhaustive analysis of the data fields of each sector on the disc volume is done. A spare track is assigned to each track it determines to be defective. If the volume is a flexible disc, three read-verify passes are performed. Following this, the message:

  The bad track(s) found previously will now be marked defective and spared

is output and all defective tracks as well as those listed during the LS phase are marked defective and spared (bad tracks on flexible disc are made "invisible"). Upon completion of the volume verification, the message:

  If additional tracks are to be marked defective, please use the AS option

is output to inform the user to use Assign Spares to spare additional tracks not found by PV that are suspected to be bad. At this point, Prepare Volume has successfully completed and the message:

  Prepare Volume completed successfully

is output.

In the Copy Volume function only a retry after a write/verify is attempted. If an error is encountered during the first write/verify pass the function will output the location of the problem and automatically retry. A verify failure during the second write/verify pass will stop CV before it completes.

Figure 2.1 - Restore Volume Drive Combinations

A date and time should be included in the ID label because if the label typed in is identical to the previous store volume label the disc would be rejected. This will prevent the current store operation from accidently overwriting a valid store disc.

3.0 ERROR INTERPRETATION

If any I/O errors are encountered during the execution of any of the File Management Utility or Patch Volume functions the user will be informed of the error through one of the messages detailed in this section.

The error messages are divided into correctable errors and uncorrectable errors. For a correctable error the user will be informed of the problem and asked to respond 'YES' to a prompt when the condition has been corrected. If the user responds 'NO' then the function is terminated and the user is prompted to select another function.

For an uncorrectable error the user is informed of the problem and the function is automatically terminated and the user is prompted to select another function. The messages are as follows:

Correctable Errors

```
Device on chan/dev/unit is not ready
Device on chan/dev/unit is write protected
Flexible disc mounted is not in HP format
Turn format switch on - chan/dev/unit
```

Uncorrectable Errors

```
Drive attention on chan/dev/unit
I/O failed after 'n' retries
Device is not a disc
Device is not a printer
Channel type is not GIC
Channel is not present
I/O error on chan/dev/unit
Watchdog timer or parameter error
```

followed by:

```
Cannot report status
Cylinder 'c', head 'h', sector 's'
Relative sector number 'r'
```

and then:

```
Channel status      Status1        Status2
   (channel          (status        (status
     status)          1 word)        2 word)
```

"chan/dev/unit" will be the decimal numbers for the channel, device, and unit. 'n' will be the decimal number of retries. 'c', 'h', 's', and 'r' are, respectively, the decimal numbers for

the cylinder, head, sector, and relative sector addressed. All COPY, STORE, and RESTORE volume commands read/verify every data transfer.

Whenever the system encounters a Disc I/O error the appropriate message listed above will be followed by a message of the following form:

```
Disc I/O error xxx-- (text message) on device (dev name)
,channel 'c', device 'd', unit 'u' at relative sector z
(cylinder 'c', head 'h', unit 'u').
```

The error number xxx and (text message) which would be used are the same as those output by Amigo/300. The error numbers and messages are listed in Handbook Section 199 -- Error and Status Codes.

The message:

```
Verification of data written has failed.
```

is output by the system if a read verify of the lower platter of a 7906 has failed (a read verify is done after every write) . If the read verify fails after a write to a removable cartridge the above message will be followed by the message:

```
Error detected on upper platter. Please remove.
```

The function will continue after the cartridge has been replaced.

If an error occurs and the printer echo is not enabled, the message:

```
Is the above message recorded?
```

is output. The user may wish to record the message and then type 'Y' or 'N' ('YES' or 'NO'). If the above is not typed, the message:

```
Expecting YES or NO
```

is output.

Following the output of any of the correctable errors the user will be prompted with:

```
Inquiry:

   Drive ready?

Valid Response:

   Expecting YES or NO
```

The user should respond 'YES' only after correcting the problem. A response of 'NO' results in the function being terminated and the user prompted to select another File Management Utility or Patch Volume function.

-hp-

SECTION 447C

HP 300 FILE MANAGEMENT

STAND ALONE FILE MANAGEMENT UTILITY

(FMUTILIT)

EXTERNAL REFERENCE SPECIFICATION

(Program Revision 01.XX)

NOTICE

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

Hewlett-Packard Company
19447 Pruneridge Ave., Cupertino, California 95014 U.S.A.

Printed in U.S.A.

TABLE OF CONTENTS

1.0 INTRODUCTION

1.1 Product Abstract

This document details the external reference specifications for FMUTILIT, one of the File Management standalone utilities on the HP 300 system.

1.2 Overview

File Management provides the user several basic utilities which, because of the functions they perform, must be executed in the stand alone environment. These utilities are presented to the user as options within the FMUTILIT (File Management Utilities), FMDISCUT (File Management Volume Utilites), and FMPATCHV (File Management Patch Volume) programs which are provided to the user on the cold loadable Diagnostic and Utility Package (DUP) flexible discs. The intended uses of each program are listed below:

UTILITY	USES
FMDISCUT:	detect and spare bad tracks indicate the condition of a surface prepare a surface for use [covered in Handbook Section 447B]
FMUTILIT:	copy non-standard volumes and cartridges back up systems Store or Restore information replace the primary with the back-up operating system [covered in this Handbook Section]
FMPATCHV: (DUPCE only)	inspect or modify the the contents of a volume output the contents of a volume in hex, octal or decimal [covered in Handbook Section 447D]

The details of how to load a DUP flexible disc and the environment it initially provides the user are described in the DUS handbook section (440). This document describes only the FMUTILIT program.

1.3 Required Hardware

The HP 300 configuration sufficient to run DUS is required. This includes 256 Kbytes of memory, a GIC, CPU, 7902 and an IDS or ADCC with a terminal as a console.

1.4 Required Software

The Diagnostic and Utility Package containing FMUTILIT (DUP or DUPCE).

2.0 OPERATION

Once the DUP flexible disc has been successfully cold loaded and the prompt character ":" is present the user must enter FMUTILIT to enter the File Management Utilities program.

The FMUTILIT program provides those functions listed in the menu below.

```
 Welcome to the File Utility Program Revision xx.xx

 Please select one of the following options:

  * Copy 7906 Cartridge       CC
    Clear printer echo        CE
    Copy volume               CV
    Enable back-up system     EB
    Exit                      EX
  * Restore lower             RL
    Restore volume            RV
    Set printer echo          SE
  * Store lower               SL
    Store volume              SV

  *Please consult with your SE before using these functions

 Enter option:
```

The user must now enter the two-character code of the desired option. A response of 'EX' terminates the program and returns to the environment that was initially cold loaded. If a code is entered that is not in the menu list then the message:

```
  Illegal option selected
```

is output and the user is reprompted for the option. Once a legal option, other than 'EX', has been entered the function is invoked. Upon completion, successful or unsuccessful, of the function the previously detailed menu is output again and the user is prompted to enter the next option.

Whenever data must be copied from the fixed platter to the disc cartridge (or vice versa) on a 7906 one of the following messages will be output:

```
   Function pass number (n) started
```

Where n = 0 means copy original lower to upper backup cartridge
1 copy original upper to lower platter
2 copy stored original upper to cartridge
3 copy stored original upper to lower platter
4 copy lower platter data to removable cartridge
5 copy stored original lower to lower platter
6 copy original lower to upper platter
7 copy original lower stored on cartridge to lower platter

One or more of these messages will be output whenever CC, RL, RV, SL, SV is run. These step numbers will make it easier to determine exactl what step is being performed by the program, therefore, if an error should occur during a copy use the pass number to find out what was being done so that the appropriate corrective action can be taken.

Any I/O errors that are encountered during the execution of a function are communicated to the user through the messages detailed in Section 2.0.

2.1 Copy 7906 Cartridge *****THIS FUNCTION SHOULD BE USED IN EMERGENCY SITUATIONS ONLY*****

This function is invoked whenever the option 'CC' is selected. 'CC' makes copies of one or more 7906 disc cartridges.

WARNING: because the information to be copied must be written down to the lower platter ALL ORIGINAL LOWER PLATTER DATA WILL BE LOST!

The function initially outputs the message:

COPY disc CARTRIDGE function now in operation

and then interacts with the user in the identical manner detailed in Analyze Volume (Paragraph 2.1 of Handbook Section 447B) to obtain the channel, device, and unit number of the 7906 on which the copies are to be made. If the channel, device, and unit numbers do not specify a 7906 then the message:

Device is not a 7906

is output and the function is terminated. If the channel, device, and unit numbers specify a 7906 then the following interaction occurs:

Inquiry:

Will you please mount the disc cartridge to be copied?

Valid Response:

Expecting YES or NO

A response of 'NO' outputs the message:

COPY disc CARTRIDGE terminated by user

and then terminates the function. A response of 'YES' should be entered only after the disc cartridge has been mounted. The message:

Function pass No. 1 started

is output by the system when 'YES' has been entered. Once the upper platter has been successfully stored on the lower platter the message:

Upper platter successfully copied to lower platter

is output and then the following interaction occurs:

Inquiry:

Will you please mount the disc cartridge on which the upper platter is to be stored?

Valid Response:

Expecting YES or NO

A response of 'NO' results in the message which requests you to mount the disc cartridge that you want to make a copy of, being reissued. A response of 'YES' should be made only after the disc cartridge has been mounted. Upon entering "YES" the system will output the following message:

Function pass number 4 started.

Once the lower platter, which now contains the data to be copied, has been successfully stored on the upper platter the message:

Lower platter successfully stored

is output and the message which asks you to mount a disc cartridge in order to make a copy is reissued. If a "YES" is entered another copy of the lower platter will be made to the new cartridge just inserted. If "NO" is entered the system reissues the question:

Will you please mount the disc cartridge to be copied.

A "YES" will cause the function to copy the upper platter to the lower platter. A "NO" will terminate the function.

If a write error occurs on the upper platter the message:

Error detected on upper platter. Please remove

is output and then the user is prompted, with the previously detailed mount message, to mount another disc cartridge. To use the cartridge just removed the defective track must be found and spared or made invisible. The AS function can be used.

2.2 Clear Printer Echo

This command disables printer echoing when it is invoked.

2.3 Copy Volume

This function is invoked whenever the option 'CV' is selected and results in the duplication of a volume. The subclasses of the volumes must be the same and must reside on separate 7906, 7920 or 7925 drives. The function initially outputs the message:

Welcome to the COPY VOLUME function

followed by:

Enter SOURCE volume information

and then interacts with the user in the identical manner detailed in Analyze Volume (Paragraph 2.1 of Handbook Section 447B) to obtain the channel, device, and unit number of the source volume. It next outputs:

Enter DESTINATION volume information

and then interacts as above to obtain the channel, device, and unit number of the destination volume.

If the subclasses are the same and the units are not the same, the function starts the duplication; otherwise, the message:

Illegal device/unit selected

is output and the function is terminated.

The function will output a single dot on the console for each successful I/O transfer of 255 sectors. Upon completion, the message:

Volume copy completed successfully

is output.

This function can be interrupted by depressing the ATN key or the "control y" (key on the system console). The message:

Do you want to abort the volume copy?

is output. Legal response is either 'YES' or 'NO'. If some other response is entered, then the message:

Expecting YES or NO

is output.

In the Copy Volume function only a retry after a write/verify is attempted. If an error is encountered during the first write/verify pass the function will output the location of the problem and automatically retry. A verify failure during the second write/verify pass will stop CV before it completes.

2.4 Enable Backup System

This function is invoked whenever the option 'EB' is selected and results in making the Backup (Secondary) System the Primary System and the Primary System the Backup (Secondary) System on a SYSTEM volume. The function initially outputs the message:

ENABLE BACKUP SYSTEM now in operation

followed by:

Enter SYSTEM volume information

and then interacts with the user in the identical manner detailed in Analyze Volume (Paragraph 2.1 of Handbook Section 447B) to obtain the channel, device, and unit number of the SYSTEM volume whose Primary and Backup Systems are to be swapped. Once the Systems have been swapped the message:

ENABLE BACKUP SYSTEM successfully completed

is output. If the volume is not a system volume the message:

Volume is not a SYSTEM volume

is output and the function is terminated. If the volume does not have a back-up system, then the message

Volume has no BACK-UP system

is output. After the message,

ENABLE BACK-UP SYSTEM successfully completed

is output the message,

The PRIMARY (or SECONDARY) system is now the CURRENT system

is output.

2.5 Exit

The selection of the 'EX' option results in the FMUTILIT program being terminated and control returned to DUS.

2.6 Restore Lower

This function is invoked whenever the option 'RL' is selected. It restores the lower platter of a 7906 from the disc cartridge on which it was stored with the Store Lower function (Paragraph 2.9).

NOTE: Restore Lower is only to be used for the following:

(1) To recover from a Store Volume (detailed in 2.10) that had stored the lower platter and was then abnormally terminated.

    When this occurs you are left with a disc cartridge that contains the data stored from the lower platter and a disc cartridge that contains the upper platter data that was never successfully stored. The data on these two cartridges are matched but because the lower platter was stored on its disc cartridge with the partial completion of a Store Volume and the disc cartridge that contains the upper platter data was not stored with Store Volume the labels on these two platters will not match.

    THIS IS THE ONLY TIME THAT YOU SHOULD RESPOND 'YES' TO THE PROMPT THAT REQUESTS PERMISSION TO MAKE THE LABELS MATCH.

(2) To restore data to the lower platter of a 7906 from a disc cartridge that was created from the Store Lower function detailed in 2.9.

The function initially outputs the message:

RESTORE LOWER platter function now in operation

and then interacts with the user in the identical manner described in Analyze Volume (Paragraph 2.1 of Handbook Section 447B) to obtain the channel, device, and unit number of the 7906. If the channel, device, and unit numbers do not specify a 7906 then the function is terminated. If the channel, device, and unit numbers specify a 7906 then the following interaction

occurs:

Inquiry:

Will you please mount the disc cartridge on which the lower platter was stored?

Valid Response:

Expecting YES or NO

A response of 'NO' terminates the function. A response of 'YES' should be entered only after the disc cartridge has been mounted.

Function pass number 7 started

is then output by the system as it copies the upper platter to the lower platter. Once the platter has been successfully restored the message:

RESTORE LOWER platter successfully completed

is output and then the following interaction occurs:

Inquiry:

Will you please mount the disc cartridge that contains the upper platter data?

Valid Response:

Expecting YES or NO

A response of 'NO' terminates the function. A response of 'YES' should be entered only after the disc cartridge has been mounted. If you respond 'YES' without mounting another disc cartridge the message:

Volume label mismatch

is output and the previously detailed mount message is reissued. Once the disc cartridge has been mounted the message:

RESTORE VOLUME successfully completed
System can now be cold loaded

is output if the data on the upper and lower platters contain identical labels for a system volume. If the labels do not reflect a system volume then the message:

Volume does not contain a system

is output. If the upper and lower platters contain labels for a system volume which are not identical then the following interaction occurs:

Inquiry:

Labels do not match. Do you want them matched?

Valid Response:

Expecting YES or NO

A response of 'NO' terminates the function with the 7906 containing data that CANNOT be cold loaded as the data on the upper and lower platters will not match. A response of 'YES' makes the labels identical and then outputs the message:

RESTORE volume successfully completed
System can now be cold loaded

NOTE: A response of YES does not guarrantee that the system will work. If the labels do not match it is usually a syptom of major systems incompatibility between the upper and lower halves of the system.

2.7 Restore Volume

This function is invoked whenever the option 'RV' is selected. It restores a standard volume from a set of flexible discs generated with the Store Volume command in the Amigo/300 operating system or restores any standard volume stored via the Store Volume function described in Paragraph 2.10. Restore volume guarantees that the flexible discs and disc cartridges are from the same set and may be restored in any order.

The function initially outputs the message:

RESTORE VOLUME function now in operation

followed by:

Enter DESTINATION volume information

and then interacts with the user in the identical manner described in Analyze Volume (Paragraph 2.1 of Handbook Section 447B) to obtain the channel, device, and unit number of the volume that is to have data restored to it.

The STORE/RESTORE options allow the user to choose a STORE/RESTORE device. Currently devices of class DISC are supported. The chart below shows which discs can be used as the

source device when Restore Volume is invoked and which can be used as the destination.

SOURCE (Restored From)	DESTINATION (Restored To) 7902	7906	7910	7920	7925
7902	x	x	x	x	x
7906	x	x	x	x	x
7910					
7920	x	x	x	x	x
7925	x	x	x	x	x

x: indicates that the restore combination is allowed

Figure 2.1 - Restore Volume Drive Combinations

When the channel, device, and unit number input indicates that the user has selected one of the supported discs the message:

Please select SOURCE volume (7902, 7906, 7920, 7925)

is output.

If a wrong device name is entered, the message:

Illegal device name entered

is output and the user is again prompted for a legal device name.

If the user inputs a legal device name, but inputs an address which does not correspond to the choosen device name, the message:

Illegal device address entered

is output and the menu is again presented.

The system will now ask for the source volume address information as follows:

Source and destination devices must not be the same

is output (unless the device is a 7906) and the menu is output. Once valid channel, device, and unit numbers have been specified the following interaction occurs:

Inquiry:

Will you please mount a STORE volume?

Valid Response:

Expecting YES or NO

A response of 'NO' terminates the function. A response of 'YES' should be entered only after a STORE volume has been mounted. A STORE/SYSSTORE volume is any media which has been written in the STORE/SYSSTORE format. This is the format used when a volume is Stored using the STORE VOLUME function. After the Store has completed successfully the ASCII characters "DUMP =" will immediately precede the label assigned to that volume (except when storing a 7906 to itself).

Once a STORE volume has been successfully mounted the message:

Volume will be restored from STORE whose ID CODE is <DATE> consisting of <NUMBER> STORE volumes

will be output where <DATE> is of the format month, day, year, time (e.g., May 12, 1978, 10:03 AM) if the Amigo/300 SV command was used and <NUMBER> is the number of volumes in the STORE set. If the volume was stored by the FMUTILIT Store Volume function (Paragraph 2.10), then the 20 character ID CODE is output instead of the <DATE>. Data is then restored to the volume. When all data on the currently mounted STORE volume is restored the message:

Processing on current STORE volume completed

is output followed by:

Inquiry:

Will you please mount the next STORE volume?

Valid Response:

Expecting YES or NO

A response of 'NO' terminates the function. A response of 'YES' should be entered only after the next STORE volume has been mounted. Once all data has been restored the message:

RESTORE VOLUME successfully completed

is output. One of the following messages will be output and then followed by a reissuance of the previously detailed mount message whenever a volume is mounted that cannot be used:

Source volume is not in STORE format

STORE volume has already been restored

STORE volume is not part of current STORE set as it belongs to STORE whose ID CODE is <DATE>

STORED volume must be the same subclass

SYSSTORE volume contains <dev name> data. CANNOT RESTORE.

Cartridge mounted is part of this STORE (or RESTORE)

Cartridge is not in SYSSTORE format. Do you want to use the volume as a STORE volume?

The message:

Flexible disc must be of the same subclass

is output whenever a single sided flexible disc is mounted during a restore from double sided flexible discs and whenever a double sided flexible disc is mounted during a restore from single sided flexible discs. <DATE> will be the date as described above.

If a 7906 was specified and it is to be restored from disc cartridges then the following interaction occurs:

Inquiry:

Will you please mount the disc cartridge on which the upper platter was stored?

Valid Response:

Expecting YES or NO

A response of 'NO' terminates the function. A response of 'YES' should be entered only after the disc cartridge has been mounted. If the disc cartridge mounted does not contain a stored copy of upper platter data then the message:

Disc cartridge does not contain upper platter data

is output and the previously detailed mount message is reissued. If "YES" is entered before the cartridge has been changed then the message:

Disc cartridge was not changed or wrong cartridge mounted.

is output and again it asks for the proper cartridge to be mounted. The system then outputs:

Volume will be restored from STORE whose ID CODE is <xxx>

Function pass number 3 started.

The <xxx> will be the identification code entered by the user when the system was stored using Store Volume. Once the disc cartridge has been successfully copied to the lower platter the message:

STORED copy of upper platter successfully copied to lower platter

is output followed by:

Inquiry:

Will you please mount a disc cartridge to which the upper platter is to be stored?

Valid Response:

Expecting YES or NO

A response of 'NO' terminates the function. A reponse of 'YES' should be entered only after the disc cartridge has been mounted. If you respond 'YES' without mounting another disc cartridge the message:

Disc cartridge was not changed or wrong cartridge mounted.

is output and the previously detailed mount message is reissued. If a write error occurs on the upper platter then the message:

Error detected on upper platter. Please remove

is output and the previously detailed mount message is reissued. The message:

Function pass number 4 started

is issued when the system begins to copy the lower platter to the disc cartridge.

Once the lower platter, which now contains a copy of the original upper platter is copied the message:

Upper platter successfully restored

is output followed by:

Inquiry:

Will you please mount the disc cartridge on which the lower platter was stored?

Valid Response:

Expecting YES or NO

A response of 'NO' terminates the function. A response of 'YES' should be entered only after the disc cartridge has been mounted. If the disc cartridge mounted does not contain a stored copy of lower platter data then the message:

Disc cartridge does not contain lower platter data

Disc cartridge was not changed or wrong cartridge mounted

is output and the previously detailed mount message is reissued. The message:

Function pass number 5 started

is output by the system when the system begins to copy the disc cartridge to the lower platter.

Once the lower platter has been successfully restored the message:

Lower platter successfully restored

is output followed by:

Inquiry:

Will you please mount the disc cartridge to which the upper platter was restored?

Valid Response:

Expecting YES or NO

A response of 'NO' terminates the function. If the wrong cartridge is inserted or the cartridge was not changed the message;

Disc cartridge was not changed or wrong cartridge mounted.

is output and the previous request is made. A response of 'YES' should be entered only after the volume has been mounted. The message:

RESTORE VOLUME successfully completed
System can now be cold loaded

is output if the data on the upper and lower platters contain identical labels for a system volume. If the labels do not reflect a system volume then the message:

Volume labels mismatch

is output and you do not have a cold loadable system. In either case the FMUTILIT menu is output indicating the function has completed.

2.8 Set Printer Echo

This command enables printer echoing when it is invoked. It interacts with the user in the same way as the SPE function of FMDISCUT to obtain the channel and device number of the printer. SE initially will output the FMUTILIT function menu. From that time until FMUTILIT has been exited or the CE command disables echoing, all dialogue will be output to the printer.

2.9 Store Lower

*****USE THIS FUNCTION IN EMERGENCY SITUATIONS ONLY*****

This function is invoked whenever the option 'SL' is selected. It stores the lower platter of a 7906 to a disc cartridge. The 7906 must contain a standard volume when this function is invoked. The function initially outputs the message:

STORE LOWER platter function now in operation

and then interacts with the user in the identical manner described in Analyze Volume (Paragraph 2.1 of Handbook Section 447B) to obtain the channel, device, and unit number of the volume. If the channel, device, and unit numbers do not specify a 7906 then the message:

Device is not a 7906

is output and the function is terminated. If the channel, device, and unit numbers specify a 7906 then the volume is checked to guarantee that it contains a system. If it does not contain a system the message:

Volume label mismatch

is output and the function is terminated. If the volume is a standard volume then the following interaction occurs:

Inquiry:

Will you please mount the disc cartridge on which the lower platter is to be stored?

Valid Response:

Expecting YES or NO

A response of 'NO' terminates the function. A response of 'YES' should be entered only after the disc cartridge has been mounted. If you respond 'YES' without mounting another disc cartridge the message:

Disc cartridge was not changed or wrong cartridge mounted.

is output and the previously detailed mount message is reissued. After the desired cartridge has been inserted and "YES" has been entered the system begins to store the lower platter immediately upon issuing the message:

Function pass number 6 started

Once the lower platter has been stored the message:

STORE LOWER platter successfully completed

is output. If a write error occurs on the upper platter the I/O error message will be output followed by the message:

Error detected on upper platter. Please remove

The previously detailed mount message is then reissued.

2.10 Store Volume

This function is invoked whenever the option 'SV' is selected. The function stores a standard volume to a 7902, 7906, 7920, or 7925 disc drive or stores the upper and lower platters of a 7906 standard volume to two disc cartridges. The chart below shows which drives may be used as the Source (Stored From) and those used as the Destination (Stored To) volumes.

SOURCE (Stored From) \ DESTINATION (Stored To)	7902	7906	7910	7920	7925
7902	x	x		x	x
7906	x	x		x	x
7910	x	x		x	x
7920	x	x		x	x
7925	x	x		x	x

x: indicates that the store combination is allowed
(1): 7902 volumes can be stored under Amigo/300

Figure 2.2 - Store Volume Drive Combinations

As mentioned in Section 2.0 the "Function pass..." statement indicates where in the Store process one currently is at during a STORE/RESTORE from a 7906 to itself (See Section 2.0).

The function initially outputs the message:

Store Volume function now in operation

and then interacts with the user in the identical manner described in Analyze Volume (Paragraph 2.1 of Handbook Section 447B) to obtain the channel, device, and unit numbers of the volume to be stored to and from. If the channel, device, and unit numbers specify a 7906, and the volume's label sectors do not compare, the message:

Volume label mismatch

is output and the function is terminated. Otherwise, a check is made to insure that the volume is labelled properly. If not, then the message:

Please use the 'CV' option to copy a NON-STANDARD volume.

is output and the function is terminated. Otherwise, the system issues the following message:

Please select source volume (7902, 7906, 7920, 7925)

Enter (dev name) STORE volume information

After the address of the source device is input the system follows with the message:

Please input STORE VOLUME user identification label--20 chars

Valid response:

any 20 characters

The Store Volume function will not complete successfully if this label typed in is identical to the previous store volume label. Therefore, a date and time should be included in the ID label. This will prevent the current store operation from accidently overwriting a valid store disc.

If the source and destination devices selected by the addresses input are the same the store must be done on a 7906 and the interaction proceeds as in paragraphs 2.10.1, otherwise, the store is as described in paragraph 2.10.2.

2.10.1 Store 7906 to itself

When a 7906 contains a standard volume and it will be both the source and destination device the interaction proceeds as follows:

Inquiry:

Will you please dismount the disc cartridge and mount another cartridge on which the lower platter is to be stored?

Valid Response:

Expecting YES or NO

A response of 'NO' terminates the function. A response of 'YES' should be entered only after the disc cartridge has been mounted. If you respond 'YES' without mounting another disc cartridge the message:

Disc cartridge was not changed or wrong cartridge mounted.

is output and the previously detailed mount message is reissued. If the cartridge inserted is not in the expected format the system will output the message:

Cartridge is not in SYSSTORE format

Do you want to use the volume as a STORE volume?

is output and the previously detailed mount message is reissued. The message:

Function pass number 0 started

is output by the system as it copies the lower platter to the upper platter.

Once the lower platter has been successfully stored the message:

Lower platter successfully stored

is output. At this point you have a disc cartridge which contains the stored lower platter data and a disc cartridge that contains the upper platter data that has not yet been stored.

If the Store Volume function does not continue to the point where the message "STORE VOLUME successfully completed" is output you must use the Restore Lower function (Paragraph 2.6) to restore the system. If the Store Volume function does continue to the point where the message "STORE VOLUME successfully completed" is output then you can use the Restore Volume function (Paragraph 2.7) to restore the system. However there is an automatic restore provided with the Store Volume function which you have the option of invoking following output of the message "STORE VOLUME successfully completed".

After the lower platter is stored the following interaction then occurs:

Inquiry:

Will you please mount the disc cartridge which contains the upper platter data?

Valid Response:

Expecting YES only

A response of 'YES' should be entered only after the disc cartridge has been mounted. If you respond 'YES' without mounting another disc cartridge the message:

Disc cartridge was not changed or wrong cartridge mounted.

is output and the previously detailed mount message is reissued. The message:

Function pass number 1 started

is output by the system as it copies the lower platter to the upper platter.

Once the upper platter has been successfully stored on the lower platter the message:

Upper platter successfully copied to lower platter

is output followed by:

Inquiry:

Will you please mount the disc cartridge on which the upper platter is to be stored?

Valid Response:

Expecting YES only

A response of 'YES' should be entered only after the disc cartridge has been mounted. If you respond 'YES' without mounting another disc cartridge the message:

Disc cartridge was not changed or wrong cartridge mounted.

is output and the previously detailed mount message is reissued. If a write error occurs on the upper platter then the message:

Error detected on upper platter. Please remove

is output and the previously detailed mount message is reissued. The message:

Function pass number 2 started

is output by the system as it copies the lower platter to the upper platter.

Once the lower platter, which now contains a copy of the upper platter, has been successfully stored the message:

STORE volume successfully completed

is output. The message:

RESTORE lower platter phase started

is output and then the following interaction occurs:

Inquiry:

Will you please mount the disc cartridge on which the lower platter was stored?

Valid Response:

Expecting YES or NO

A response of 'NO' terminates the function and requires you to use the RESTORE VOLUME function of FMUTILIT (detailed in 2.7) to restore a system to the 7906. A response of 'YES' should be entered only after the disc cartridge has been mounted. If the disc cartridge mounted does not contain a stored copy of lower platter data then the message:

Disc cartridge does not contain lower platter data

is output and the previously detailed mount message is reissued. After "YES" is entered the message:

Function pass number 5 started

is output by the system while the upper platter is being copied to the lower platter.

Once the lower platter has been successfully restored the message:

Lower platter successfully restored

is output followed by:

Inquiry:

Will you please mount the disc cartridge which contains the upper platter data?

Valid Response:

Expecting YES or NO

A response of 'NO' terminates the function. A response of 'YES' should be entered only after the disc cartridge has been mounted. If you respond 'YES' without mounting another disc cartridge the message:

Disc cartridge was not changed or wrong cartridge mounted.

is output and the previously detailed mount message is reissued. The message:

RESTORE VOLUME successfully completed System can now be cold loaded

is output if the data on the upper and lower platters contain identical labels for a system volume.

2.10.2 Store to 7902/6/20/25 disc from any disc (except 7902)

When the source and destination addresses entered do not reference the same device the message:

> Next STORE volume on channel no. 'c' device no. 'd' unit no. 'u' ready?

is output. A 'YES' response will continue the function and a NO response will terminate it.

If the addresses entered do specify the same device and it is not a 7906 the message:

> Source and destination devices must not be the same

is output and the menu is presented. After the first store disc is written and the user does not change the store disc but answers 'YES', the following message is output:

> STORE volume on channel 'c' device 'd' unit 'u' was not changed.

The user is again prompted for the next STORE disc.

If the disc is not in STORE format (Reference 3.7 for STORE format explanation), the message:

> Volume mounted on channel no. 'c' device no. 'd' unit no. 'u' is not in STORE format.
>
> Do you want to use the volume as a STORE volume?

is output. If the answer is 'YES', then the function continues. Otherwise, the message:

> Next STORE volume on channel no. 'c' device no. 'd' unit no. 'u' ready?

is again output. A 'NO' response will terminate the function. If the new volume is not in STORE format, the above is repeated.

After the user answers 'YES' to the question above and the new disc volume is found to be in STORE format, the message:

> Volume Store will require xxx STORE volumes

is output where xxx is the number of volumes required to complete the store.

When storing a 7920 to a 7920 or a 7925 to another 7925 and more than 1 pack must be used because the pack is completely full the message;

Please use the CV option to STORE the volume

will be output and the menu will be presented.

After the completion of one volume (other than the above mentioned case), the message:

Processing of current STORE volume on channel no. 'c' device no. 'd' unit no. 'u' completed

is output and the above system outputs are repeated starting with the message:

Next STORE volume on channel no. 'c' device no. 'd' unit no. 'u' ready?

until the entire system volume has been stored. Then the message:

Volume STORE successfully completed

is output.

If an error is detected while writing to a volume, the message:

Bad STORE volume. Please replace

is output after the I/O error message. The user is then prompted to mount the next STORE disc.

3.0 ERROR INTERPRETATION

If any I/O errors are encountered during the execution of any of the File Management Utility or Patch Volume functions the user will be informed of the error through one of the messages detailed in this section.

The error messages are divided into correctable errors and uncorrectable errors. For a correctable error the user will be informed of the problem and asked to respond 'YES' to a prompt when the condition has been corrected. If the user responds 'NO' then the function is terminated and the user is prompted to select another function.

For an uncorrectable error the user is informed of the problem and the function is automatically terminated and the user is prompted to select another function. The messages are as follows:

Correctable Errors

```
Device on chan/dev/unit is not ready
Device on chan/dev/unit is write protected
Flexible disc mounted is not in HP format
Turn format switch on - chan/dev/unit
```

Uncorrectable Errors

```
Drive attention on chan/dev/unit
I/O failed after 'n' retries
Device is not a disc
Device is not a printer
Channel type is not GIC
Channel is not present
I/O error on chan/dev/unit
Watchdog timer or parameter error
```

followed by:

```
Cannot report status
Cylinder 'c', head 'h', sector 's'
Relative sector number 'r'
```

and then:

```
Channel status      Status1        Status2
   (channel          (status        (status
     status)          1 word)        2 word)
```

"chan/dev/unit" will be the decimal numbers for the channel, device, and unit. 'n' will be the decimal number of retries. 'c', 'h', 's', and 'r' are,

respectively, the decimal numbers for the cylinder, head, sector, and relative sector addressed.
All COPY, STORE, and RESTORE volume commands read/verify every data transfer.

Whenever the system encounters a Disc I/O error the appropriate message listed above will be followed by a message of the following form:

```
Disc I/O error xxx-- (text message)
on device (dev name) ,channel 'c', device 'd', unit 'u'
at relative sector z (cylinder 'c', head 'h', unit 'u').
```

The error number xxx and (text message) which would be used are the same as those output by Amigo/300. The error numbers and messages are listed in Handbook Section 199 -- Error and Status Codes.

The message:

```
Verification of data written has failed.
```

is output by the system if a read verify of the lower platter of a 7906 has failed (a read verify is done after every write) . If the read verify fails after a write to a removable cartridge the above message will be followed by the message:

```
Error detected on upper platter. Please remove.
```

The function will continue after the cartridge has been replaced.

If an error occurs and the printer echo is not enabled, the message:

```
Is the above message recorded?
```

is output. The user may wish to record the message and then type 'Y' or 'N' ('YES' or 'NO'). If the above is not typed, the message:

```
Expecting YES or NO
```

is output.

Following the output of any of the correctable errors the user will be prompted with:

```
Inquiry:

   Drive ready?

Valid Response:
```

Expecting YES or NO

The user should respond 'YES' only after correcting the problem. A response of 'NO' results in the function being terminated and the user prompted to select another File Management Utility or Patch Volume function.

-hp-

SECTION 447D

HP 300 FILE MANAGEMENT
STAND ALONE PATCH VOLUME UTILITY
(FMPATCHV)

EXTERNAL REFERENCE SPECIFICATION

(Program Revision 01.XX)

NOTICE

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

Hewlett-Packard Company
19447 Pruneridge Ave., Cupertino, California 95014 U.S.A.

Printed in U.S.A.

TABLE OF CONTENTS

1.0 INTRODUCTION

1.1 Product Abstract

This document details the external reference specifications for FMPATCHV, one of the File Management standalone utilities on the HP 300 system.

1.2 Overview

File Management provides the user several basic utilities which, because of the functions they perform, must be executed in the stand alone environment. These utilities are presented to the user as options within the FMUTILIT (File Management Utilities), FMDISCUT (File Management Volume Utilites), and FMPATCHV (File Management Patch Volume) programs which are provided to the user on the cold loadable Diagnostic and Utility Package (DUP) flexible discs. The intended uses of each program are listed below:

UTILITY	USES
FMDISCUT:	detect and spare bad tracks indicate the condition of a surface prepare a surface for use [covered in Handbook Section 447B]
FMUTILIT:	copy non-standard volumes and cartridges back up systems Store or Restore information replace the primary with the back-up operating system [covered in Handbook Section 447C]
FMPATCHV: (DUPCE only)	inspect or modify the the contents of a volume output the contents of a volume in hex, octal or decimal [covered in this Handbook Section]

The details of how to load a DUP flexible disc and the environment it initially provides the user are described in the DUS handbook section (440). This document describes only the FMPATCHV programs.

1.3 Required Hardware

The HP 300 configuration sufficient to run DUS is required. This includes 256 Kbytes of memory, a GIC, CPU, 7902 and an IDS or ADCC with a terminal as a console.

1.4 Required Software

The CE Diagnostic and Utility Package containing FMPATCHV (DUPCE).

2.0 OPERATION

This program is present only on the DUPCE flexible disc and enables the CE to display and/or modify any data on a volume located in the user area. Data can optionally be output to a printer and can be addressed in a variety of modes. The default base for all numeric input and output is hexadecimal(!). The default base for input can be overridden on any input request by preceding the numeric value input with a '#'(decimal) or '%'(octal) sign. No documentation of this function is to appear in any user manual.

Once the DUP flexible disc has been successfully cold loaded and the prompt character ":" is present the user must enter FMPATCHV to enter the File Management Patch Volume program.

The program initially outputs the message:

  Welcome to the volume patch utility revision xx.xx

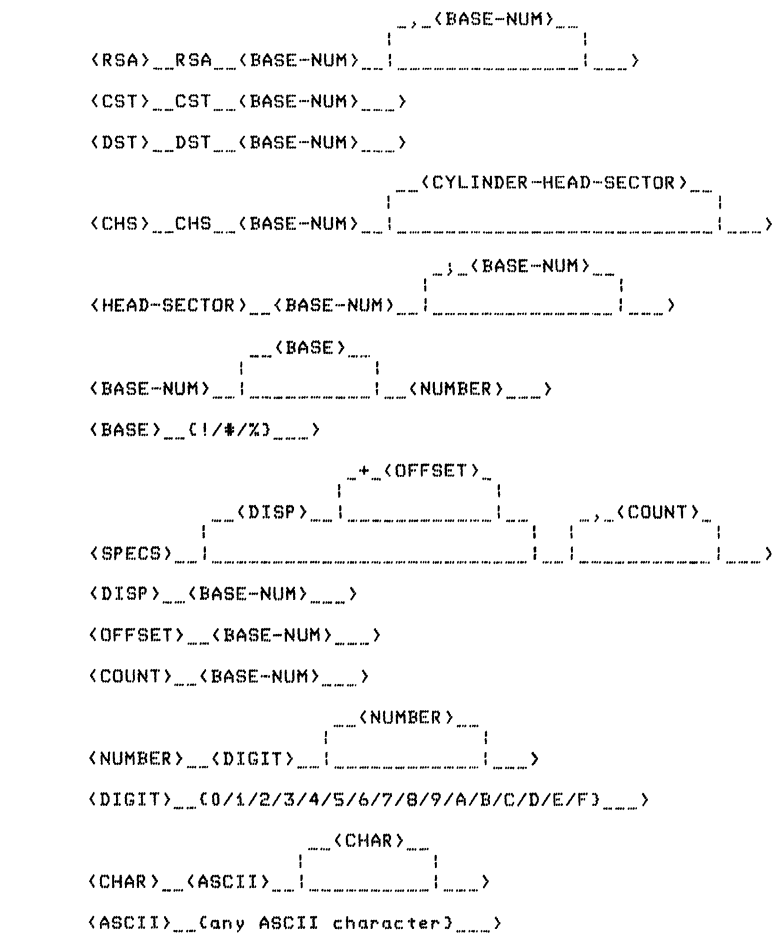
and then interacts with the user in the identical manner detailed in FMDISCUT Analyze Volume [Paragraph 2.1 of Handbook Section 447B] to obtain the channel, device, and unit number of the volume to patch and to get the volume mounted. The menu is displayed and the user is then prompted for a command with the output of:

  Enter command:

If an illegal command is entered the message:

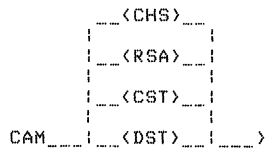
  Illegal command entered

is output followed by a prompt for another command. The commands and their functions are detailed in sections 2.1 to 2.15. The user must enter all three characters of the command mnemonic. The following definitions are used throughout this section:


,-(BASE-NUM)-
<RSA)--RSA--(BASE-NUM)--|_____|-->
<CST)--CST--(BASE-NUM)-->
<DST)--DST--(BASE-NUM)-->
--(CYLINDER-HEAD-SECTOR)--
<CHS)--CHS--(BASE-NUM)--|_____|-->
-}-(BASE-NUM)-
<HEAD-SECTOR)--(BASE-NUM)--|_____|-->
--(BASE)--
<BASE-NUM)--|_____|--(NUMBER)-->
<BASE)--(!/#/%)-->
-+-(OFFSET)-
--(DISP)--|_____|--
,-(COUNT)-
<SPECS)--|_____|--|_____|-->
<DISP)--(BASE-NUM)-->
<OFFSET)--(BASE-NUM)-->
<COUNT)--(BASE-NUM)-->
--(NUMBER)--
<NUMBER)--(DIGIT)--|_____|-->
<DIGIT)--(0/1/2/3/4/5/6/7/8/9/A/B/C/D/E/F)-->
--(CHAR)--
<CHAR)--(ASCII)--|_____|-->
<ASCII)--(any ASCII character)-->
(BASE-NUM)
(BASE)
(NUMBER)
(CYLINDER-HEAD-SECTOR)
(HEAD-SECTOR)
(BASE-NUM)
(BASE-NUM)
(DISP)
(OFFSET)
(COUNT)
(SPECS)
(BASE-NUM)
(BASE-NUM)
(BASE-NUM)
(DIGIT)
(NUMBER)
(CHAR)
(ASCII)

2.1 Choose Address Mode

The syntax graph is:



This command allows the user to select the mode in which data is to be addressed. The user can select relative sector address(<RSA>), cylinder/head/sector address(<CHS>), code segment table address(<CST>) and data segment table address(<DST>). The default is relative sector address(<RSA>) at relative sector number 0(i.e., the sector at cylinder 0, head 0, sector 0). The user remains in the selected address mode until another mode is choosen.

<BASE-NUM> specifies the code segment table entry to address in <CST> addressing mode, the data segment table entry to address in <DST> addressing mode, and the cylinder, head, and sector to address in <CHS> addressing mode. In <CHS> addressing mode the address is specifed in the following order: cylinder, head, sector. Cylinder address is required and head and sector will default to zero if they are not specified(Note: if head is not specified then sector can not be specified).

If one <BASE-NUM> is specified for <RSA> addressing then it is assumed to be the double word address of the sector to address, however, if <BASE-NUM>, <BASE-NUM> is entered then the leftmost <BASE-NUM> is taken as the value to enter in the leftmost word of the double word sector address and the rightmost <BASE-NUM> is taken as the value to enter in the rightmost word of the double word sector address.

The input of an illegal address mode, illegal numeric value, illegal CST number, illegal DST number(data segment numbers in the range !22 to !32 inclusive are illegal as these entries are used for memory bank addresses and have no disc addresses associated with them), or illegal sector address results, respectively, in one of the following messages being output:

```
Illegal address mode selected
Illegal value(s) entered
Illegal CST number
Illegal DST number
Volume address entered is out of bounds
```

followed by the user being prompted to enter another command. If <CST> or <DST> addressing is selected and the volume is not a SYSTEM volume the message:

Volume is not a system volume

is output and the user is then prompted for another command.

2.2 Clear Print Echo

The syntax graph is:

CPE___>

This command disables the echoing of all data output by the DSP, DVD, and DCA commands to a printer. The command is ignored if echoing is not currently enabled.

2.3 Display Current Address

The syntax graph is:

DCA___>

This command outputs the current address to the console and printer (if echoing is enabled). The data ouput consists of the channel, device, and unit number of the volume being patched, the relative sector number and cylinder, head, sector number of the currently addressed sector, and, optionally, the block number (if block addressing had been selected by the SBN command), or the CST number (if CST addressing was in effect), or the DST number (if DST addressing was in effect).

All data is output in hexadecimal, octal, and decimal bases except the channel, device, and unit number which are output in decimal.

2.4 Disable Decompiler

The syntax graph is:

DDC___>

This command allows the user to disable the decompiler so that data output by the DSP command is not ouput in decompiled format. The command is ignored if it is entered when decompiled format is already disabled.

2.5 Display

The syntax graph is:

```
DSP___(SPECS)___
```

This command outputs the data specified by (SPECS) from the currently addressed sector, block, code segment, or data segment to the console and printer (if echoing is enabled).

The data is output in the base specified by the SDB command. Hexadecimal(!), decimal(#), and octal(%) based output is printed 16 characters to a line with the address of the data appearing to the left of the line followed by the data itself in the current base and then the ASCII representation of the data (unprintable characters are printed as a "."). Ascii based output is printed with the address of the data to the left of the line followed by up to 48 ASCII characters per line.

(SPECS) defines relative to the start of whatever is currently addressed the displacement from which data is to be output and the length of the data to output.

The values for (BASE), (OFFSET), and (COUNT) are taken as bytes if the current base is ASCII(A), otherwise, they are taken as words. (COUNT) is optional and defaults to one if not specified. (OFFSET) and (BASE) are optional. (OFFSET) defaults to zero when not specified. (BASE) defaults to octal.

The address that appears to the left of each line displayed is output in the display base for all cases except ASCII. In this case the address is output in hexadecimal and reflects the byte address of the data being displayed.

The input of any illegal value for (BASE), (OFFSET), or (COUNT) or a (BASE) and (OFFSET) whose sum exceeds the end of the current sector, code segment, data segment, or block results, respectively, in the output of one of the following messages:

```
Illegal value(s) entered.
Illegal starting address entered.
```

and the user being prompted for another command.

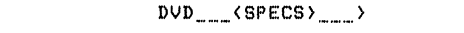
2.6 Dump Sectors

The syntax graph is:

This command dumps the number of sectors specified by ⟨BASE-NUM⟩ starting with the currently addressed sector to the console and printer(if echoing is enabled). Data is output in the same format detailed in the DSP command. If ⟨BASE-NUM⟩ is not specified then one sector is displayed.

2.7 Dump Volume Directory

The syntax graph is:



This command dumps all the used blocks of the directory to the console and printer(if echoing is enabled). The amount of data to be output from each block is specified by ⟨SPECS⟩. Data is output in the same format detailed in the DSP command with the exception that each new block that is output is preceded by its block number.

Any invalid data entered for ⟨BASE⟩, ⟨OFFSET⟩, or ⟨COUNT⟩ results in the same errors detailed in the DSP command. If the volume is not labelled the message:

Volume is not labelled.

is output and the user is prompted for another command. If the current addressing mode is CST or DST then the message:

Illegal command for address mode selected

is output and the user is prompted for another command. If the entry contained in the directory which describes the directory itself can not be found then the message:

Volume directory is destroyed

is output and the user is prompted for another command.

2.8 Enable Decompiler

The syntax graph is:

```
EDC___>
```

This command allows the user to enable the decompiler so that data output by the DSP command is output in decompiled format. Data output in this manner is displayed two words to the line where each word printed consists of its address and contents in the current base followed by its ASCII representation and the instruction mnemonic(s).

Decompiled output is automatically enabled when <CST> addressing is selected and automatically disabled when any other addressing mode is selected. If an EDC command is entered when decompilation is already enabled it is ignored. If the EDC command is entered in other than <CST> addressing mode it causes all subsequent data to be displayed in decompiled format.

2.9 Exit

The syntax graph is:

```
EXI___>
```

This command allows the user to exit the Patch Volume function and return to select another File Management Utility function.

2.10 Modify

The syntax graph is:

```
MOD___<SPECS>___>
```

This command allows the user to modify the contents of a sector, block, code segment, or data segment. <BASE> and <OFFSET> detail where the modification is to begin and <COUNT> specifies the length of the modification. The input of invalid data for <BASE>, <OFFSET>, or <COUNT> results in the same errors detailed in the DSP command.

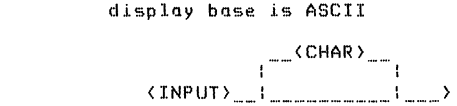
If the display base is hexadecimal(!), decimal(#), or octal(%) then the address of the data being modified is output, followed by the current value of the data(both are output in the current base) to the console. If the display base is ASCII then the byte address of the data in hexadecimal followed by up to 20 characters of the data itself is output to the console.

The user is prompted for the new value by positioning the cursor to the right of the current value of the data. <BASE>, <OFFSET>, and <COUNT> are taken as bytes if the current addressing mode is

ASCII, otherwise , they are taken as words. The user is prompted for all new values until the number of words/bytes specified in <COUNT> have been exhausted. The format of the input is:

display base is hexadecimal, decimal, or octal

display base is ASCII



If <BASE-NUM> or <CHAR> is not specified then the current value is not modified. "R" means repeat and will repeat for the number of words specified by <LENGTH>. If "R" and <LENGTH> are not specified only one word is modified. If the display base is ASCII and the data to modify contains unprintable characters then the message:

Non-printable characters in string.

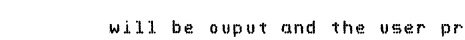
will be ouput and the user prompted for another command. If an illegal value is input for <BASE-NUM> or <LENGTH> the message:

Illegal value(s) entered.

will be ouput and the user will be reprompted for the patch. If <LENGTH> applied to the current address of the data being patched exceeds the <COUNT> of the modification then the patch will cease at the address reflected by the value of <COUNT>. If less than 20 characters are input when the display base is ASCII then the string is left justifed within the data string to patch and characters to the right remain unchanged.

2.11 Next

The syntax graph is:



This command allows the user to address the next sector, next block, next code segment, or next data segment.

The determination of what is next accessed is dependent upon the current address mode. The next code segment is addressed if the current address mode is <CST>, the next data segment if the current address mode is <DST>, the next block if the current address mode is <CHS> or <RSA> and the user has issued a SBN command, and the next sector if the current address mode is <CHS>

or <RSA> and SBN has not been specified.

The use of NXT when the last CST is currently addressed, the last DST is currently addressed, or the last block or last sector is addressed results, respectively, in one of the following messages being output:

```
Illegal CST number.
Illegal DST number.
Volume address entered is out of bounds.
```

and the user prompted for another command.

2.12 Set Block Number

The syntax graph is:

```
SBN___<BASE-NUM>___>
```

This command allows the user to specify a block to access relative to the current address(relative sector or cylinder/head/sector). <BASE-NUM> specifies the number of the block to access relative to the current address. The current block is block zero and each block is eight sectors in length.

The use of SBN when the current adddressing mode is <CST> or <DST>, or the input of an illegal block number or block number which exceeds the limits of the volume results, respectively, in one of the following messages being output:

```
Illegal command for addressing mode.
Illegal value(s) entered.
Volume address entered is out of bounds.
```

and the user prompted to enter another command. Once block number access has been selected it remains in effect until another CAM command is entered.

2.13 Set Display Base

The syntax graph is:

```
SDB___<BASE>___>
```

This command allows the user to select the display base in which all data from the DSP, DVD, and MOD commands is to be output. The default is hexadecimal(!), however, the user can select decimal(#), octal(%), or ASCII(A). All unprintable characters will be printed as a "." when ASCII has been selected. If an illegal base is selected the message:

```
Illegal display base selected.
```

is output and the user is prompted for another command.

2.14 Set Print Echo

The syntax graph is:

```
SPE____>
```

The successful completion of this command results in the echoing of all data output by the DSP, DVD, and DCA commands to a printer. Data is output only to the console until this command is entered. The command interacts with the user to obtain the channel and device number of the printer as follows:

Inquiry:

Enter printer's channel number:

Valid Response:

Expecting 1 to 15.

Inquiry:

Enter printer's device number:

Valid Response:

Expecting 0 to 7.

The values input to these prompts are assumed to be in decimal(they can not be entered in any other base). Once the channel and device number of a printer have been input printer echoing is enabled.

If the device does not identify itself as a 2631A or 2608A the message:

Device is not a printer.

is output.

2.15 Show

The syntax graph is:

```
SHW____>
```

This command allows the user to output to the console the list of all valid commands and their parameters to the console. The data output is:

Choose addressing mode	CAM	CHS (cylinder)[,(head)[,(sector)]
	CAM	CST (cst number)
	CAM	DST (dst number)
	CAM	RSA (sector1)[,(sector2)]
Clear print echo	CPE	
Display current address	DCA	
Disable decompiler	DDC	
Display data	DSP	[(disp)[+(offset)]][,(count)]
Display volume directory	DVD	[(disp)[+(offset)]][,(count)]
Dump sectors	DMP	[(count)]
Enable decompiler	EDC	
Exit from volume patch	EXI	
Modify data	MOD	[(disp)[+(offset)]][,(count)]
Next-sector,block,cst,or dst	NXT	
Set block mode addressing	SBN	(block-number)
Set display base	SDB	(!/#/%/A)
Set print echo	SPE	
Show list of commands	SHW	

3.0 ERROR INTERPRETATION

If any I/O errors are encountered during the execution of any of the File Management Utility or Patch Volume functions the user will be informed of the error through one of the messages detailed in this section.

The error messages are divided into correctable errors and uncorrectable errors. For a correctable error the user will be informed of the problem and asked to respond 'YES' to a prompt when the condition has been corrected. If the user responds 'NO' then the function is terminated and the user is prompted to select another function.

For an uncorrectable error the user is informed of the problem and the function is automatically terminated and the user is prompted to select another function. The messages are as follows:

Correctable Errors

```
Device on chan/dev/unit is not ready
Device on chan/dev/unit is write protected
Flexible disc mounted is not in HP format
Turn format switch on - chan/dev/unit
```

Uncorrectable Errors

```
Drive attention on chan/dev/unit
I/O failed after 'n' retries
Device is not a disc
Device is not a printer
Channel type is not GIC
Channel is not present
I/O error on chan/dev/unit
Watchdog timer or parameter error
```

followed by:

```
Cannot report status
Cylinder 'c', head 'h', sector 's'
Relative sector number 'r'
```

and then:

```
Channel status      Status1        Status2
   (channel         (status        (status
     status)         1 word)        2 word)
```

"chan/dev/unit" will be the decimal numbers for the channel, device, and unit. 'n' will be the decimal number of retries. 'c', 'h', 's', and 'r' are, respectively, the decimal numbers for

the cylinder, head, sector, and relative sector addressed. All COPY, STORE, and RESTORE volume commands read/verify every data transfer.

Whenever the system encounters a Disc I/O error the appropriate message listed above will be followed by a message of the following form:

```
Disc I/O error xxx-- (text message) on device (dev name)
,channel 'c', device 'd', unit 'u' at relative sector z
(cylinder 'c', head 'h', unit 'u').
```

The error number xxx and (text message) which would be used are the same as those output by Amigo/300. The error numbers and messages are listed in Handbook Section 199 -- Error and Status Codes.

The message:

```
Verification of data written has failed.
```

is output by the system if a read verify of the lower platter of a 7906 has failed (a read verify is done after every write) . If the read verify fails after a write to a removable cartridge the above message will be followed by the message:

```
Error detected on upper platter. Please remove.
```

The function will continue after the cartridge has been replaced.

If an error occurs and the printer echo is not enabled, the message:

```
Is the above message recorded?
```

is output. The user may wish to record the message and then type 'Y' or 'N' ('YES' or 'NO'). If the above is not typed, the message:

```
Expecting YES or NO
```

is output.

Following the output of any of the correctable errors the user will be prompted with:

```
Inquiry:

   Drive ready?

Valid Response:

   Expecting YES or NO
```

The user should respond 'YES' only after correcting the problem. A response of 'NO' results in the function being terminated and the user prompted to select another File Management Utility or Patch Volume function.

-hp-

SECTION 448

WORKOUT300

(PROGRAM REVISION 01.XX)

EXTERNAL REFERENCE SPECIFICATION

NOTICE

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

Hewlett-Packard Company
19447 Pruneridge Ave., Cupertino, California 95014 U.S.A.

Printed in U.S.A.

TABLE OF CONTENTS

1.0 WORKOUT300 OVERVIEW

Purpose: WORKOUT300 uses system services to test system access to peripherals under a normal or heavy system load environment. By peripheral access, one means using the physical interconnections between the peripherals and the system. By system services, one means the software services available to the user through the intrinsic calls.

WORKOUT300 currently enables a user to selectively exercise any or all of the following peripherals attached to the HP 300: IDS, any disc, any non menu terminal, any printer.

The heart of WORKOUT300 is a collection of functions that individually exercise certain portions of the operating system and/or peripherals.

Through a user supplied script, these functions (and/or user supplied programs) will be selectively run serially and/or concurrently.

WORKOUT300 builds an exercising environment according to this user supplied script. The script gives the user the option of 1) constructing or not constructing a multitasking and/or multiprogramming environment, 2) specifying which peripherals are to be exercised, including/excluding the IDS.

In addition to WORKOUT300 supplied functions, the user has the option of specifying his own program in the script file. The program will be PLOADed by WORKOUT300 and launched into execution according to the script.

WORKOUT300 is written in BASIC and runs under the operating system in non-privileged mode.

2.0 THE SCRIPT LANGUAGE

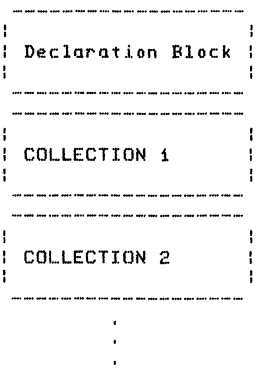
The script language allows two levels of concurrency: unsynchronized global concurrency and synchronized local concurrency.

Local concurrency allows the user to form groups of tasks and/or programs. All tasks/programs within an active group begin execution at the same time and run concurrently. All tasks/programs within the active group must terminate before the next group becomes active. In this way concurrency is synchronized at the beginning of an active group.

Global concurrency allows the user to form collections of groups. Each collection runs with local concurrency amongst its member groups, as described above, while all the collections run concurrently and independently. The only synchronization between the collections is that they all start execution at the same time.

Backward branching capabilities within a collection exists, allowing a sequence of groups to be executed repeatedly, indefinitely. Forward branching capabilities do not exist. Such a branch would not be useful since decision capabilities do not exist.

Script Language Syntax



2.1 Declaration Block

The declaration block is the region of the script that associates task identifiers with WORKOUT300 supplied functions and program identifiers with user supplied programs. These terms are defined below. The declaration block appears at the beginning of the script and consists of a non-empty list of declarations.

Task or program identifiers, once declared, can be used any number of times in any number of collections. Furthermore, a single WORKOUT300 supplied function or user supplied program can be associated with any number of identifiers. This allows a single function or program to run with different parameters in a multi-tasking and/or multiprogramming mode. Hence one can have the following:

```
T1:=TASK1[,,10000,2]
T2:=TASK1[,,5000,4]
COLLECTION
T1,T1,T1,T1,T2
T2
```

2.1.1 WORKOUT300 Supplied Functions:

```
Tn:=TASKm[(parameter list)]
```

where

n,m are 1, 2, or 3 digit numbers.

Tn is the task identifier associated with the TASKm[...] function specification and used to refer to this specification in the collection blocks.

TASKm is a WORKOUT300 supplied function name. For a list of existing functions see Section 6.0 below. The writeup on each Exerciser function describes its operation and functional parameters.

(parameter list) is a list of functional parameters to be passed to the function when TSTARTed (see writeup on individual exerciser functions).

EXAMPLE:

```
T4:=TASK3[]
T17:=TASK4[TERM3,2,13]
```

The square brackets enclosing the parameter field are a necesary part of the script language syntax. Even if all default values are assumed and the parameter field is empty, the square brackets must be present.

2.1.2 User Supplied Programs:

Pn:=programname[(parameter list)]

where

n is a 1, 2, or 3 digit number.

Pn is the program identifier associated with the programname[...] specification and used to refer to this specification in the collection blocks.

programname is a fully qualified user program workspace name to be PLOADed and PSTARTed by WORKOUT300.

(parameter list) is the list of parameters to be passed to the user program via the PARMS parameter of the PSTART system intrinsic. This information is available to the program via the BGETPARMS intrinsic (see Section 5.3).

EXAMPLE:

P32:=IOEXER(DIAG),(DSKOV)[INFILE(MYDOM),(DSFLOP),,,33,2]

The square brackets enclosing the parameter field are a necessary part of the script language syntax. Even if all default values are assumed and the parameter field is empty, the square brackets must be present.

For more information concerning subprograms running under WORKOUT300, see Section 5.0.

2.2 Collections

Collections are non-empty lists of groups, beginning with a COLLECTION statement, and ending with either a group or a GOTO statement. All collections run concurrently and unsynchronized (they all start at the same time, however.) A current maximum of ten collections may be defined in a script.

COLLECTION:

```
COLLECTION [comment]

Group1
Group2
  .
  .
  .

COLLECTION [comment]

Group1
Group2
  .
  .
  .

  .
  .
  .
```

where

comment is any user supplied message. It is optional and can be used for documentation purposes.

Group1, Group2, ... are non-empty lists of program or task identifiers separated by commas (Groups are defined below).

EXAMPLE:

```
            .
            .
            .
      COLLECTION     IO TESTER
      T1, T7, T2, P4, P4, T3
      P4,P4,P4
      P4
      T4, P3, T1
      COLLECTION          DEVICE EXERCISER
      T3
      T4, P4, T1
   15 T2
      T2, T3, T7
      GOTO 15
            .
            .
            .
```

2.3 Groups

A group is a list of one or more (previously defined) task/program identifiers, possibly preceded by a line number. A group occupies one line of script. All functions and/or programs associated with the identifiers in an active group run concurrently, all starting approximately at the same time. The next group in the collection will not become active until all tasks/programs in the currently active group have finished executing. There is a current maximum of twenty identifiers per group.

Identifiers within a group must be separated by commas. A group cannot end in a comma.

Group:

[label] identifier, identifier, ..., identifier

where

label is optional and consists of a unique (within current collection) 1 or 2 digit number followed by a blank.

identifier is a task/program identifier declared in the DECLARATION BLOCK.

EXAMPLE:

```
     T4, T7, P14, T3
  15 T3, T7
```

NOTE: The difference between

```
COLLECTION
T1, T2, T3
```

and

```
COLLECTION
T1
T2
T3
```

is that, in the first instance, WORKOUT300 functions associated with the task identifiers T1, T2, and T3 are run concurrently (since all are within the same group). In the second instance, the functions are run serially (since each is in a different group).

2.4 GOTOs

A GOTO statement directs WORKOUT300 operation to a previously defined group within the same collection. The presence of a GOTO in a collection establishes an indefinite looping situation; the collection will never terminate on its own.

GOTO:

GOTO label

where label is a label previously defined in the current COLLECTION and consisting of a 1 or 2 digit number.

EXAMPLE:

GOTO 14

Note: Since no decision making is possible in the script language, more than one GOTO statement per collection will not result in a meaningful script. Since GOTOs must reference back, all groups following a GOTO within a collection will not be used. A GOTO results in indefinite looping.

3.0 OPERATING INSTRUCTIONS

The script needs to exist prior to execution of WORKOUT300 due to the inability of WORKOUT300 to interactively build a script. There are at least two ways to build a script. First, one could start with one of the standard scripts and modify it appropriately in the WORKOUT300 environment. Second, one could use the TYPIST subsystem to create the script from scratch.

If a script has been created from scratch (the second method), then one must issue the command

```
COPY MODULE modulename TO UNN FILE scriptfilename
```

while in TYPIST in order to get the necessary sequential file containing the script. This script can again be modified while in the WORKOUT300 environment.

The standard scripts and the WORKOUT300 program are located on a store volume floppy. To load the program and standard scripts into the system, issue the following AMIGO commands while the WORKOUT300 floppy is in the floppy drive:

```
ID PUBLIC

RESTORE WORKSPACE WORKOUT FROM FLEXDISC
RESTORE FILE SCRIPT1 FROM FLEXDISC
RESTORE FILE SCRIPT2 FROM FLEXDISC
```

If the user does not intend on using one of the standard scripts, the last two RESTORE commands need not be issued.

The script file name can be communicated to WORKOUT300 in a variety of ways:

(1) by the USING parameter of the run command:

```
RUN WORKOUT USING scriptfilename
```

(2) by EQUATING file SCRIPT with the actual script file name:

```
EQUATE FILE SCRIPT TO scriptfilename

RUN WORKOUT
```

(3) by being prompted by WORKOUT300 via ASKOP

```
                    RUN WORKOUT
 ASKOP message:  PLEASE ENTER SCRIPT FILE NAME
 User response:  REPLY scriptfilename
```

WORKOUT300 next attempts to create a keyed sequential file by the name SXLOG(PUBLIC) to serve as the disc log file. If a file by this name already exists at WORKOUT300 run time, then WORKOUT300

will ASKOP permission to destroy the file and recreate it type keyed sequential. If a negative response is given to this request, WORKOUT300 will prompt for a new file name (fully qualified or default to current domain).

Once the source file for the script and the disc log file have been determined, WORKOUT300 will scan the script for errors and display the script on the IDS.

If errors are encountered, WORKOUT300 will prompt the user for a correction. If all corrections are made, or if no errors were encountered, the entire script will be displayed on the IDS and WORKOUT300 will enter a state in which the user can modify the script.

At this point the user can do anything to the script with the exception of inserting lines.

NOTE: If all groups of a collection have been deleted, then the collection statement must also be deleted.

Two softkeys are enabled:

(1) START WORKOUT causes WORKOUT300 to rescan the script and start the exercise.

(2) ABORT WORKOUT causes WORKOUT300 to abort.

EXAMPLE

To run the default script SCRIPT1:

Restore the workspace WORKOUT and the file SCRIPT1 as indicated above.

PURGE FILE SXLOG(PUBLIC) ... if it exists already

RUN WORKOUT USING SCRIPT1

When the windows for the program appear, press the softkey labeled START WORKOUT. WORKOUT300 should run to completion.

NOTE: If the printer test is not to be run, or if the device name of the printer is different than "PRINTER", then the script will have to be modified.

4.0 OUTPUT FROM WORKOUT300

Output from WORKOUT300 comes in a variety of forms: as IDS displays, called Exerciser State Displays; and as log file output.

4.1 WORKOUT300 State Displays

There are currently two state-of-the-exerciser IDS displays available; the STATUS display and the LOGFILE display. Switching from one to the other (as well as aborting and backgrounding WORKOUT300) is accomplished interactively through labelled softkeys.

4.1.1 The STATUS Display State

The STATUS display displays the script using video enhancements to indicate the "state" of WORKOUT300. Each identifier in every group in all collections of the script is in exactly one of three states:

(1) Currently executing (Logged on). Here the identifier is displayed in inverse video. See section 5.2.1.

(2) Queued for execution (TSTARTed or PSTARTed) but not yet logged-on. Here the identifier is blinking.

(3) Not queued and not logged on. Here the identifier is not enhanced.

The status window is opened scrollable for horizontal scrolling. Vertical scrolling should be accomplished via the softkey options. This avoids having the window repositioned whenever a state change occurs in a portion of the script not currently displayed in the window.

4.1.2 The LOGFILE Display State

The LOGFILE display displays the contents of the disc log file. This window is opened scrollable to allow examination of any portion of the file.

4.1.3 The BACKGROUND State

WORKOUT300 defaults to the foreground position; i.e., status displayed on the IDS. Except when an IDS exerciser function is exercising the IDS, the user has the option of forcing WORKOUT300 into a state where the IDS can be reallocated to another job without hanging WORKOUT300. This state is called "BACKGROUND" and can be accomplished by depressing the "BACKGROUND" labelled

softkey. Therefore, to prevent WORKOUT300 from hanging in an attempt to update an IDS that is elsewhere allocated, WORKOUT300 should be put into background prior to interrupting into the AMIGO environment.

When in background mode, the IDS displays an environment window (WORKOUT300 environment) and a softkey window. These softkeys are armed to allow the user to either enter foreground mode (SHOW STATUS), or to abort WORKOUT300 (ABORT).

4.2 WORKOUT300 Log

Each task logs into a memory file whenever it calls an intrinsic or returns from a call to an intrinsic. Only if an error is detected during the call is a permanent record put on disc. Otherwise the disc log file contains the log on and log off messages from the tasks and programs.

Whenever possible, recovery from a detected error is attempted. If in a single exerciser task, the total number of detected errors exceeds a certain maximum (currently 10), then the task will self-abort. If in a task an error occurs that is critical to the operation of the task (such as opening the device) then the task will again abort. If an error is detected in a task or in any of the control tasks making up WORKOUT300, and the call is critical to the operation of WORKOUT300 as a whole, then WORKOUT300 will abort and an error message will be TELLOPed to the console.

This message will contain the error condition (STATUS) and error message, name of the routine where the intrinsic call occurred, and the intrinsic called.

If a non-zero status is returned from the intrinsic, the parameter values used in the call will be written to the memory file. A WORKOUT300 control task reads this data from the memory file, and writes it to the disc log file.

The memory file will be available for output from user supplied programs operating under the WORKOUT300 system. The proper format for records written into this memory file is documented below in Section 5.2.

5.0 ADVANCED USAGE -- RUNNING SUBORDINATE PROGRAMS

The user has the option of having WORKOUT300 load and execute a user program via the program identifiers in the script. These programs are called subordinate programs.

5.1 Communication from WORKOUT300 to Subordinate Program

Each exercising function task and subordinate program TCREATEd and TSTARTed or PLOADed and PSTARTed by WORKOUT300 is assigned a unique key to be used by these tasks and programs for communication with the WORKOUT300 control program.

For user supplied programs, this key is passed to the program via the PARMS parameter of the PSTART command along with the other program parameters specified in the program identifier declaration. The program acquires this key and its other parameters by calling the system intrinsic BGETPARMS. For example, consider the following script:

```
      P14:=MYPROG(KOV)[INFILE(SCRATCH),5,,4]
   COLLECTION
      P14
```

When MYPROG ICALL's the system intrinsic BGETPARMS, a typical response would the following:

```
      "14,0,17,INFILE(SCRATCH),5,,4"
```

where the first three numbers represent the key passed to the user program by the WORKOUT300 control program. The key contains the program identifier, collection number (zero origin), and program ID number (and possibly a fourth number. See Section 5.2.5.)

5.2 Communication from the Subordinate Program to the Logger Control Task

As stated in Section 5.2 the key is used to identify the source of any message sent to the WORKOUT300 control program. It is the responsibility of the user subordinate program to send the following communication:

(1) LOG ON.
(2) LOG OFF
(3) ERROR REPORTS

All communication is sent to the memory file

```
      LOGFILE.MEMORY
```

which has already been created by the time the user program begins execution. The user must therefore open this memory file shared update by issuing a command like the following:

```
ICALL OPEN("LOGFILE.MEMORY",14,MEMFILE.FID,STATUS(), SHARE:
          "U")
```

All communication written to the memory log file LOGFILE.MEMORY is read by the logger control task. In order for the logger to act upon the communication, the communication must observe the following format (except for ERROR-CONTINUATION messages. See Section 5.2.8):

```
OPCODE,KEY,PARAMETERS
```

where

OPCODE — is a number that identifies the operation being requested

KEY — is the three tuple (or four tuple if subtask. See SUBTASK-LOGON, Sec. 5.2.5) that is passed to the user program via the PARMS parameter of the PSTART intrinsic (see above).

PARAMETERS — is a possibly empty list of additional items, the content of which depends upon the OPCODE.

All messages sent to LOGFILE.MEMORY that do not observe this format will be dumped into the disc log file. For all such "unformatted messages", CARE MUST BE TAKEN TO PREFIX THE MESSAGE WITH A NON-NUMERIC, NON-BLANK CHARACTER. Random OPCODES can have a very serious effect on the operation of WORKOUT300 as a whole.

The current logger control task operations are the following:

5.2.1 LOG-ON

```
OPCODE:      0
KEY:         three-tuple only
PARAMETERS:  None
```

A function task or subordinate program sends this message to the log memory file to notify WORKOUT300 that the function/program has begun execution.

EXAMPLE: "0,14,0,17"

5.2.2 LOG-OFF

OPCODE: 1
KEY: three-tuple only
PARAMETERS: None

A function task or subordinate program sends this message to notify WORKOUT300 that the function/program is about to terminate.

EXAMPLE: "1,14,0,17"

5.2.3 START-ICALL

OPCODE: 2
KEY: three-tuple or four-touple
PARAMETERS: INTRINSIC

The function task or subordinate program is about to ICALL a system intrinsic. This communication serves the purpose of telling WORKOUT300 which intrinsic is being called.

EXAMPLE: "2,14,0,17,GETN.1" for a three-tuple KEY
"2,14,0,23,17,GETN.1" for a four-tuple key

NOTE

The INTRINSIC parameter is treated as a string by WORKOUT300 and is stored temporarily until a STOP-ICALL or ERROR message is issued. It is not tested for content. The INTRINSIC parameter will be used when recording an error in the logger disc file. This parameter may therefore contain information useful for identifying which intrinsic caused the error. Hence GETN.1 may be used rather than just GETN in order to uniquely identify which GETN issued the message (assuming multiple GETN calls in the user subordinate program).

5.2.4 STOP-ICALL

OPCODE: 3
KEY: three-tuple or four-tuple
PARAMETERS: None

This message is paired with a START ICALL message and notifies WORKOUT300 that the intrinsic called has completed successfully (zero status returned).

EXAMPLE: "3,14,0,17" for a three-tuple KEY
"3,14,0,23,17" for a four-tuple KEY

5.2.5 SUBTASK-LOG-ON

OPCODE: 7
KEY: four-tuple only
PARAMETERS: None

A function task or subordinate program has TCREATEd and TSTARTed another task or PLOADed and PSTARTed another program. This subtask/subprogram logs on via the SUBTASK-LOG-ON message. For such a subtask, KEY is now a four-tuple: PROGRAM IDENTIFIER (of parent task/program), COLLECTION NUMBER, ID (of TSTARTed subtask or PSTARTed subprogram), PARENT ID (of calling task/program). For instance, suppose a task with KEY="14,0,17" TSTARTs a subtask with task ID 23. Then the subtask would adopt the KEY

14,0,23,17

and would use this KEY for all communication with WORKOUT300.

EXAMPLE: "7,14,0,23,17"

5.2.6 SUBTASK-LOG-OFF

OPCODE: 8
KEY: four-tuple only
PARAMETERS: None

A subtask or subprogram (see SUBTASK-LOG-ON, Section 5.2.5) sends the message to notify WORKOUT300 that it is about to terminate execution.

EXAMPLE: "8,14,0,23,17"

5.2.7 ERROR

OPCODE: 9
KEY: three-tuple or four-tuple
PARAMETERS: STATUS(0),STATUS(1),ICALL PARAMETERS

A system intrinsic (see START-ICALL, Sec. 5.2.3) returns with a non-zero status indicating an error. The task/program sends the error message to supply error report information about the ICALL and to have WORKOUT300 generate an error message in the WORKOUT300 disc log file. This message also notifies the WORKOUT300 of the termination of the intrinsic, making a STOP-ICALL message unnecessary; i.e., this message is also paired with a START-ICALL message, but only in the event of an error return from the intrinsic.

The ICALL PARAMETERS portion of the parameter list is a possibly empty list of the parameters used in the ICALL and is treated as a comment string by the logger-control task.

EXAMPLE: "9,14,0,17,2,-11011,PROGID:60" for a three-tuple KEY
"9,14,0,23,17,2,-11011,PROGID:60" for a four-tuple KEY

5.2.8 ERROR-CONTINUATION

OPCODE: 10
KEY: None
PARAMETERS: ICALL PARAMETERS

This message serves as a continuation of an ERROR message and is used when the ERROR message alone would otherwise exceed 254 characters. To use this message, the LOGFILE.MEMORY must not be FUNLOCKed between the ERROR message and the ERROR-CONTINUATION message nor between multiple ERROR-CONTINUATION messages.

This guarantees that the ERROR message and its continuation messages will appear contiguously in the LOGFILE.MEMORY file.

NOTE: The ERROR-CONTINUATION messages should not contain a KEY.

EXAMPLE: "9,14,0,17,24,-1000,WNDID:77,ROW:2,INVDEO:0"
"10,HALFB:1,BLINK:1"
"10,UNDLINE:0"

6.0 EXERCISER FUNCTIONS

Exerciser functions are procedures available to the user for the purpose of exercising certain peripherals. These are linked to task identifiers in the task declarations of the script. They are referenced in these task declarations by

TASKn

where n is from 1 to 4, inclusive.

The four functions are the following:

(1) Generalized Disc exerciser
(2) Printer exerciser
(3) IDS exerciser
(4) Generalized terminal I/O exerciser

6.1 Disc Exerciser Function Task: TASK1

The Disc Exerciser Function Task will exercise a 7902, 7906, 7910, 7920, or 7925 disc unit containing either the system disc or a standard volume. This is done by reading and writing from and to exerciser created work files. Verification of the contents written to a work file is also performed.

TASK1[DOMAIN,DEVICE_NAME,VOLUME_NAME,NUM_RECORDS,NUM_CYCLES]

Parameters:

DOMAIN — Specifies an existing domain name where the DEFT will create its work files. The default value is "SCRATCH".

DEVICE — Specifies the device where the above specified domain resides. (This name, associated with the disc device, can be found by issuing the command SHOW STATUS OF ALL DEVICES. Default is ""; i.e., the system disc.

VOLUME — Specifies the name of the volume where the above specified domain resides. Default is ""; i.e., the system volume. Note: If the VOLUME parameter is left to default, the DEFT will force the DEVICE parameter to default overriding any specified device.

NUM_RECORDS — Specifies the number of records initially deposited in the source file. This will be the number of records copied between files and the number of comparisons between files. Default value is 1000.

NUM_CYCLES — Specifies the number of read-write-read-read-compare cycles to perform. If negative, TASK1 will cycle indefinitely. Default is 1 (one cycle).

At any one time, there will be at most 3 files being managed by the TASK1: a source file, a destination file, and an intermediate file. If more than one cycle is specified, the role of each file, whether source, destination or intermediate will change with time. This will become clearer from what follows. Currently only the Device Management functions have been implemented.

The TASK1 performs the following:

INITIAL SETUP:

(1) Create a relative file in the designated or default domain. This file will be the source file.

(2) Fill up the source file with records of lengths varying between 1 and 254 bytes, inclusive. The number of records is specified in NUM_RECORDS.

(3) Create a second relative file in the designated or default domain. This file will be the destination file.

CYCLE STEPS:

(4) Copy records sequentially, one at a time, from the source file to the destination file.

(5) Read records from the source file and compare them with records read from the destination file.

If the specified number of cycles has been performed, then TASK1 purges all files and exits. Otherwise, the following next cycle setup steps are performed.

NEXT CYCLE SETUP:

(6) Create a third relative file on the designated or default domain to serve as the intermediate file.

(7) Purge the current source file

(8) The previous destination file now serves as the new source file, and the intermediate file now serves as the new destination file.

(9) Continue with step 4 above.

EXAMPLE:

```
TS: = TASK1[TESTDOM,,D5SYSTST,2500,10]
```

Here all work files will be created in the domain TESTDOM on the floppy volume D5SYSTST. The initial source file will be filled with 2500 records and the TASK1 will cycle 10 times.

EXAMPLE:

```
TASK1[ ]   is equivalent to

TASK1[SCRATCH,DISC7906,D6SYS,1000,1]
```

for a 7906 based system, with volume label D6SYS and device name DISC7906, or

```
TASK1[SCRATCH,DISC7910,D1SYS,1000,1]
```

for a 7910 based system with volume label D1SYS and device name DISC7910.

6.2 Printer Exerciser Function Task: TASK2

The Printer Exerciser Function Task will exercise any 2631 or 2608 printer attached to the HP 300. The exerciser will print a series of print tests on the printer that can easily be scanned for printer errors. Errors detected by the system intrinsics will be reported in the log file. For a 2631, one TASK2 cycle produces 12 pages of output, and take about 15 minutes to complete. For a 2608, one cycle produces 2 pages of output and takes a few minutes to complete.

```
TASK2[DEVICE_NAME,NUM_CYCLES,PRINT_WIDTH,SECONDARY]
```

Parameters:

DEVICE_NAME	Specifies the name associated with the printer. Default is "PRINTER".
NUM_CYCLES	Specifies the number of times the TASK2 will loop through the printer exerciser tests. If negative, TASK2 will cycle indefinitely. Default is 1.
PRINT_WIDTH	Specifies the maximum number of characters that can be printed per line while in normal print mode (10 cpi). Default is 136.
SECONDARY	Specifies whether a secondary character set exists and if so whether or not to exercise it. If the SECONDARY parameter is non-zero, then a secondary character set is assumed to exist and will be exercised using the same print tests as used for the primary. Default is zero.

NOTE: Multiple copies of the printer exercisor task can execute concurrently on the same printer only if the printer is SPOOLED. It is possible to write a script using this task which will execute correctly on a system with a spooled printer which will execute with errors on a system with the same printer un-spooled.

6.3 IDS Exerciser Function Task: TASK3

TASK3 performs several tests that exercise various IDS management intrinsics. These tests exercise alternate character sets (when specified), enhancement options, and all four window types.

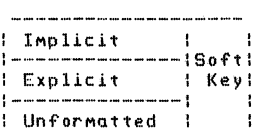
```
TASK3[DEVICE_NAME,ALTCHARSET]
```

Parameters:

DEVICE_NAME	Specifies the name associated with the IDS. Default is "CONSOLE".
ALTCHARSET	Specifies the alternate character sets to be tested. This parameter consists of a 1, 2, or 3 digit number (or omitted of no alternate character sets). If a "1" appears as any one of the digits in this number, then the Alt1 character set will be assumed to exist. Similarly for the digit "2" and "3". Therefore ALTCHARSET = 312 will be equivalent to 123, 213, 321, etc., and will exercise all three alternate character sets.

The first IDS exerciser test has the screen split into a softkey window and an Implicit window. The Implicit window is exercised by a separate subtask that writes a print pattern sequentially to the window, applying varying enhancement options to the lines in the window.

The second test splits the screen into a softkey window, an Implicit window, and Explicit window, and an unformatted window as shown below:



Each window is controlled by a different subtask and has a different alternate character set (if that option is specified) according to the following scheme:

Window Type	Character Set
I	Std.
E	Alt1
U	Alt2
S	Alt3

If any one of the alternate character sets is not specified in the ALTCHARSET parameter, then the Standard (Std.) character set will be used instead.

The Explicit Window also has eight protected fields.

6.4 Non-menu Terminal Exerciser Function Task: TASK4

TASK4 performs output and device control tests on any non-menu 2640B, 2644A, 2645A, 2648A, 2647A, 2649A, or 2621A terminal attached to the HP 300. In the case of the 2645A terminal, IO tests using the cartridge tapes can optionally be performed.

TASK4[DEVICE_NAME,TAPE,ALTCHARSET]

Parameters:

DEVICE_NAME — Specifies the name associated with the terminal. Default is "TERM1".

TAPE — (Only for terminals with tape cartridges)

- 0==> No tape cartridge drives tested (default)
- 1==> Left tape cartridge tested
- 2==> Right tape cartridge tested

ALTCHARSET — (Same as for TASK3)

The terminal exerciser performs a test similar to that performed by the printer exerciser: A block of asterisks are printed followed by a block of all printable characters. When alternate character sets are indicated in the ALTCHARSET parameter, one test is performed for each specified alternate character set.

After the printing tests, the tape tests are performed (when requested via the TAPE parameter). The one tape drive test performs the following:

(1) the LEFT tape is rewound, (2) ten records are written onto the LEFT, (3) the LEFT tape is rewound, (4) the ten records are read off of the LEFT tape and checked for errors, (5) the LEFT tape is rewound.

The two tape drive test performs the following:

(1) the LEFT and RIGHT tapes are rewound, (2) ten records are written onto the LEFT tape, (3) the LEFT tape is rewound, (4) the ten records are copied from the LEFT to its RIGHT tape, (5) both tapes are rewound, (6) the ten records are read from the right tape and checked for errors, (7) the RIGHT tape is rewound.

7.0 STANDARD SCRIPTS

There are two standard scripts that are shipped with the WORKOUT300 program: SCRIPT1, corresponding to a minimal configuration, and SCRIPT2, which exercises a larger configuration.

Due to the flexible nature of system configurability, these scripts are necessarily skeletal in form: some work must be done to the scripts and to the environment that WORKOUT300 will see.

Concerning the scripts, volume names and device names in the script need to be changed to correspond to actual names. These actual names can be obtained from the system build listing and/or by issuing one or both of the following AMIGO environment commands:

```
SHOW STATUS OF ALL VOLUMES

SHOW STATUS OF ALL DEVICES.
```

Furthermore, task identifiers in the script that correspond to system functions exercising non-existent devices should be deleted from the script.

All such modifications to the script can be accomplished in the WORKOUT300 Environment without having to resort to TYPIST.

TYPIST needs to be used only when additional lines need to be added to a script.

Concerning the environment, WORKOUT300 expects certain domains and volumes to exist for all disc exercising functions. These domains must either be created on their respective device/volume or the domain name in the identifer declarations in the script must be changed to an existing domain name on that device/volume. Similarly, volumes with the appropriate labels must be supplied or created, or the volume names must be altered in the script.

7.1 SCRIPT1 (Quick and Dirty -- 20 min.)

This script is found in SCRIPT1 on the WORKOUT300 Diskette. It exercises the following:

(1) The System disc (domain SCRATCH)

(2) The line printer (Name PRINTER)

(3) The IDS (device name CONSOLE)

The script is relatively short and should take fifteen to twenty minutes to complete. It is listed below:

```
     T10:=  TASK1[]
     T20:=  TASK2[PRINTER]
     T30:=  TASK3[CONSOLE]
COLLECTION
     T10
     T10,T10
COLLECTION
     T20,T30
     T30
```

If the printer is run spooled, the script will complete faster, but the printer will continue to print after WORKOUT300 is finished. If the printer is a 2608, the script will also run more quickly. To delete testing of the printer, use the cursor control and delete character keys to delete T20 from the second collection before hitting the START WORKOUT softkey. If the names of the printer or console have been changed, they can be edited before hitting the START WORKOUT softkey.

7.2 SCRIPT2 -- More extensive

This script is found in SCRIPT2 on the WORKOUT300 diskette, and adds a test of the flexdisc and a single non-menu mode terminal (device name TERM1) to the script SCRIPT1. To run this script, it is necessary to first prepare a formatted floppy as follows:

```
CREATE LABEL TEST ON DEVICE FLEXDISC
```

followed by

```
CREATE DOMAIN TEST7902 ON VOL D5TEST
CREATE DOMAIN TESU7902 ON VOL D5TEST
```

The script is listed below:

```
     T10:=TASK1[]
     T11:=TASK1[TEST7902,,D5TEST]
     T12:=TASK1[TESU7902,,D5TEST]
     T20:=TASK2[PRINTER]
     T30:=TASK3[CONSOLE]
     T40:=TASK4[TERM1]
COLLECTION
     T10
     T10,T10
COLLECTION
     T20,T30
     T30
COLLECTION
     T40
     T40
     T40
COLLECTION
     T11,T12
```

Note that since the flexible disc is a private volume, a private domain must be first created by the operator, and then it must be specified in the Declaration Block. If different names other than PRINTER, CONSOLE, or TERM1 are assigned to the devices, the script will have to be edited. This script could be modified to test other private volumes (i.e. a 7925) by simply specifying a different volume-name in place of D5TEST.

-hp-