Preface To The HP 3000/33
Diagnostic Manual Set

HEWLETT PACKARD

Manual Part No. 30070-90044
Diagnostic Set Part No. 30070-60068

Print Date: Sept 1978
Changed March 1979

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 or anyone's failure to comply with these requirements.

GROUND THE INSTRUMENT

To minimize shock hazard, the system chassis and cabinet must be connected to an electrical ground. The instrument is equipped with a three-conductor ac power cable. The power cable must either be plugged into an approved three-contact electrical outlet or used with a three-contact to two contact adapter with the grounding wire (green) firmly connected to an electrical ground (safety ground) at the power outlet. 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 fefinite safety hazard.

LEAVE COMPONENT AND POWER REPLACEMENT TO QUALIFIED PERSONNEL

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 exits; even with the power cable removed. To avoid injuries, always disconnect power and discharge circuits before touching them.

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 INSTRUMENT

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 death, are present in this system. Use extreme caution when handling, testing, and adjusting.

HP 3000/33 DIAGNOSTIC MANUAL SET ORGANIZATION

The HP 3000/33 Diagnostic Manual Set is a collection of stand-alone manuals that are organized in the following manner:

- System Self Tests
- Stand-Alone Diagnostic Languages
- System Hardware Diagnostics
- Peripheral Diagnostics

System Self Tests

This Section contains the Maintenance Interface Diagnostic Manual and the Cold Load Self Test Manual.

Stand-Alone Diagnostic Languages

This section contains the Diagnostic Utility System (DUS) Manual, the Advance I/O Diagnostic (AID) Manual., Slueth Simulator Manual, and the I/O Map Manual.

System Hardware Diagnostics

This section contains the Asyncronous Data Communication Channel (ADCC) Diagnostic Manual, the General Interface Channel (GIC) Manual, and the Memory Diagnostic Manual.

Peripheral Diagnostics

This section contains the 2631 Matrix Printer Exerciser Manual, the 7902 Flexible Disc Diagnostic Manual, the 7970 Magnetic Tape Diagnostic Manual, the 79XX/13037 Disc Controller Diagnostic Manual, and the 79XX Verifier Manual.

MANUAL CONTENT ORGANIZATION

Each HP 3000/33 Diagnostic Manual in this set is organized in the following manner:

Section 1 - General Information

Contains Hardware/software Requirements, Message and Prompt description, Diagnostic Limitions, and Mini-Operating instructions.

Section 2 - Operating Instructions

Contains details on all possible loading and operating options provided by the specific program being executed.

Section 3 - Execution Times

In most cases, this section will attempt to provide elapsed execution time for each of the test sections in a program. Note that diagnsotic programs are subdivided, logically, by test section and then step within test section.

Section 4 - Test Section Descriptions

Contains a description of each test section and each of the steps within each test section where the function, possible causes of error, and the error messages that can occur are provided.

Section 5 - Error/Action Summary

This section is primarily a quick reference summary of the possible error messages that can occur, their causes, and recommended action to be taken.

Further sections or appendices contain supplemental information that supports the above sections (i.e., glossary, extensive tables, etc.).

OCTAL TO HEXADECIMAL CONVERSION CHART

The following table is provided for your convenience. In the manual text, octal and hexidecimal codes are differentiated by the following notations:

! 08 (exclamation point is used for hex)
% 008 (percent sign is used for octal)

CHAR CODE			CHAR CODE			CHAR CODE			CHAR CODE			CHAR CODE			CHAR CODE		
Dec	Oct	Hex	Dec	Oct	Hex	Dec	Oct	Hex	Dec	Oct	Hex	Dec	Oct	Hex	Dec	Oct	Hex
0	000	00	48	060	30	96	140	60	144	220	90	192	300	C0			
1	001	01	49	061	31	97	141	61	145	221	91	193	301	C1			
2	002	02	50	062	32	98	142	62	146	222	92	194	302	C2			
3	003	03	51	063	33	99	143	63	147	223	93	195	303	C3			
4	004	04	52	064	34	100	144	64	148	224	94	196	304	C4			
5	005	05	53	065	35	101	145	65	149	225	95	197	305	C5			
6	006	06	54	066	36	102	146	66	150	226	96	198	306	C6			
7	007	07	55	067	37	103	147	67	151	227	97	199	307	C7			
8	010	08	56	070	38	104	150	68	152	230	98	200	310	C8			
9	011	09	57	071	39	105	151	69	153	231	99	201	311	C9			
10	012	0A	58	072	3A	106	152	6A	154	232	9A	202	312	CA			
11	013	0B	59	073	3B	107	153	6B	155	233	9B	203	313	CB			
12	014	0C	60	074	3C	108	154	6C	156	234	9C	204	314	CC			
13	015	0D	61	075	3D	109	155	6D	157	235	9D	205	315	CD			
14	016	0E	62	076	3E	110	156	6E	158	236	9E	206	316	CE			
15	017	0F	63	077	3F	111	157	6F	159	237	9F	207	317	CF			
16	020	10	64	100	40	112	160	70	160	240	A0	208	320	D0	240	360	F0
17	021	11	65	101	41	113	161	71	161	241	A1	209	321	D1	241	361	F1
18	022	12	66	102	42	114	162	72	162	242	A2	210	322	D2	242	362	F2
19	023	13	67	103	43	115	163	73	163	243	A3	211	323	D3	243	363	F3
20	024	14	6B	104	44	116	164	74	164	244	A4	212	324	D4	244	364	F4
21	025	15	69	105	45	117	165	75	165	245	A5	213	325	D5	245	365	F5
22	026	16	70	106	46	118	166	76	166	246	A6	214	326	D6	246	366	F6
23	027	17	71	107	47	119	167	77	167	247	A7	215	327	D7	247	367	F7
24	030	18	72	110	48	120	170	78	168	250	A8	216	330	D8	248	370	F8
25	031	19	73	111	49	121	171	79	169	251	A9	217	331	D9	249	371	F9
26	032	1A	74	112	4A	122	172	7A	170	252	AA	218	332	DA	250	372	FA
27	033	1B	75	113	4B	123	173	7B	171	253	AB	219	333	DB	251	373	FB
28	034	1C	76	114	4C	124	174	7C	172	254	AC	220	334	DC	252	374	FC
29	035	1D	77	115	4D	125	175	7D	173	255	AD	221	335	DD	253	375	FD
30	036	1E	78	116	4E	126	176	7E	174	256	AE	222	336	DE	254	376	FE
31	037	1F	79	117	4F	127	177	7F	175	257	AF	223	337	DF	255	377	FF
32	040	20	80	120	50	128	200	80	176	260	B0	224	340	E0			
33	041	21	81	121	51	129	201	81	177	261	B1	225	341	E1			
34	042	22	82	122	52	130	202	82	178	262	B2	226	342	E2			
35	043	23	83	123	53	131	203	83	179	263	B3	227	343	E3			
36	044	24	84	124	54	132	204	84	180	264	B4	228	344	E4			
37	045	25	85	125	55	133	205	85	181	265	B5	229	345	E5			
38	046	26	86	126	56	134	206	86	182	266	B6	230	346	E6			
39	047	27	87	127	57	135	207	87	183	267	B7	231	347	E7			
40	050	28	B8	130	58	136	210	BB	184	270	BB	232	350	E8			
41	051	29	89	131	59	137	211	B9	185	271	B9	233	351	E9			
42	052	2A	90	132	5A	138	212	8A	186	272	BA	234	352	EA			
43	053	28	91	133	5B	139	213	88	187	273	BB	235	353	EB			
44	054	2C	92	134	5C	140	214	8C	188	274	BC	236	354	EC			
45	055	2D	93	135	5D	141	215	8D	189	275	BD	237	355	ED			
46	056	2E	94	136	5E	142	216	8E	190	276	BE	238	356	EE			
47	057	2F	95	137	5F	143	217	8F	191	277	BF	239	357	EF			

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

Maintenance Interface Diagnostic

Maintenance Interface
Diagnostic Manual

HEWLETT PACKARD

Manual Part NO. 30070-90028

Print Date: Sept 1978
Change 1 March 1979

CONTENTS

GENERAL INFORMATION

SECTION I

1.0 INTRODUCTION

The Maintenance Interface (M.I.) Diagnostic is intended for use by the CE and manufacturing in testing the maintenance interface (M.I.). It is also available to the customer combined with the Cold Load Self Test diagnostic.

The M.I. Diagnostic includes a linker program which is automatically loaded into console memory at address !C000 and is then followed by the remote console program at !C100. The M.I. diangostic is then loaded at address !C800. Note that in this manual, the symbol "!" denotes hexadecimal. The M.I. Diagnostic is written in 8080 microprocessor assembly language.

When the diagnostic completes, it will issue a system reset and leave the CPU in micro run and program halt.

1.1 MAINTENANCE INTERFACE OVERVIEW

The M.I. is not a smart device and therefore its operation is under direct control of the system console and the CPU.

The M.I. may be viewed as a set of thirty 8-bit functional registers that are accessed through a central data path by either the system console or the CPU. All functions provided by the M.I. result from the manipulation of these addressable registers. Some functions require combinations of registers and some registers control a combination of functions. For further information on the M.I., refer to the HP **3000/33** Reference Training Manual, Section VII.

1.2 REQUIRED HARDWARE

- HP **3000/33** System Console
- HP **3000/33** minimum configuration with 128K bytes of memory (the CPU must be able to execute microinstructions).

1.3. REQUIRED SOFTWARE

The Maintenance Interface diagnostic must be properly recorded on a Terminal Data cartridge tape that can be accessed by the HP **3000/33** console.

1.4. MESSAGES AND PROMPTS

Four types of messages are output by the diagnostic: information, error, error communication, and prompt messages.

1.40 INFORMATION MESSAGES (i.e., title and end-of-pass) will be displayed with no program pause.

1.41 ERROR MESSAGES are used to inform the operator when the system responds unexpectedly to a given stimulus. Error messages will begin with "**ERROR DETECTED IN STEP xx" (where xx is the step number in which the error occured). Then the text of the error message is output. The following is an example of an error message:

```
**ERROR DETECTED IN STEP 20
  IMB DATA REGISTERS TEST FAILED.
  STATUS READ !A0     REG!10 WROTE!37     STATUS READ !A0
  REG!23 WROTE!10     REG!24 WROTE!11     REG!25 READ !27
  EXPECTED TO READ !23
```

Note that in the standard mode of operation (see Section 2) only the first 2 lines are output. In the optional mode of operation, the entire message is output. The error messages supply the following information:

1) The number of the step which failed.
2) A general description of the failure.
3) The register numbers involved in the test.
4) Data written to and read from the registers.
5) The expected data.

In the standard operating mode the diagnostic is set to pause if an error is detected. After the error message is output, the following message is also output:

```
CONTINUE (Y OR N)?
```

If "N" is entered, the diagnostic is aborted.

1.42 COMMUNICATION ERROR MESSAGES are printed whenever communications with the M.I. fail. If this type of failure is intermittent, a communication error could occur during any of the tests.

1.43 PROMPT MESSAGES are output whenever the program requires input from the operator (i.e., whether or not to suppress the pause after an error message is output). A complete explanation of prompt messages during program operation is covered in Section 2.

NOTE

For a complete explanation and summary of all error messages refer to Section 5 of this manual.

1.5 DIAGNOSTIC LIMITATIONS

No software can be run by the CPU while the M.I. diagnostic is being run.

The M.I. diagnostic can test the internals of the Maintenance Interface PCA with a high degree of accuracy. Note, however, the interfaces to the rest of the system cannot be fully tested. In particular, the diagnostic cannot test some failures which could cause the Maintenance Interface's IMB interface to interfere with normal IMB traffic. Also, the CPU interface is not tested at the speed of the CPU. Intermittent errors may not be detected.

The M.I./HP-IB interface is not tested. The primary reason for this is that the diagnostic does not have access to the HP-IB side of the interface, and, consequently, cannot tell when this interface is working.

Note that the M.I./HP-IB interface is utilized in the Cold Load Self-Test. Therefore, passage of this diagnostic and the failure of the Cold Load Self-Test may point to an M.I. failure.

1.6 MINI-OPERATING INSTRUCTIONS

```
+==============================================================+
| 1.  Perform an MPE SHUTDOWN.                                 |
|                                                              |
| 2.  Run the console Self Test.                               |
|                                                              |
| 3.  Ensure the system in in Micro run and Program run.       |
|                                                              |
| 4.  Fully reset the terminal via RESET TERMINAL key.         |
|                                                              |
| 5.  Set the REMOTE key to its up position.                   |
|                                                              |
| 6.  Insert M.I./Cold Load Self-Test cartridge and press READ |
|     when appropriate.                                        |
|                                                              |
| 7.  Answer STANDARD TEST (Y OR N)? appropriately.            |
+==============================================================+
```

2.0 INTRODUCTION

There are two modes of operation possible for the Maintenance Interface diagnostic; the standard mode and the optional mode. Following is the operating procedure for each mode.

2.1 STANDARD OPERATING MODE

The M.I. diagnostic is stored on a Terminal Data cartridge tape. To load the diagnostic into the system console, perform the following steps:

1. Perform an MPE SHUTDOWN to log everyone off the system in an orderly manner.

2. Run the console Self Test by pressing the TEST key on the keyboard. Verify the displayed output.

3. Fully reset the console by pressing the RESET TERMINAL key twice. An inverse video status line is displayed at the top of the screen. It should indicate a RUN or HALT condition.

4. Set the REMOTE key on the console keyboard to its up position so that the console will be in local mode.

5. Insert the maintenance interface diagnostic tape into the left cartridge console tape drive. The tape will automatically rewind to its beginning.

6. Depress the READ key on the console. The term LOADING is output immediately and then the following title message and question.

   ```
   MAINTENANCE INTERFACE DIAGNOSTIC   VERSION MX.XX
   STANDARD TEST (Y OR N)?
   ```

7. To cause the diagnostic to run once and halt on error (i.e., standard mode), enter "Y cr".

If no errors are detected, the diagnostic completes its operation in 60 seconds by outputting the following message to the console:

END OF DIAGNOSTIC - NO ERRORS DETECTED

If an error is detected during execution, an error message is output and the diagnostic completes by outputting the following message:

END OF DIAGNOSTIC - REPLACE THE MAINTENANCE INTERFACE

NOTE

The diagnostic can be stopped at any time by depressing any key and carriage return while the diagnostic is running. The diagnostic then prints the CONTINUE (Y OR N) question. Entering "N cr" causes the diagnostic to be aborted.

2.2 OPTIONAL OPERATING MODE

To enter the optional operating mode, perform steps 1 through 7 of the standard operating procedure. Then, to allow special options to be chosen, enter "N cr" in response to the STANDARD TEST question in step 7. As soon as "N cr" is entered, the following question is output to the console:

LOOP (Y OR N)?

The loop option causes the entire diagnostic to loop. When set to loop (i.e., you enter "Y" to the LOOP questions), the diagnostic will print "PASS xxx" (where xxx is the pass number) at the end of each loop through the diagnostic.

Once you have answered the LOOP question, the following question is immediately output:

SUPPRESS HALT (Y OR N)?

If you wish to eliminate the CONTINUE message and halt, enter "Y" to the SUPPRESS HALT message. Every time an error is detected, an error message is output (as described in Sections 1 and 5) and the message "CONTINUE (Y OR N)?".

Once you have answered the SUPPRESS HALT question the diagnostic begins its execution.

NOTE

The diagnostic can be stopped at any time by depressing any key while the diagnostic is running. The diagnostic then prints the CONTINUE (Y OR N) question. Entering "N" causes the diagnostic to be aborted.

2.3 M.I. STATE AFTER DIAGNOSTIC EXECUTION

Upon sucessful completion of the diagnostic, the state of the MI and CPU will be as follows:

- The CPU will be set to program halt and micro run.
- The CPU will be set on internal clock.
- IMB timeouts will be enabled.
- All breakpoints will be disabled; including IMB, ROM, and DSW breakpoints.
- The IMB interface on the Maintenance Interface PCA will be disabled.
- The front panel function register (!0D) on the M.I. PCA will be set to zero (0).
- The most significant byte of the CPU communications register (!16) will be set to !88 to indicate that the CPU communications is inactive.
- A system reset will be issued.

If errors are detected during execution, the diagnostic will try to set the M.I. PCA and the CPU to the state described above, but may be unseccessful.

WARNING

> If a full terminal reset is performed while the diagnostic is executing, the diagnostic is aborted and the state of the M.I. PCA and the CPU will be unknown. In this case, it is very likely that the M.I. PCA and CPU will be in a state which will prevent the system from performing a "cold load" or "warm start". If this occurs, cycle power on and off on the processor.

EXECUTION TIMES

SECTION III

The Maintenance Interface diagnostic takes less than 1 minute to run for a single pass.

TEST DESRIPTIONS

SECTION IV

4.0 INTRODUCTION

The following is a description of each test that is performed by the Maintenance Interface Diagnostic. Also included is the error message that will be output should an error occur in that particular step.

If an M.I. communication error occurs during any step, that message will also be output along with the error message.

Note that a summary of error messages, generated by each step, and the appropriate action to be taken can be found in Section 5 of this manual.

TEST	DESCRIPTION
1	This test sends interface clear and device clear to the M.I. If the M.I. cannot be initialized in this manner, the following error message is output: INITIALIZE MI TEST FAILED Faults could be in the console, the HP-IB cable to the M.I., or in the M.I. itself. A communication error will also be printed. See Section 5. NOTE: This test will pass if the cable to M.I. is disconnected. It only checks to make sure the HP-IB to the M.I. is not "hung".
2	This test repeatedly reads status from the M.I. and examines the status for gross errors. The following error message is output to the console should the status be in error: READ STATUS TEST FAILED This error could be caused by failures in the M.I. PCA's HP-IB interface to the console (to I/O junction panel) or the HP-IB cable (from the I/O junction panel) to the M.I. could be disconnected. This can also be caused by a power supply failure in the mainframe. Sometimes the status cannot be read and sometimes in can be read. In this instance, the above message is output plus the following addition:

TEST	DESCRIPTION
	INTERMITTENT Somtimes the status is incorrect and other times it is correct. In this instance, the above message is output plus the following addition: BAD DATA READ
3	This test gets and releases control of the Maintenance Interface. Should an error occur during this step it usually indicates a failure in the M.I. HP-IB interface to the console. The error message is: GET/RELEASE MI TEST FAILED Also, if the diagnostic cannot get control or cannot release control of the M.I., one of the following messages may be appended to the above error message: GET MI FAILED CAN'T RELEASE MI
4	This test determines whether the special front panel console keys (numeric pad) are disabled or not. These console keys can be disabled via a switch on the front panel being set to NO. A NO condition discovered, causes the diagnostic to abort. The diagnostic is aborted in this instance because ther is a strong possibility that the MPE operating system is active (usually signified by this group of keys being diabled). The M.I. should not run simultaneously with MPE as catestrophic errors can occur. TESTS ABORTED - CONSOLE KEYS ARE DISABLED
5	This test first disables the breakpoints. Then the test checks to make sure the run/halt flip-flop, in the CPU, can be set to micro run and micro halt. It also checks the status from the M.I. to verify that the HALT and MICROHALT bits are accurate. When the CPU cannot be set to micro halt, micro run, program halt, or program run, the following error message is output to the system console: MICROHALT-PROGRAM HALT TEST FAILED The status displayed along with this message will indicate exactly which function cannot be performed.

TEST	DESCRIPTION
6	This test writes a unique pattern to most of the M.I. registers and then reads the values back while comparing the data written to the data read. When the registers cannot be correctly set, the following error message is output to the console: REGISTER INITIALIZATION TEST FAILED The register last accessed is the register which could not be initialized. The value expected to be read will also be printed. The problem could be in the register last accessed or in the addressing logic on the M.I. PCA.
7-10	These tests write patterns to most of the M.I. registers and then they read from the registers. Step 7 tests registers !0 to !7. Step 8 tests registers !8 to !F. Step 9 tests registers !10 to !17. Step 10 tests registers !18 to !1F. The following message inidcates that the last register accessed did not respond as expected. The value read and expected indicates which bit of the register responded improperly: REGISTER PATTERN TEST FAILED
11	This test exercises the controls of the IMB such as priority out and bus request, as well as all the address, data, and handshake drivers and receivers. If an error occurs during the exercise, the followng message is output: IMB INTERFACE TEST FAILED Additional messages will be printed to more fully describe the error. These messages are as follows: NO BUS ACK. Bus acknowledge was not received. This tends to indicate that the IMB is being held by some other device.

TEST	DESCRIPTION
	IMB IN USE
	Some other device is using the IMB. This tends to indicate that some defective device is holding the IMB.
	DATA DRIVERS
	It is possible the IMB data drivers (registers !0E and !0F) are not operating properly. Maybe that drivers cannot be enabled by ENDAT in register !18.
	ADDRESS DOIT OR WAIT
	One or more of the signals enabled by ENDO in register !18 are not working. The signals enabled by ENDO are the address and opcode lines plus address DO-IT (ADO), DATA DO-IT (DDO), and WAIT.
	DONE OR PARITY
	One or more of the signals enabled by ENDN in register !18 cannot be set on the IMB. These signals are ADN, DDN, DNV, and IMBPER.
	AUTOPRO FAILED
	Automatic priority out failed.
	AUTOTRM FAILED
	The automatic termination of an IMB transfer failed.
12	In this test patterns are written to the CPU communications register (!16) and then the status is checked to verify that the MSGIN and MSGOUT bits of status are working.
	When an error occurs in this process, the following message is output to the console:
	STATUS MESSAGE BITS TEST FAILED
	The error message will be accompanied by a dump of the last values read from and written to the M.I. The last operation will be a read from the M.I. which returned an unexpected value. The value expected will also be displayed.

TEST	DESCRIPTION
13	The M.I. will either not issue a system reset or will not respond to a system reset if the following message is output. SYSTEM RESET TEST FAILED
14	This test checks the half step function and verifies that the CPU clock can be frozen in both the high and low state. The following message is output should an error occur: CPU MICROSTEP TEST FAILED
15	This test checks the ROM instruction register drivers and recievers on the M.I. The following message is output in case of error: RIR TEST FAILED
16	In this test, the M.I. forces the CPU to execute jump long microinstructions to several addresses and then verifies that the M.I. reads the address correctly. The following message is output in case of error: RAR TEST FAILED One of the following messages is also output as further description of the failure: PRESTROBE FAILED The prestrobe function of registers !3 and !4 on the M.I. is not working correctly. NO STEP The CPU did not execute the jump long microinstruction. TOO MANY STEPS The CPU executed many instructions when it was requested to perform only one. RECEIVER FAILED The CPU executed the jump long microinstruction but the M.I. cannot correctly read the RAR.

TEST	DESCRIPTION
17	This test forces the M.I. to request a cold load. The the CPU is forced to read external register 1 to see if a cold load message got to the CPU. A failure in this test causes the following message to be. output to the console: LOAD TEST FAILED Possible causes are hardware failure in M.I., backplane, system front panel and its cables, or the CPU. No cold load is actually performed.
18	This test is identical to the LOAD test except that a dump is requested. As with the LOAD test, failues of this test could be caused by hardware failures in the M.I., backplane, the system front panel or its cables, or the CPU. No dump is actually performed. The following message is output when an error occurs: DUMP TEST FAILED
19	This test forces the CPU to execute microinstructions to change the pause indicator. The error could indicate failures in the M.I.´s CPU interface or in the pause hardware on the M.I. The following message is output in case of error: PAUSE INDICATOR TEST FAILED The following messages are additional message that may occur to further explain the error condition: CAN NOT RESET The pause bit cannot be set to 0. CAN NOT SET The pause bit cannot be set to 1. CAN NOT TOGGLE The pause bit cannot be toggled from a 1 to a 0 or from a 0 to 1.

TEST	DESCRIPTION
20	This test exercises the DSW breakpoint logic and indicators. If an error is detected, the following message is output: DSW BP TEST FAILED One of the following messages may also be output for further explanation: DSW1-3 BAD One or more of the DSW indicator bits in M.I. register !1F does not respond properly. CAN NOT DISABLE A DSW breakpoint cannot be disabled by bit 2 in M.I. register !1D. CAN NOT ENABLE A DSW breakpoint cannot be enabled or the CPU fails to micro halt.
21	This test writes patterns into the M.I.´s ROM address register and also into the breakpoint compare registers. Then the rom breakpoint (ROMBP) indicator is checked to make sure that it indicates when the breakpoint address matches the ROM address and a ROM breakpoint is enabled. If an error is detected, then one of the following messages will be printed. ROM BP DETECT TEST FAILED CAN NOT DISABLE The ROM breakpoint indicator in M.I. register !1F is set even though the ROM breakpoint is disabled. CAN NOT ENABLE The ROM breakpoint indicator is not set as expected when the addresses are equal. This could be caused by the ROM breakpoint failing to be enabled.

TEST	DESCRIPTION
	EXTRA COMPARE
	The ROM breakpoint indicator indicates that the breakpoint address matches the ROM address when the addresses are different.
	NO COMPARE
	The ROM breakpoint indicator indicates that a breakpoint address does not match the ROM address when they do match.
22	This test verifies that when a ROM breakpoint is detected, the CPU is microhalted and when the ROM breakpoint is disabled, the CPU is not microhalted.
	The following message is output should this test fail:
	ROM BP HALT TEST FAILED
23	This test first gets control of the IMB. If this step is unseccessful, then either 'NO BUS ACK' or 'IMB IN USE' will be printed (see step 11). This test writes addresses onto the IMB and writes corresponding patterns into the breakpoint compare registers with IMB breakpoints enabled. Then the IMB breakpoint indicator, in register !1F, is checked to determine whether a breakpoint occurred. If the compare is bad, one of the following message is output to the console.
	IMB BP COMPARE TEST FAILED
	The breakpoint logic of the M.I. is not functioning properly. Replace the M.I.
	EXTRA COMPARE
	If this message is appended to the above message, a breakpoint occurred when the breakpoint compare address did not match the IMB address.
	NO COMPARE
	If this message is appended to the IMB BP COMPARE message, a breakpoint did not occur when the breakpoint compare address did match the IMB address.

TEST DESCRIPTION

24 This test checks several different functions of the IMB breakpoint logic of the M.I. The test gets control of the IMB and may print 'NO BUS ACK' or 'IMB IN USE' if it cannot get control (see step 11). If an error occurs after control of the IMB is accomplished, the following message is output to the system console.

IMB BP FUNCTIONS TEST FAILED

There is a problem with the M.I. breakpoint logic. Replace the M.I.

CAN NOT DISABLE

If this message is appended to the above message, an IMB breakpoint cannot be disabled.

CAN NOT ENABLE

If this message is appended to the IMB BP FUNCTIONS message, an IMB breakpoint cannot be enabled.

WRITE BP ON READ

If appended to IMB BP FUNCTIONS message, this indicates that when a write only breakpoint was set, a memory read operation triggered the breakpoint.

NO READ BP

If appended to IMB BP FUNCTIONS message, a breakpoint does not occur on a memory read operation when read/write breakpoints are enabled.

NO WRITE BP

If appended to IMB BP FUNCTIONS message, a breakpoint does not occur when a write breakpoint was set and a write operation was performed.

BP ON I/O

If appended to IMB BP FUNCTIONS message, a breakpoint occurs on an I/O operation when it should only occur for a memory operation.

TEST	DESCRIPTION
	ADO If appended to IMB BP FUNCTIONS message, a breakpoint occurs even if ADO is not set on the IMB.
25	This test checks the FREEZE bit in the M.I. register !0 to verifiy that a breakpoint will either set IRQ or microhalt the CPU. If the test cannot use the IMB, then 'NO BUS ACK' or 'IMB IN USE' will be printed on the system console (see step 11). If this test fails the following message is output to the console along with one of the other messages that follow. IMB BP HALT TEST FAILED Replace the M.I. NO HALT If this message is appended to the IMB BP HALT message, the breakpoint should have halted the CPU but did not. BAD IRQ If appended to the IMB BP HALT message, the breakpoint was set to microhalt the CPU but it also set IRQ on the IMB. NO IRQ If appended to the IMB BP HALT message, an IRQ on the IMB did not occur as expected. BAD HALT If appendedto the IMB BP HALT message, a microhalt occured when an IRQ should have been issured instead.

ERROR/ACTION SUMMARY

SECTION V

5.0 INTRODUCTION

Throughout this manual explanation for errors, their cause, and possible action to be taken have been interspersed with the appropriate test description. The purpose of this section is to summarize this information for easier reference.

Table 5-1. Error/Action Summary

MESSAGE	FAILURE	ACTION
COMMUNICATION ERROR |STEP=ALL|	Console cannot analyze the error. Possible console failure	Perform console self-test or replace M.I. HP-IB interface cable
CPU MICROSTEP TEST FAILED |STEP=14 |	Cannot freeze CPU clock high or low	Replace M.I. first If same error occurs perform CPU self-test or replace BIC PCA
DSW BP TEST FAILED DSW1-3 BAD CAN NOT DISABLE CAN NOT ENABLE |STEP=20 |	Breakpoint logic failed 1 or more DSW indicator bits in MI reg.!1F does not respond OK DSW breakpoint cannot be disabled DSW breakpoint cannot be enabled or CPU fails to microhalt	Replace M.I. or replace CPU
DUMP TEST FAILED |STEP=18 |	Request to dump failed to get to CPU	Replace M.I. first then check front panel, then verify CPU,and, finally, verify cables and IMB components
GET MI FAILED |STEP=ALL|	Console cannot gain control of MI	Replace MI
GET/RELEASE MI TEST FAILED |STEP=3 |	Cannot get or release control of MI	Replace MI or replace MI HP-IB interface cable

MESSAGE	FAILURE	ACTION
GET STATUS FAILED |STEP=ALL|	MI did not respond to status byte request from console	Replace MI
IMB BP COMPARE TEST FAILED EXTRA COMPARE NO COMPARE |STEP=23 |	M.I. breakpoint logic not working BP occurred and BP address did not compare with IMB address BP did not occur and BP address and IMB address do match.	Replace M.I.
IMB BP FUNCTIONS TEST FAILED CAN NOT DISABLE CAN NOT ENABLE WRITE BP ON READ NO READ BP NO WRITE BP BP ON I/O ADO |STEP=24 |	M.I. breakpoint logic in not working IMB BP cannot be disabled. IMB BP cannot be enabled. Memory read operation triggered write only breakpoint Read/write BP enabled does not cause BP to occur on Memory read Write BP enabled does not cause breakpoint during write operation Breakpoint should not occur in I/O operation Breakpoint occurs even if ADO is not set	Replace M.I.
IMP BP HALT TEST FAILED NO HALT BAD IRQ BAD HALT |STEP=25 |	M.I. FREEZE logic bad BP did not halt CPU BP set to microhalt CPU but it also set IRQ on IMB Microhalt occurred and only IRQ should've been issued	Replace M.I.

MESSAGE	FAILURE	ACTION
IMB INTERFACE TEST FAILED	IMB controls failed	
NO BUS ACK.	Bus acknowledge not recieved	Check for active device holding IMB
IMB IN USE	Some other device is using IMB	Check for defective device on IMB
DATA DRIVERS	Data drivers (!0E,!0F) not operating properly	Replace MI
ADDRESS DOIT OR WAIT	ENDO signals not working in reg. !18 of MI	Replace MI
DONE OR PARITY	ENDN signals not working in reg. !18 of MI	Replace MI
AUTOPRO FAILED	Automatic priority out failed	Replace MI
AUTOTRM FAILED STEP=11	Automatic termination of IMB transfer failed	Replace MI
INITIALIZE MI TEST FAILED STEP=1	Diagnostic cannot initialize M.I.	Run console self-test or check for faulty cable to M.I. or Replace it
LOAD TEST FAILED STEP=17	Request to load failed to get to CPU	Replace M.I. first then check for shorts on backplan then verify front panel and cables finally verify CPU
MICROHALT-PROGRAM HALT TEST FAILED STEP=5	CPU cannot be set to microhalt,microrun, program halt, or program run.	Replace M.I. first then check for possible bad CPU
NON-RESPONDING MI STEP=ALL	MI not accepting HP-IB commands from console	Replace M.I.

MESSAGE	FAILURE	ACTION
PAUSE INDICATOR TEST FAILED	Failure in MI's CPU interface or in pause hardware on M.I.	Replace M.I.
CAN NOT RESET	Pause cannot be set 0	
CAN NOT SET	Pause bit cannot be set to 1	
CAN NOT TOGGLE |STEP=19 |	Pause bit cannot be toggled	
RAR TEST FAILED	CPU jump long micro-instructions cannot be read by M.I.	
PRESTROBE FAILED	prestrobe function on M.I. not working	Replace M.I.
NO STEP	CPU didn't execute jump long microin-struction	Verify operation of CPU
TOO MANY STEPS	To many instructions executed by CPU	Verify CPU operat-ion
RECEIVER FAILED |STEP=16 |	M.I. cannot read in-struction correctly	Replace M.I.
READ FAILED |STEP=ALL|	M.I. didn't transfer data byte requested by console	Replace M.I.
READ STATUS TEST FAILED	Gross error detected when diagnostic reads M.I. status	Check HP-IB inter-face from M.I. to console or cable to M.I. is discon-nected. Secondly check for power failure
INTERMITTENT	Sometimes status can-not be read	
BAD DATA READ |STEP=2 |	Status read is some-times incorrect	

MESSAGE	FAILURE	ACTION
REGISTER INITIALIZATION TEST FAILED STEP=6	Data pattern written then read from M.I. registers does not compare	Replace M.I.
REGISTER PATTERN TEST FAILED STEP=7-10	Last register accessed did not respond as expected	Replace M.I.
RIR TEST FAILED STEP=15	ROM Instructions Register drivers and/or receivers on M.I. not working properly	Replace M.I.
ROM BP DETECT TEST FAILED	Breakpoint address (ROM) does not match ROM address and/or a ROM breakpoint not enabled	Replace M.I.
CAN NOT DISABLE	ROM Breakpoint indicator is set though ROM breakpoint is disabled	
CAN NOT ENABLE	ROM breakpoint indicator is not set and should be set	
EXTRA COMPARE	ROM indicator says breakpoint address matches ROM address when they´re different	
NO COMPARE STEP=21	ROM BP indicator says breakpoint addres does not match ROM address when they do match	
ROM BP HALT TEST FAILED STEP=22	CPU does not respond to breakpoint enable/disable	Replace M.I.

MESSAGE	FAILURE	ACTION
STATUS MESSAGE BITS TEST FAILED STEP=12	Status of CPU communications register(!16) has error in MSGIN and MSGOUT bits	Replace M.I.
SYSTEM RESET TEST FAILED STEP=13	M.I. will not respond to system reset nor issue a system reset	Replace M.I.
TESTS ABORTED - CONSOLE KEYS ARE DISABLED STEP=4	Possiblity that MPE is running due to console keys being disabled	Make sure that MPE is not running and enable console keys

GLOSSARY

SECTION IV

6.0 INTRODUCTION

The following terms and abbreviations are used in the manual.

TERM	MEANING IN THIS DOCUMENT
ADN	Address done. This is a handshake line on the IMB.
ADO	Address do-it. This is a handshake line on the IMB.
AUTOPRO	This is a capability of the M.I. which allows the M.I. to automatically set priority out when bus request is received from another device.
AUTOTRM	This is a capability of the M.I. which allows the M.I. to automatically terminate a transfer from the M.I. to another device over the IMB when the other device signals that it has received the data.
BRQ	Bus request. A handshake line on the IMB used by a device requesting access to the IMB.
CPU	Central Processor Unit. In this document it refers to the 2 board HP 3000/25 CPU.
DDN	Data Done. This is one of the handshake signals for data on the IMB.
DDO	Data do-it. This is one of the handshake signals for data on the IMB.
DNV	Data Not Valid. This is one of the handshake signals for data on the IMB.
DSW	Data Switch. This is an instruction which can be placed in the ABUS field of a CPU microinstruction to request that a microhalt occur. This microhalt is not performed by the CPU but is performed by the M.I. if a DSW breakpoint is enabled.
ENDAT	This is a bit in M.I. register !18 which enables the M.I.'s data drivers onto the IMB.
ENDN	This is a bit in the M.I. register !18 which enables the M.I.'s ADN,DDN,DNV, and IMBPER signals onto the IMB.

ENDO	This is a bit in M.I.register !18 which enables the M. I.'s address, ADO,DDO, and WAIT signals onto the IMB.
IMBPER	IMB parity error. This is a signal on the IMB indicating that a memory parity error has occurred.
IRQ	Interrupt request. This is a line on the IMB used by an interrupting device to notify the CPU.
M.I.	The Maintenance Interface PCA in the HP 3000/25.
RAR	The ROM Address Register in the CPU.
RIR	The ROM Instruction Register in the CPU.

Cold Load Self Test
Manual

Computer
Museum

HEWLETT hp PACKARD

Manual Part No. 30070-90017 Print Date: July 1979

PREFACE

CONTENTS

ILLUSTRATIONS

GENERAL INFORMATION

SECTION I

1.0 INTRODUCTION

The Cold Load Self-test program is designed to check the subset of the HP 3000/33 hardware that is used when a "cold load" operation is performed.

There are eleven (11) test sections of the Cold Load Self-test program. These test sections, briefly, are:

CPU Processor Board Tests

1. Test to set RAR to %10000

2. PCU Chip Test

3. RALU Chip Test

4. RASS Chip Test

System Module Testing

5. BIC (Bus Interface Controller) board test

6. ROM CRC (CPU - PCB) Chip Test (20 chips)
   ROM CRC (Firmware - PCB) Chip Test (8 chips)

7. Memory Test (128K bytes)

8. GIC Board Test

9. ID test of cold load device

10. Write/Read Loopback Test (device controller)

11. ADCC Board Test

Each of the eleven test sections can be broken down into several steps. When the program is executing any one of the sections, a message to that effect is displayed on the screen of the system console. When an error is detected, the sequence of testing pauses, and an error message is displayed on the screen. Program execution may be continued by typing GO and pressing return. To terminate execution, type "EX cr". This will clean all registers and flags, and restart microrun.

Testing will always run from the start of the entire program to the end of the program (if no errors are detected).

1.1 REQUIRED HARDWARE

The minimum system configuration assumed is shown in Figure 1-1. All tests require that this system configuration be present.

```
================================================================
                                         |----------|
                                         |   MAIN   |
                                         |  MEMORY  |
                                         |----------|
                                              |
IMB (Inter Module Bus)                        |
----------------------------------------------------------------
----------------------------------------------------------------
    |                  |      |          |
    |                  |      |          |
|--------|             |      |      |-------| General
|  ADCC  |             |      |      |  GIC  | I/O
|--------|             |      |      |-------| Channel
    |  Asyncronous     |      |          ||
    |  Data Communica- |      |          ||
    |  tion Channel    |      |          ||
    |                  |  |-------|      ||
    |     Central      |  |  CPU  |      ||
    |     Processing   |  |-------|      ||  HP-IB
    |     Unit         |      |          ||
    |               |-----------|        ||
    |               |Maintenance|        ||           7902
    |               |Interface  |        ||       |---------|
    |               |-----------|        ||-------|Cold Load|
    |                     |              ||       | Device  |
    |                     |              ||       |---------|
    |                 /---------|        ||
    |                /          |        ||
    |             * /   System  |        ||
    |           |---            |
    |-----------|     Console   |
                |---------------|
================================================================
```

Figure 1-1. System Architecture for Cold Load Self-Test

* The system console is always connected to ADCC channel 1 as device number 0.

WARNING

The Cold Load Self-test program assumes that the system console and the Maintenance Interface board are functional. Therefore, it is important that these modules be tested prior to execution of this program. The testing of these two modules is described in the following documents:

- The Reference Training Manual (Section on 2649 System Console)
- Maintenance Interface Diagnostic Manual

1.2 SOFTWARE REQUIREMENTS

The only software required is the Cold Load Self-test program, properly recorded on a Terminal Data Cartridge, that can be accessed by the HP 3000/33 system console.

1.3 MESSAGES AND PROMPTS

When the program is first loaded, it displays its title message and prompt (>). You are asked to enter "GO" at the system console. From this point on no further input is required. The program will then output either information messages pertaining to each test section, or, error messages, should they occur.

When an error message is output, the program pauses and then displays the following messages:

TO RESTART PUSH RESET TERMINAL

TO CONTINUE TYPE "GO"

TO EXIT TYPE "EX"

NOTE

Every execution should be completed by entering the "EX" command to avoid problems with the next process (i.e., Loading MPE, etc.).

1.4 DIAGNOSTIC LIMITATIONS

The following are not included in this diagnostic program:

- There is no Read Self Test.
- ROM parity checking is done in Test Section 6. If an error occurs, one message is output to indicate that a ROM chip is bad. Refer to manual Section 4 for more information.

1.5 MINI-OPERATING INSTRUCTIONS

```
+=============================================================+
| 1. Set COLD LOAD thumbwheels to cold load device you wish to  |
|    test - make sure they match physical device settings and   |
|    device number of cold load device is not 7.                |

| 2. Appropriately place the system in Micro Run/Progam halt.   |
|                                                               |
| 3. Ensure that the HP-IB test cable is properly connected to  |
|    I/O channel selected by the COLD LOAD switch.              |
|                                                               |
| 4. Place REMOTE key in LOCAL, CAPS LOCK key in down position. |
|                                                               |
| 5. Insert cartridge that contains Cold Load Self-Test and     |
|    press READ twice to execute just Cold Load Self-Test.      |
|                                                               |
| 6. Answer all question appropriately to begin execution.      |
+=============================================================+
```

2.0 INTRODUCTION

Before executing the Cold Load Self-test be sure that the system console and the Maintenance Interface module are functioning properly. To do this perform the self test for the console and execute the Maintenance Interface diagnostic. If either one of these modules are operating incorrectly, the Cold Load Self-test will not be valid.

2.1 COLD LOAD SELF-TEST LOADING PROCEDURE

1. If MPE is running, perform an MPE SHUTDOWN to properly logoff all current sessions.

2. Set the COLD LOAD thumbwheel switches to the Cold Load device you want to test. Refer to the following table for settings:

COLD LOAD DEVICE	CHANNEL	DEVICE CONTROLLER
7902	7	1
Magnetic Tape	-	-
system disc	6	1

NOTE: The front panel switches and the channel/device controller switches must match.

3. Place the REMOTE key in its up position and the CAPS LOCK key in its down position.

4. Fully reset the console by rapidly pressing the RESET TERMINAL key twice.

5. Insert the cartridge tape that has the Maintenance Interface diagnostic and the Cold Load Self-Test in the left slot (default) of the system console. If you are forced to use the right side make sure the console is set to read from this side (refer to the Reference Training manual for instructions).

6. Press the READ key on the console. The M.I. Diagnostic is loaded first. If you do not want to run the M.I. Diagnostic, press the READ key twice to only load the Cold Load self test. The following message will be output to the system console.

   LOADING COLD LOAD SELF TEST

   Then, after the program is loaded, the following instruction and prompt (>) are output:

   ```
   COLD LOAD SELF TEST
   VERSION x.xx - MM/DD/7Y
   TO START TEST TYPE "GO" RETURN
   >
   ```

7. Enter "GO cr" (where cr = RETURN key). The console will start the test. If anything besides "GO cr" is entered, the "TO START TEST TYPE GO" message is repeated.

   When testing is started, the message "COLD LOAD SELF TEST STARTED" is output to the console.

Once testing has started, no further action is required from the operator and all tests will be run. The console will update the screen to indicate which test is being performed at any given time. Refer to sections 4 and 5 for explanations of all messages.

When an error is detected, testing will pause and an error message will be displayed on the system console.

2.2 RESTARTING OR CONTINUING COLD LOAD SELF TEST

To restart the Cold Load Self-Test, press RESET TERMINAL one time only and follow the instructions described in 2.1.7. To continue execution after an error message has been displayed, type "GO cr" and the program will continue.

2.3. SYSTEM STATE AFTER COLD LOAD SELF TEST EXECUTION

The execution should have been completed by entering "EX". Then, whether an error occured or not, the Cold Load Self-Test leaves system hardware in a state that allows the loading and execution of either the MPE operating system or the Diagnostic Utility System.

EXECUTION TIMES

SECTION III

The Cold Load Self-Test program takes approximately 80 seconds from the beginning of testing to completion.

As each test section begins, the program outputs a message to the console which includes the approximate amount of time that that particular test section will take. Also, the time is included in the step messages output as each step is completed within a particular test section.

COLD LOAD SELF TEST

TEST DESCRIPTIONS

SECTION IV

4.0 INTRODUCTION

This section describes each test section and all steps associated with it. There will also be test data paths shown and expanded troubleshooting guides given where applicable.

4.1 TEST SECTION 1

This test verifies that the RAR can be set to the address %17777 (to ensure that all bits can be set) then that it can be set to address %10000 (where % means octal). Address %10000 is the starting address for the self test section of microcode. To set the RAR, the CPU processor board must be able to execute a JMPL microinstruction to location %10000. To accomplish this, the following steps are performed via the Maintenance Interface PCA.

NOTE

> The M.I. does not use any of the system busses for this test. The M.I. has direct control of the registers RIR and RAR.

1. The CPU is microhalted

2. A 32 bit microinstruction is entered into the external RIR of the CPU.

3. The CPU is then enabled to microstep.

4. After the microstep in step 3 (above), the CPU RAR (ROM address register) should be at %10000.

5. The RAR is then read and compared against the correct value.

If the RAR is NOT correct, the problem is most likely the CPU processor board or the BIC board. The error message is displayed as follows:

    RAR IN ERROR

Also output will be the value that was supposed to be set and what was actually read from the RAR.

If the test passes no message is displayed and testing continues.

4.2 TEST SECTIONS 2-7

These test sections are actually the subdivision of the hardwired CPU self test steps (10 through 64). The code for these test steps resides in ROM on the CPU processor PCA. The test steps must be executed via the following mode:

1. A ROM breakpoint is set on the M.I. This breakpoint is the address of the next sequential test to be executed (the first address being %10000).

2. The RAR (ROM Address Register) is set to the start of the test to be executed.

3. A microrun command is issued to the CPU via the M.I.

4. The self-test control portion goes into a wait loop for 1 or more seconds. The time is determined by the test being executed.

5. At the end of the WAIT loop the status of the CPU is examined by the M.I.

If the test was successful (test executed is test that RAR was set to), the CPU will be halted at the breakpoint that was set in step 1 above. If the test was NOT successful, then the CPU will be in one of the following states;

- The CPU will be halted at an error trap location.

- The CPU will be halted but not at the correct location.

- The CPU will not be halted at all. This is the condition where the CPU is executing some code, but there is no way of determining what that code is.

For example: Test 1 on the CPU processor board starts at ROM location %10013. The next sequential test will start at location %10055. The RAR is set to %10013 (start of test) and the ROM breakpoint is set to %10055 (start of the next sequential test). The following sequence of events would occur:

1. The CPU would be issued a command to microrun via the M.I.

2. The program would wait for 1 or more seconds depending on the execution time of the test.

3. The program would then monitor the status of the CPU and breakpoint registers via the M.I.

   If the CPU is halted at the correct breakpoint then a successful completion of the test which starts at location %10013 would be indicated.

4. The program would then set the ROM breakpoint to the address of the next test and repeat the process in steps 1 through 3.

If the CPU was in any of the error states, then a failure is indicated and an error message output to the console.

These error messages will indicate which step of a particular test section failed. The following paragraphs describe the steps within each test section, the possible error conditions, and the action to be taken.

4.20 Test Section 2 - PCU Testing

The following test steps verify the proper operation of the PCU portion of the CPU processor board.

STEP	DESCRIPTION
10	TAV, TBV and the STACK bit are verified for proper operation.
11	SKIP on immediate, DBUS, INDR, and LINK are verified.
12	AV, BV, SAVEA, SAVEB, JMP, JMPL, JSB, and RSB are verified for proper operation.
13	CIR, MAPPER, ATTN are verified for proper operation.

Should any of the above test steps fail, the following error message is output to the console:

PCU TEST#xx FAIL

where xx equals the step number. A failure in these steps indicates a problem with the CPU processoror its connector ; and in particular the PCU portion of this board. The action to be taken is presented below in the order of probable cause.

1. Swap in a new CPU processor board.
2. Swap in a New BIC board (Steps 10-13 also check some hardware on the BIC board).
3. Remove PCU chip, clean connector pins and return it.

4.21 Test Section 3 - RALU Testing

The following test steps verify the proper operation of the RALU portion of the CPU processor board.

STEP	DESCRIPTION
14	The RALU registers are checked for proper operation.
15	The RALU extended registers are checked for proper operation.
16	The ALU (arithmetic logic unit) is verified.
17	The arithmetic shifting logic is tested along with the linking logic.

NOTE

During test step 15, the extended address lines E4-E1 are exercised and may be observed with a logic probe to determine if they are still low or high.

Should any of the above steps fail, the following error message is output to the console:

RALU TEST#xx FAIL

where xx equals the step number. A failure in these steps indicates a problem with the CPU processor board; and in particular the RALU chip and associated logic. The action to be taken is presented below in the order of most probable cause.

1. Replace the entire CPU processor board
2. Replace the BIC board (Some hardware checking is done on the BIC board in these test steps).
3. Remove RALU chip, clean connector pins and return it.

4.22 Test Section 4 - RASS Testing

The following test steps verify proper operation of the RASS portion of the CPU processor board.

STEP	DESCRIPTION
20	The RASS Counter register and the Status register (bits 0 and 3) are verified for proper operation.
21	The Interrupt Status register (bits 10-13), the Bounds Violation logic, the Comparator, and Attention are checked for proper operation of the overflow and underflow function.

22	The Pre-adder logic and the Special Current Instruction Register (CIR) are verified.
23	The RASS registers are exercised and verfied.
24	The RASS Status register (bits 4-7) are verified for proper operation.

Should any of the above steps fail, the following error message is output to the console:

RASS TEST#xx FAIL

where xx equals the step number. A failure in these steps indicates a problem in the CPU proccessor board; and particularly the RASS chip logic. The action to be taken is presented below in the order of most probable cause.

1. Replace the CPU processor board.
2. Replace the BIC board (some hardware checking is done to the BIC board in the above tests).
3. Remove RASS chip, clean connector pins and return it.

4.23 Test Section 5 - BIC Testing

The following test steps verify proper operation of the BIC board.

STEP	DESCRIPTION
25	The Interrupt Status register (bits 2-6,8,9,14, and 15), the skip-on-test logic, the internal sync register, and Attention are verified for proper operation.
26	The CPU DOIT and DONE command logic, the time out logic, and the float state of the IMB are tested for proper operation.
27	The Freeze logic is tested for proper operation.

Should any of the above steps fail, the following error message is output to the console:

BIC TEST#xx FAIL

where xx equals the step number. A failure in these steps indicates a problem on the BIC board. The action to be taken is presented below in the order of most probable cause.

1. Replace the BIC board

2. Replace the CPU processor board (there is much interdependence between the BIC and the CPU processor)

3. Special action should be taken if step 26 fails because this is the first test to check the float state of the IMB (Inter-Module- Bus). To discover if an IMB data line is "stuck" at a low voltage level, perform the following procedure:

   a. Power down the system

   b. Pull all channel and memory boards out of the backplane

   c. Power up the system

   d. Load and execute the Cold Load Self Test again

If, after removing all channel and memory boards, step 26 passes, probably an IMB data driver on a channel or memory board is defective (shorted to ground). Re-install each board one at a time and run the Cold Load Self-Test each time you install a board until the test fails again. In this instance you have isolated a bad channel or memory board.

4. If step 26 still fails confer with your system specialist.

4.24 Test Section 6 - Control Store CRC Testing

The steps in this test section verifiy each ROM chip on the CPU processor and firmware boards (i.e., each parity code that is burned into a ROM chip is compared against the parity actually found).

Please note that one test step is executed per ROM chip.

When any step fails, one of the CRC test message lists the location of the faulty chip with the PCB name:

ROM CRC (CPU PROC-PCB) TEST #nn FAILED Chip Location Uxxx
ROM CRC (FIRMWARE-PCB) TEST #nnn FAILED Chip Location Uxxx

The following method may also be used to determine chip location:

1. Go to the rear of the system and open the door.

2. Locate the BIC PCA and the ten (10) LEDs near the top of the board.

3. The lowest LED (*), next to the CPU TEST switch should be lighted. The upper nine (9) LEDs should be displaying the test step number of the ROM that failed in octal. The first 9 LEDs are read from top to bottom. The table below lists the step number, the LED pattern in octal, and the ROM chip # being tested. It also tells the number of optional XXX PROM ROW(s) checked. This number should be at least 002 and possibly more if additional microcode is installed at some later date. Check the label on the stiffener bar of firmware board to see how many rows are installed.

Error*	ROM	Error*	ROM	Error*	ROM
		Processor Board			
30(300)	U-131	40(400)	U-133	50(500)	U-135
31(310)	U-141	41(410)	U-143	51(510)	U-145
32(320)	U-151	42(420)	U-153	52(520)	U-155
33(330)	U-161	43(430)	U-163	53(530)	U-165
34(340)	U-132	44(440)	U-134	54(540)	U-136
35(350)	U-142	45(450)	U-144	55(550)	U-146
36(360)	U-152	46(460)	U-154	56(560)	U-156
37(370)	U-162	47(470)	U-164	57(570)	U-166
		Firmware Board			
100(001)	U-23	104(041)	U-93	110(101)	U-24
101(011)	U-33	105(051)	U-103	111(111)	U-34
102(021)	U-73	106(061)	U-123	112(121)	U-74
103(031)	U-83	107(071)	U-133	113(131)	U-84
114(141)	U-94	120(201)	U-25	124(241)	U-95
115(151)	U-104	121(211)	U-35	125(251)	U-105
116(161)	U-124	122(221)	U-75	126(261)	U-125
117(171)	U-134	123(231)	U-85	127(271)	U-135
130(301)	U-27	134(341	U-97		
131(311)	U-37	135(351)	U-107		
132(321)	U-77	136(361)	U-127		
133(331)	U-87	137(371)	U-137		

* Notation is shown for contents of NIR and LED display.
Example: 100(011) = NIR(LED display), in octal.

4.25 Test Section 7 - Memory Testing

The following test steps verify the memory controller handshake along the IMB path to the NIR, and, the lower 128K bytes of memory (area of memory used by the Cold Load process).

STEP	DESCRIPTION
60	This is the first step to use the IMB handshake lines. The test is designed to read the status of the memory controller board; so there may be other faults on this board other than the handshake lines.
61	This test is designed to check all IMB data lines for array number 4 of the lower 128K bytes of memory (00,140000 to 00,177777).
62	This test is designed to check all IMB data lines for array number 3 of the lower 128K bytes of memory (00,100000 to 00,137777).
63	This test is designed to check all IMB data lines for array number 2 of the lower 128K bytes of memory (00,040000 to 00,077777).
64	This test is designed to check all IMB data lines and address lines for array number 1 of the lower lower 64K of memory (00,000000 to 00,037777).

If step 60 fails, the following message is output to the console:

MEMORY CONTROLLER FAIL

Since this is the first time the IMB handshake lines are used, there is a possibility that one or more of the lines may be stuck high or low. To gain more information, the following may be performed:

1. Replace Memory Controller board and restart test.
2. If test still fails, remove all channel boards and restart the Cold Load Self Test.
3. If the test passes, then a channel board is loading down the IMB handshake lines. Replace boards one at a time until bad board is discovered.
4. If the test fails with all channel boards removed, replace the BIC.
5. If the test still fails, then a handshake line is shorted in the IMB backplane. In this instance, confer with your system specialist.

Should steps 61, 62, 63, or 64 fail, the following error message is output to the console:

MEMORY TEST #xx FAILED

where xx equals the test step number. The Memory Array PCA #0 has possibly a bad bit. To determine this for sure, perform the following steps:

1. Replace the Memory Array PCA #0 with the highest #´d array and run test again.
2. If test still fails, replace the memory controller board.
3. If test fails again, remove all channel boards and run test again.
4. If the test passes, then a channel board is loading down the IMB handshake lines. Replace boards one at a time until bad board is discovered.
5. If the test fails with all channel boards removed, replace BIC and re-run test.
6. If the test still fails, then a handshake line is shorted and you should confer with your system specialist.

4.26 Testions 2-7 Complete Without Error

The following message is output to the system console if no failures occured in test sections 2 through 7.

CPU AND MEMORY TEST COMPLETE NO ERRORS

This message indicates that the following subset of hardware in the HP 3000/33 system is functional:

- CPU processor board
- Firmware board
- BIC board
- IMB backplane
- Memory Controller PCA
- Lower 64K words of memory

4.3 COMMON ERROR MESSAGES FOR TEST SECTIONS 1-7

In addition to the standard error messages that are printed, there are three error messages that might occur when executing test sections 1 through 7.

RAR IN ERROR

This indicates that the RAR cannot be set correctly. Refer to test section 1 for a more detailed explanation of this error.

ERROR - CPU WILL NOT HALT

This indicates that after the CPU was initially set to microrun, it never halted. The control program will wait for 8 seconds and then check the HALT status of the CPU. If the CPU is not halted then this error message will be displayed. Replace the CPU processor board.

CPU NOT HALTED AT BREAKPOINT

Output only for the first breakpoint set. This error indicates that the CPU is in the microhalt mode and it never reached the first breakpoint that was set.

If either of the last two error messages are encountered, then either the BIC board is defective or the CPU processor board is defective.

4.4 TEST SECTION 8 - GIC TESTING

In this test section, the channel/device number is read from the switch register and checked for their validity (i.e., channel # <> 0).

Error messages are issued when an illegal number occurs. This process is as follows:

This preliminary step reads the channel and device number from the switch register via the following sequence of events:

- Check the channel number. If OK pass control to step 65.
- If in error, the following message is output with a pause (error if channel = 0 or >15).

NO COLD LOAD CHANNEL OR EQUAL 0, Correct conditions and RESTART Test

The operator should fix the problem and restart the execution of the Cold Load Self-Test.

Then steps 65 through 73 are executed to verify all GIC control functions, data transfer, service requests and interrupts. These steps are described below:

STEP	DESCRIPTION
65	This test verifies the GIC configuration at the IMB through the following sequence of events: • Checks that bit 0 of GIC register 14 is reset and that bits 12 through 15 are set. If the test passes, control is transferred to step 66. • If an error occurs, the following message is output followed by a pause: TEST #65 FAILED - No GIC in Configuration
66	This test verifies data communication between the MI and GIC registers 3,4,5,7,8 (lower byte only) and 9 and 10 (upper and lower bytes) through the following sequence: • Writes all 1´s and then all 0´s to the selected registers. The registers are read after each write and compared with inputted values. If the test passes, control is transferred to step 67. • If an error occurs, the following message is output followed by a pause: TEST #66 FAILED- GIC Register is Bad
67	This test verifies DNV (Data Not Valid) through the following sequence: • Executes an ADD microinstruction to simulate DNV and checks for skip. If test passes, control is transferred to step 70.

STEP	DESCRIPTION
	• If an error occurs, the following message is output followed by a pause:
	TEST #67 FAILED - DNV Accepted as Data
70	This step tests CSRQ (Channel Service Request) after SIOP is issued to the GIC through the IMB, via the following sequence:
	• Executes SIOP from the stack and checks skip in RAR. If the test passes, control is transferred to step 71.
	• If an error occurs, the following message is output followed by a pause:
	TEST #70 FAILED - No CSRQ After SIOP
71	This step tests CSRQ from the PHI interrupt for all device numbers (0-7) via the following sequence of events:
	• Places channel numbers (which were read before step 65) into GIC register 3 and the device numbers into register 15. The an ADD microinstruction is executed to obtain CSRQ. If the test passes, control is transferred to step 72.
	• If an error occurs, the following message is output followed by a pause:
	TEST #71 FAILED - No CSRQ from PHI Interrupt
72	This test verifies operation of the GIC interrupt logic and IRQ (Interrupt Request) from every device number (0-7) through the following sequence of events:
	• Initializes selected GIC and executes a sequence of microcode instructions to obtain IRQ. If the test passes, control transferred to step 73.
	• If an error occurs, the following message is output followed by a pause:
	TEST #72 FAILED - No GIC Interrupt from COLD LOAD Device Channel

STEP	DESCRIPTION
73	This test verifies CSRQ from DMA circuitry, PHI, and FIFOs via the following sequence of events: • Writes 8 words into memory through the inbound FIFO and DMA circuitry. The words are read back to obtain CSRQ from DMA, PHI, and outbound FIFO. The results are compared to check proper data flow. If the test passes, control is transferred to ID test. • If an error occurs, the following message is output followed by a pause: TEST #73 FAILED - No CSRQ from DMA, PHI or IN/OUT FIFO

4.40 Additional Error Messages for GIC Testing

The following error conditions can occur throughout the GIC testing or as a result of operator error.

CAN´T GET CONTROL OF IMB FROM M/I

This indicates that the IMB did not respond to a request from the M.I. to have control. Any I/O board could be in control of the IMB.

To gain more information, remove all I/O boards except the GIC for the cold load device (7902 for HP 3000/33). If the error message still appears, after re-running the test, then the GIC, BIC, M.I., or Memory Controller could be defective.

NO HANDSHAKE BETWEEN IMB AND I/O BOARD

This message is an indication that the GIC never responded to a handshake request from the IMB. This usually happens when the front panel COLD LOAD channel number does not agree with the channel number that is set on the GIC.

If these two numbers do agree, try changing them to another set of equal numbers. But, be sure that you do not set the channel number to any other valid channel number currently in use in the system.

If this error still occurs, after running the test again, then either the GIC is defective or the front panel switches are defective or the M.I. could be defective.

NO HANDSHAKE BETWEEN IMB AND MEMORY

This message usually indicates that the handshake was never completed when the M.I. was reading the DMA data in main memory. It is usually an indication that the memory controller is defective.

4.5 TEST SECTION 9 - COLD LOAD DEVICE IDENTIFY

In this test the ID number is requested from the device that is indicated by the thumbwheel settings on the front panel COLD LOAD switches. Therefore, in order to test a particular cold load device, you must first set the COLD LOAD thumbwheel switches to the channel and device number of that device. See Figure 4-4.

The test actually looks at the settings and outputs the following message to let you know what it found:

COLD LOAD CHANNEL/DEVICE NUMBER = !7/1 (For 7902)

The test then requests the identification number of the indicated device and outputs the following message/result:

ID NUMBER OF COLD LOAD DEVICE = !0081 (indicates 7902)

It is up to you to evaluate whether the proper identification number was returned or not.

The program is not capable of determining whether the correct ID number was returned. However, when no ID number is read, the following message is output:

ID NUMBER OF COLD LOAD DEVICE = !FFFF <--NO RESPONSE

The execution of test Section 9 is complete when one of the following messages is output:

IDENTIFY WITH COLD LOAD COMPLETED-NO ERRORS

or,

IDENTIFY WITH COLD LOAD COMPLETED

The above message is output when no ID was read.

```
=================================================================
                                          |----------|
                                          |   MAIN   |
                                          |  MEMORY  |
                                          |----------|
                                               |
                                               |
IMB (Inter Module Bus)                         |
-----------------------------------------------------------------
-----------------------------------------------------------------
    |                      |      |
    |                      |      |
    |                      |      |
|--------|                 |      |      |-------|  General
|  ADCC  |                 |      |      |  GIC  |  I/O
|--------|                 |      |      |-------|  Channel
    |  Asyncronous         |      |         ||
    |  Data Communica-     |      |         ||
    |  tion Channel        |      |         ||
    |                      |  |-------|     ||
    |      Central         |  |  CPU  |     ||
    |      Processing      |  |-------|     ||   HP-IB
    |      Unit            |      |         ||
    |                      |      |         ||
    |                    |------------|     ||
    |                    |Maintenance |     ||
    |                    |Interface   |     ||        |---------|
    |                    |------------|     ||--------|Cold Load|
    |                         |             ||        | Device  |
    |                         |             ||        |---------|
    |                     /----------|      ||
    |                    /           |      ||
    |                 * /   System   |      ||
    |               |---             |
    |---------------|     Console    |
                    |----------------|
=================================================================
```

Figure 4-1. GIC Tests 65, 66

```
=================================================================
                                         |----------|
                                         |   MAIN   |
                                         |  MEMORY  |
                                         |----------|
                                              |
                                              |
IMB (Inter Module Bus)                        |
-----------------------------------------------------------------
-----------------------------------------------------------------
    |                   |      |          |
    |                   |      |          |
    |                   |      |          |
|--------|              |      |      |-------| General
|  ADCC  |              |      |      |  GIC  | I/O
|--------|              |      |      |-------| Channel
    |  Asyncronous      |      |          ||
    |  Data Communica-  |      |          ||
    |  tion Channel     |      |          ||
    |                   |  |-------|      ||
    |      Central      |  |  CPU  |      ||
    |      Processing   |  |-------|      ||   HP-IB
    |      Unit         |      |          ||
    |                   |      |          ||
    |                |-----------|        ||
    |                |Maintenance|        ||
    |                |Interface  |        ||        |---------|
    |                |-----------|        ||--------|Cold Load|
    |                      |              ||        | Device  |
    |                      |              ||        |---------|
    |                   /---------|       ||
    |                  /          |       ||
    |               * /   System  |       ||
    |            |---             |
    |------------|       Console  |
                 |----------------|
=================================================================
```

Figure 4-2. GIC Tests 67, 70, 71, 72

```
=================================================================
                                                     |----------|
                                                     |   MAIN   |
                                                     |  MEMORY  |
                                                     |----------|

IMB (Inter Module Bus)
-----------------------------------------------------------------
-----------------------------------------------------------------
    |                 |                        |
    |                 |                        |
    |                 |                        |
|--------|            |                    |-------| General
|  ADCC  |            |                    |  GIC  | I/O
|--------|            |                    |-------| Channel
    |  Asyncronous    |                        ||
    |  Data Communica-|                        ||
    |  tion Channel   |                        ||
    |                 |   |-------|            ||
    |      Central    |   |  CPU  |            ||
    |      Processing |   |-------|            ||  HP-IB
    |      Unit       |                        ||
    |                 |                        ||
    |               |-----------|              ||
    |               |Maintenance|              ||
    |               |Interface  |              ||        |---------|
    |               |-----------|              ||--------|Cold Load|
    |                                          ||        | Device  |
    |                    /----------|          ||        |---------|
    |                   /           |          ||
    |                * /   System   |          ||
    |              |---             |          ||
    |--------------|       Console  |
                   |----------------|
=================================================================
```

Figure 4-3. GIC Test 73

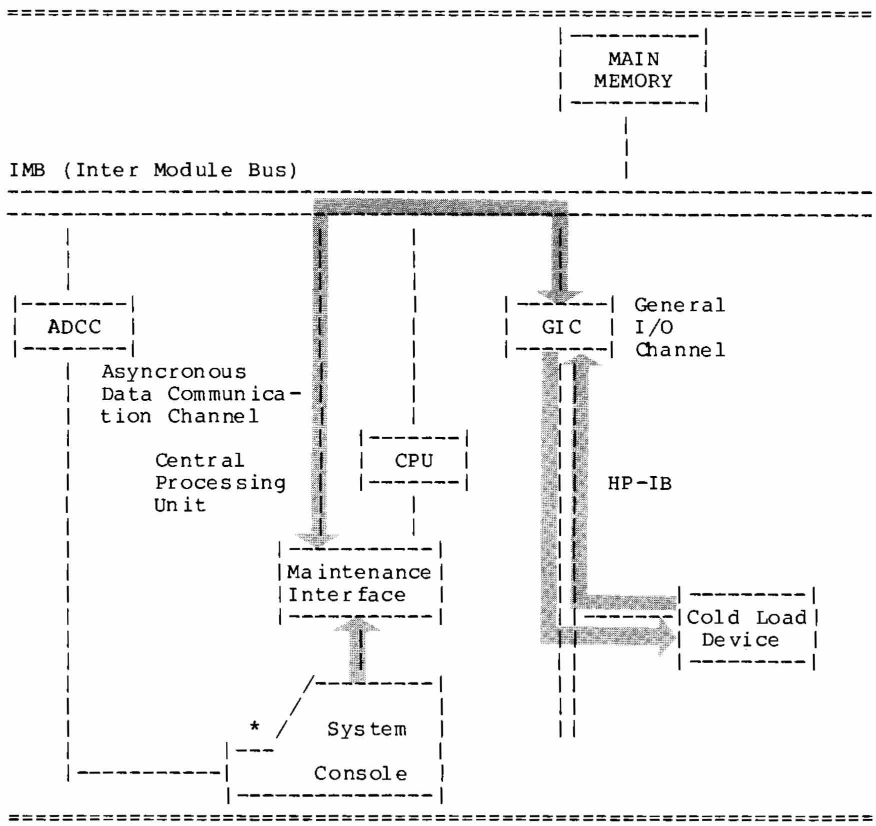

Figure 4-4. Test Cold Load Device Identity

4.6 TEST SECTION 10 - WRITE/READ LOOPBACK

This section verifies data transfer between a GIC and the controller of the Cold Load device by two instructions (WRITE LOOPBACK and READ LOOPBACK). The DMA function is assumed to be operating correctly when this test section is executed. This process is divided into 5 steps as described below and in Figure 4-5.

STEP	DESCRIPTION
1	The program creates the write buffer with 256 bytes at address !200 in the memory bank 0. The byte sequence is: !FF,!00,!01,!02,---!FD,!FE
2	The program creates the read buffer with 256 bytes equal to 0 at address !400 in the memory bank 0.
3	The program writes 256 bytes from write buffer into the controller of the Cold Load device via DMA and the GIC.
4	The program reads 129 or 256 bytes from the Cold Load device controller into read buffer in the memory via the GIC and DMA function. The program then sets the size of the read count buffer automatically depending on the ID number; 129 bytes for the magnetic tape, 256 bytes for all the other cold load controllers.
5	The program compares the 129 or the 256 bytes (magnetic tape or all other controllers, respectively) and issures one of two possible messages as shown below: WRITE/READ LOOPBACK TEST COMPLETED-NO ERRORS or, WRITE/READ LOOPBACK TEST FAILED in the instance of an error in the compare. Note that Test Section 10 takes 20 seconds to execute.

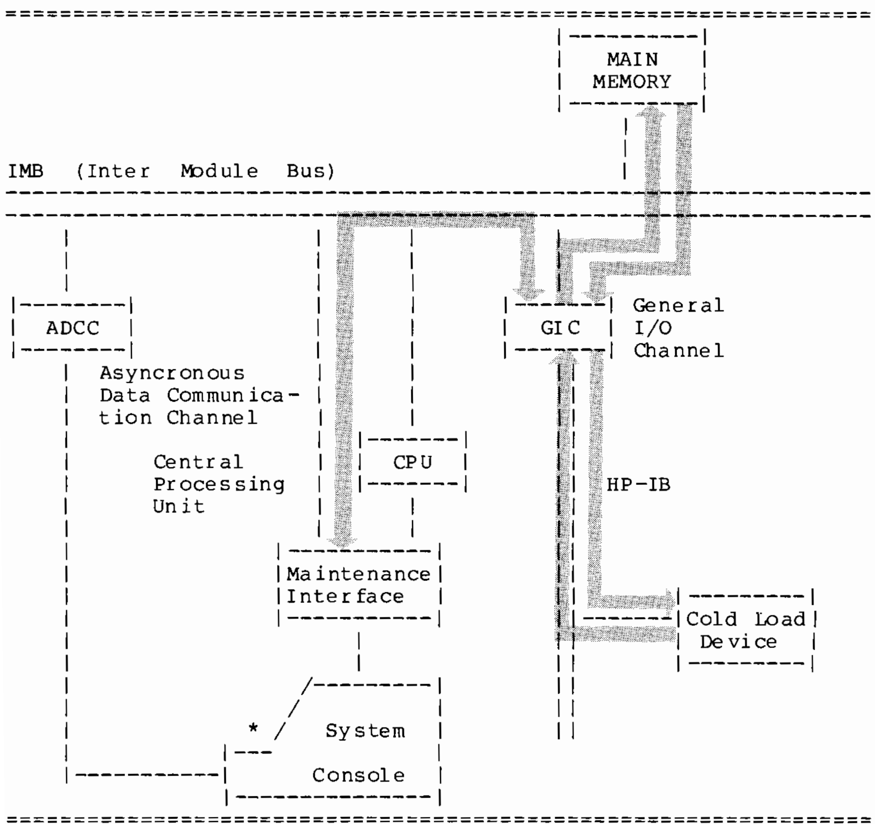

Figure 4-5. Test Write/Read Loopback

4.7 TEST SECTION 11 - ADCC TESTING

In this test section, data is transferred to and from the ADCC, to ensure proper data transfer between the console and the ADCC.

STEP	DESCRIPTION
1	This test step transmits data from the M.I., over the IMB, through the ADCC (channel 1, device 0) to the console. If the data is transferred correctly, the following message is output: ADCC TRANSMIT DATA PATH GOOD If the data is transferred incorrectly, one of the following messages is output to the console: NO HANDSHAKE BETWEEN IMB AND I/O BOARD This message output, indicates that the M.I. sent a handshake signal to the ADCC channel (number 1), and the ADCC did not respond. The ADCC is probably defective. Before replacing the ADCC board, verify that the channel number is set to 1. ADCC TEST FAILED (TRANSMIT DATA ERROR) This message indicates that the following data path is defective: ADCC-->Connector (on ADCC)-->cable (terminal ports) -->Connector (terminal ports)-->cable (console)--> connector (console) The first matter to verify is that all cable connections are correct. If this is true, and the test still fails, then the ADCC is most likely defective. However, any hardware in this path could be the source of the malfunction. ADCC TEST FAILED (NO TRANSMIT DATA) This error message indicates that the console never received any start bits from the ADCC. Usually the cause of this error is the cabling being defective. However, the ADCC may also be defective.

STEP	DESCRIPTION
5	This test step transmits data from the console, to the ADCC, over the IMB, to the M.I. to ensure proper transmission over this data path. If the data is transferred correctly, the following message is output: ADCC RECEIVE DATA PATH GOOD If the data transfer is incorrect, the following error message could result: ADCC TEST FAILED (RECEIVE DATA ERROR) One of two possible messages can be output at the end of this test section: ADCC TEST COMPLETE-NO ERRORS or, ADCC TEST COMPLETE when at least one error was encountered. When this error message is output the following messages are also output: TO RESTART PUSH RESET TERMINAL TO CONTINUE TYPE "GO" TO EXIT TYPE "EX" The operator should select one of these options and properly respond.

Upon succesful completion of self-test, the following message appears on the screen blinking:

COLD LOAD SELF TEST COMPLETE

ERROR/ACTION SUMMARY

SECTION V

5.0 INTRODUCTION

Throughout this manual explanation for errors, their cause, and possible action to be taken have been interspersed with the appropriate test description. The purpose of this section is to summarize this information for easier reference.

Table 5-1. Error/Action Summary

MESSAGE	FAILURE	ACTION
ADCC TEST FAILED (CONSOLE DEFECTIVE) TEST SEC 11- STEP 4	Failure to empty console buffer	console is defective
ADCC TEST FAILED (NO DATA TRANSMITTED) TEST SEC 11- STEP 4	Console never received any start bits from ADCC	Check out cabling or replace ADCC
ADCC TEST FAILED (RECEIVE DATA) TEST SEC 11- STEP 5	ADCC unable to receive data	Replace ADCC
ADCC TEST FAILED (TRANSMIT DATA ERROR) TEST SEC 11- STEP 4	Data path from ADCC is defective	Verify cable connections or replace ADCC
BIC TEST #XX FAIL |TEST SEC = 5|	BIC functions defective or the "float" state of IMB is stuck	Replace BIC PCA or replace CPU or verify IMB as in Section 4
CAN'T GET CONTROL OF IMB FROM M/I |TEST SEC = 8|	IMB didn't respond to request for control from M.I.	Verify if I/O boards are in of IMB. If not replace GIC then BIC, then M.I., then Memory Cont.
COLD LOAD CHANNEL CANNOT EQUAL 0 CORRECT CONDITION, RESTART |TEST SEC = 8|	Channel 0 is reserved for the BIC only	Verify fron panel COLD LOAD switch settings

MESSAGE	FAILURE	ACTION
COLD LOAD CHANNEL/ DEVICE NUMBER = !x/x |TEST SEC = 9|	Number output is the number found on the front panel COLD LOAD switches	You must verify correctness of number output, if incorrect check that switches on device and front panel match
CPU NOT HALTED AT BREAKPOINT |TEST SECs=1-7|	CPU in microhalt mode and never reached first breakpoint	Replace BIC then CPU
DMA PATH FROM M/I TO MEMORY ID DEFECTIVE |TEST SEC = 8|	GIC cannot do DMA transfer	Replace GIC or M.I.
ERROR - CPU WILL NOT HALT |TEST SECs=1-7|	Once CPU set to micro-run it won´t halt	Replace CPU
GIC TEST FAILED |STEP= 65-73 |	Transmit data path be-tween M.I. and GIC is defective	Verify the M.I. HP-IB port and M.I. are talking If not replace MI or Replace GIC or verify that I/O PCA´s are not holding IMB lines as shown in sec-tion 4 of manual
DMA TEST FAILED	Transmit data path be-tween GIC and memory is defective	Verify HP-IB and IMB port. If not replace first 64K memory
ID NUMBER OF COLD LOAD DEVICE = !xxxx |TEST SEC = 9|	Program outputs ID # after ID test to cold load device indicated in switch settings	Verify ID # is correct for device indicated in switch setting
MEMORY CONTROLLER FAIL |STEP=60 |	Memory Controller status incorrect	Verify IMB hand-shake line not holding IMB and /or replace Mem-ory controller

MESSAGE	FAILURE	ACTION
MEMORY TEST #XX FAILED |STEPS=61-64|	Bad bit in Memory Array board #0	Replace Memory or replace Memory Controller
NO HANDSHAKE BETWEEN IMB AND MEMORY |TEST SEC = 8|	Handshake not com- pleted when M.I. was reading DMA data in main memory	Replace Memory Controller PCA
PCU TEST #XX FAILED |STEPS 10-13|	CPU processor PCU failure	Replace CPU Replace BIC
RALU TEST #XX FAILED |STEPS=14-17|	CPU processor RALU chip failure	Replace CPU Replace BIC
RAR IN ERROR |STEP=1 |	RAR cannot be set correctly	Replace CPU Replace BIC
RASS TEST #XX FAILED |STEPS=20-24|	CPU processor RASS chip failure	Replace CPU Replace BIC
ROM CRC TEST #30-137 FAILED Chip Location U23-U164 STEPS=30-137	A ROM chip in the CPU or in firmware has failed	Replace ROM Chip being tested by that step
ID NUMBER OF COLD LOAD DEVICE = !FFFF-NO RESPOND	No ID number was read	Check HP-IB cable ,proper set of CHA/DEV #'s on device and switch register. Restart
TEST ABORTED	Any eror in Sec. 1-7	Fix CPU, Restart
WRITE READ LOOPBACK TEST FAILED	Controller of cold load device failed.	Check HP-IB cable cha/dev number replace cntrller of cold load dev

MESSAGE	FAILURE	ACTION
TEST #65 FAILED - NO GIC IN CONFIGURATION |TEST SEC = 8|	No response from any GIC	Check the configuration and set proper channel number
TEST #66 FAILED - GIC REGISTER IS BAD |TEST SEC = 8|	At least one of the GIC registers 3,4,5, 7,8,9,10 failed	Replace GIC
TEST #67 FAILED - DNV ACCEPTED AS DATA |TEST SEC = 8|	PHI failed	Replace GIC
TEST #70 FAILED - NO CSRQ FROM SIOP |TEST SEC = 8|	PHI failed, no CSRQ from GIC	Replace GIC
TEST #71 FAILED - NO CSRQ FROM PHI INTERRUPT |TEST SEC = 8|	PHI failed, no interrupt from GIC	Replace GIC
TEST #72 FAILED - NO GIC INTERRUPT FROM COLD LOAD DEVICE CHANNEL |TEST SEC = 8|	No response from GIC	Replace GIC
TEST #73 FAILED - NO CSRQ FROM DMA, PHI, OR IN/OUT FIFO'S |TEST SEC = 8|	At least one CSRQ failed to be received from DMA, PHI, or inbound or outbound FIFOs	Replace GIC

GLOSSARY

SECTION VI

6.0 INTRODUCTION

The following terms and abbreviations are used in this manual.

TERM	MEANING IN THIS DOCUMENT
ADCC	Asyncronous Data Communications Controller which is the link between the CRT terminals and the system.
BIC	Bus Interface Controller which is one of two boards that make up the CPU (Central Processing Unit).
CPU	Central Processing Unit comprised of two boards located in the first card cage.
CRC	Cyclic Redundency Check that insures all ROM locations contain the correct information.
FIFO	First In/First Out register buffer of the Mainten- ance Interface board used for data transfers up to a maximum of 40 bytes.
GIC	General Input/output Channel which is the link be- tween the peripheral devices and the system.
IMB	Inter-Module-Bus; The central data and control path between the CPU, the memory, and the channels. It is an asynchronous handshake-controlled bus.
M.I.	Maintenance Interface board, a non-intellegent de- vice, provides the interconnection between the HP 3000/33 and system console subsystem in order to provide maintenance panel and system self-test ca- pabilities. Refer to the HP 3000/33 Reference Train- ing Manual for further explanation.
PCU	Processor Control Unit is a large scale integration (LSI) chip in the CPU.
RALU	Register and Arithmetic-Logic Unit is a large scale integration (LSI) chip in the CPU.
RAR	ROM Address Register

RASS Register Address-Skip-Special is a large scale integration (LSI) chip in the CPU.

RIR ROM Instruction Register

SP5 Scratch Pad register number 5 located in the CPU.

Diagnostic/Utility System

Computer Museum

Diagnostic/Utility System
Reference Manual

HEWLETT hp PACKARD

Manual Part NO. 30070-90043

Print Date: Sept 1978

Change 1 March 1979

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.

CONTENTS

ILLUSTRATIONS

Figure

GENERAL INFORMATION

SECTION I

1.0 INTRODUCTION

The Diagnostic/Utility System is a memory-resident means of running diagnostic and utility programs. The Stand Alone File Manager (hereafter referred to as FMGR) is a disc based software module forming the heart of the Diagnostic/Utility System. In addition to the FMGR, the Diagnostic/Utility System includes AID 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 FMGR gives you access to files (located on a disc) and enables you to modify, delete, add or create those files. Also, a disc-based directory allows interchange of file information with other discs.

It is assumed that the operator is familiar with the nomenclature used in describing Keyboard terminals and the Control Panel.

It is implied that all user inputs are terminated with ENTER, carriage return/line feed or similiar function on the console device.

1.1 HARDWARE REQUIREMENTS

The following hardware is required:

a. HP 300 or HP 3000/33 consisting of:

(1) Memory - 64K words minimum

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

HP 3000/33: Console or HP 264X terminal

(3) Disc - HP 7902 Flexible Disc Unit

(4) Printer- HP 2631 printer (optional)

Figure 1.1 provides a pictoral view of how the FMGR integrates with other program modules.

```
                                        MEMORY
   DISC DEVICE            .-------------------------------------------.
.---------------.         |                                           |
| Diagnostic    |<-->|    | .----------------------------.            |
|  -Utility     |    |    | | Enter Your Program Name    |<----.      |
|    Disc       |    |    | | :                          |     |      |
|               |    |    | `----------------------------'     |      |
|               |    |    | valid responses are:               |      |
|  .----------. |    |    |                                    |      |
|  | Directory| |    |    |  .--------------.                  |      |
|  |   and    | |    |    |  |AID type file |->Load and        |      |
|  |  Files   | |    |    |  `--------------'   execute AID    |      |
|  `----------' |    |    |                      program file-'       |
`---------------'    |    |                                    |      |
                     |    |  .---------------.                 |      |
  KEYBOARD           |    |  |SPL-II type file|->Load and      |      |
  CONSOLE            |    |  `---------------'   execute       |      |
.---------------.    |    |                      SPL-II        |      |
| Operator      |<-->|    |                       program      |      |
| Interaction   |    |    |                          file----'        |
`---------------'    |    |                                    |      |
                     |    |  .---------.                       |      |
                     |    |  |MANAGER  |-------->Execute DUS--'       |
                     |    |  `---------'                       |      |
  PRINTER +          |    |  .-------.                         |      |
.---------------.    |    |  |  AID  |--------->Execute AID---'       |
| Hardcopy      |<-->|    |  `-------'                                |
|    Output     |    |    |                                           |
`---------------'         `-------------------------------------------'
```

+ Optional

Figure 1.1 - Diagnostic/Utility System Structure

OPERATING INSTRUCTIONS

SECTION II

2.0 INTRODUCTION

This section covers the specific operating instructions required to load the Diagnostic Utility System (DUS) and to manipulate the file manager portion of the DUS.

2.1 LOADING THE SYSTEM

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

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

(5) Insert a Diagnostic/Utility Disc into the 7902 Flexible Disc Unit.

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

(7) Press HALT, then press COLD LOAD.

(8) 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 FMGR program.)

(HELP is an AID program that presents file and command information.)

2.2 RUNNING PROGRAMS

To execute an AID or SPL-II program file, enter the program name as follows:

\ image

```
  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 (type HELP for program information)
  :
```

\ FORMAT

2.3 USING THE FILE MANAGER

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

\ image 3

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

2.4 USING AID

If you wish to create, modify, or make changes to an AID program, you may do so by typing "AID". The resulting interaction is described in the AID Diagnostic Lanaguage Manual, located in this binder.

FILE STRUCTURES AND FORMATS

SECTION III

3.0 INTRODUCTION

The information in this section pertains to the file management structure and how diagnostic and utility program are classified.

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
SPLII	SPL-II program	DUS SAVE Command
DATA	data file	CREATE Command

The AID and SPL-II 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 SPL-II 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).

At file creation time each AID program file is classified as a DIAGNOSTIC, each SPL-II 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 Diagnotic/Utility Disc. In other words, 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 files AID, IDSBOOT*, DIREC and SCRATCH** are permanent files; they cannot be modified in any way.
- The files AID,IDSBOOT,DIREC and SCRATCH can all be read but only SCRATCH can be written.
- Files that are protected must be unprotected before alteration (See CHANGE Command).

* The IDSBOOT file is reserved for IDSBOOT cold-load data.

**The SCRATCH file is a 60 sector scratch area usable by anyone as an unprotected scratch file.

D.U.S. COMMANDS

SECTION IV

4.0 INTRODUCTION

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 which should be easily understood by the operator. However, should difficulty arise understanding an error message refer to the Error Messages Section.

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(S):
```
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 device type 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 a device is not available to the DUS. See LISTIO command.

EXAMPLE(S):

```
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]

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

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(S): 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(S): 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.

EXAMPLES(S): 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(S): Enter Command (LC for List Commands)
>LF

Stand Alone File Directory

File-name	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	SPLII	D	U	400	4	0	11	00.00	300	100	200
ABC	DATA	A	U	1280	4	0	16	00.00	0	0	0

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 execution
SPLII 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
SPLII 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.
SPLII 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)
SPLII 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 implies that a device is not available to the DUS.

EXAMPLES(S): 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(S): 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(S): 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, SPLII 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(S): Enter Command (LC for List Commands)
>SAVE DIAG, 01.02

ERROR INTERPRETATION

SECTION V

E R R O R   I N T E R P R E T A T I O N

5.0 INTRODUCTION

This manual section discusses the possible error conditions that may occur during Diagnostic/Utility System operatons.

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 octal 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 MESSAGE

Message	Meaning
Invalid Filename	The filename parameter did not meet the requirements of a valid filename (See FILENAMES Section).
Disc Failure!! Did not respond within 10 seconds	The system disc didn't complete a seek, read or write within a reasonable time (approx-imately 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 or an attempt is made to alter the files AID, DIREC, IDSBOOT or SCRATCH.
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.

NOTES

NOTES

AID Diagnostic Language
Reference Manual

HEWLETT hp PACKARD

Manual Part No. 30070-90042 Print Date: Nov.1978

CONTENTS

CONTENTS (continued)

CONTENTS (continued)

Section 6 Page
OPERATORS

Section 7
RESERVED VARIABLES

CONTENTS (continued)

CONTENTS (continued)

ILLUSTRATIONS

GENERAL INFORMATION

SECTION I

1.0 INTRODUCTION

AID 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 128K 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	Must be pressed after every command and or statement. It terminates the line and ENTER 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 LOADING THE AID DIAGNOSTIC PROGRAM

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

(2) Enter 'AID'

(3) AID will display its title message and prompt.

1.4 AID COMMANDS AND STATEMENTS OVERVIEW

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 concelled 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
  ----------
> 40 (statement 30 is now PRINT S)
----
```

To delete a statement use the following format:

```
> 100   DELETE 30
-----
```

1.5 AID PROGRAMMING STRUCTURES

Any statement or group of statements constitutes a program. The following 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 AN AID 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 EXECUTING 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 results:

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

See the RUN Command (Section 3) 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:

Enter and run the following program:

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

Erase the program as follows:

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

The user's resident program area is now cleared and another program 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 by using the insert line function:

```
> 40   5. THIS PROGRAM AVERAGES
----
> 40   7.  5 NUMBERS
----
> 40   10 INPUT A,B,C,D,E   .GET
       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:

```
> 40 LIST
----
 5   . THIS PROGRAM AVERAGES
---------------------------
 7   . 5 NUMBERS
--------------
10    INPUT A,B,C,D,E   .GET 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) or the period (.) Special Character (SECTION 5) for further details.

1.10 AID OPERATOR MODE STATE DIAGRAM

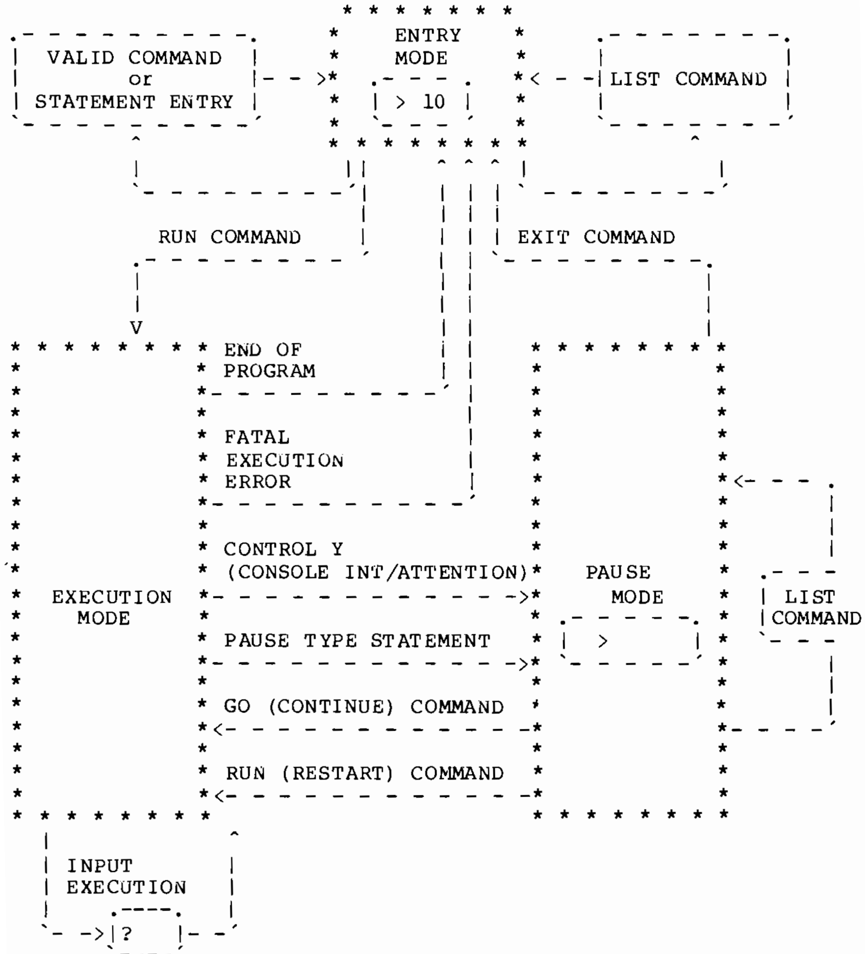

Figure 1.1 -- AID Operator Mode State Diagram

ESSENTIALS OF AID

SECTION II

2.0 INTRODUCTION

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 var-
      iable A)
-!16  (or !FFEA)
```

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:

```
.-----------------------.
|           |           |
|  BYTE 0   |  BYTE 1   |
|           |           |
`-----------------------'
```

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 CHARACTER
----
> 60   LET &AA(0):=&AA(1):="B"   .MULTIPLE STRING ASSIGNMENTS
----
> 60   LET &AA(2,5):=&BB(7,10):="ABCD"    .NOT ALLOWED
----
```

2.6 OPERATORS (OVERVIEW)

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.

2.7 RESERVED VARIABLES (OVERVIEW)

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.
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+(char.lngth/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+(char.lngth/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 statement usage 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 statement usage saves six words:

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

The following statement 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 statement usage 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
----
 .
 .
 .
```

The above can be condensed into the following single statment:

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

  The above statements will execute much faster than the following:

```
> 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.

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 follows:

```
> 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
-----
> 60  SECTION 4,300
----
```

or bet

```
> 10  GOTO 50
-----
       .
       .
> 50  SECTION 4,300  .BEGIN XYZ TEST
----
```

AID COMMANDS

SECTION III

3.0 INTRODUCTION

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.

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
----          ten)
```

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.

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.

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.

Entry	Meaning
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
                     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
        -----
```

Examples (continued)

```
> 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 ten (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
> 40  LIST    DOESN'T 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 AID statement, SECTION, for further examples and explanations.

AID STATEMENTS (NON I/O)

SECTION IV

4.0 INTRODUCTION

The AID statements available to the operator are listed, in detail 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 in data buffer (element+N-1). If repeat factor is included in the data pattern is repeated repeated factor times. Data1 through dataN must be numeric constants.

EXAMPLES:

```
> 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
                     parameter)
    .
    .
    .

 ---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, you must be 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 (.) inside a string). Comments should be kept short and used sparingly since they can only be used as source data thus consume a lot of user data storage 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 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 buffer where the entire buffer is stored with that numeric or variable. 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   (SUBSEQUENT ERRORS WILL NOW BE
-----            REPORTED DURING 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 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 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 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 program to enter a subroutine and then return to the next sequential statement* after 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 THEN
  -----
                    .RETURN CONTROL TO THE STATEMENT
                    .FOLLOWING THE GOSUB WHICH CAUSED
                    .TRANSFER OF CONTROL TO 500.
```

*See 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 50
  ----
```

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 true (non-zero)* to transfer control to 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 results 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 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 string buffer, the value of any variable, numeric, expression, 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 lable 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 console.

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. need not be checked). The "checks" condition is always enabled until this statement is encountered and then no checks are done until execution is completed.

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 an AID user program. This statement is suppressed only by the SNPS command. After a prompt (>) 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 pause may be suppressed by SNPS. The optional (*) will suppress pause which follows 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 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 variables specified (variabl1 to variableN). If an 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 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 buf-
                         fer 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 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 to execute a particular section. The executable sections are predefined by the TEST command and/or by assigning values to the Reserved Variable 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 number 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 manual 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 zeros)

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%0076570048 39
```

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

SPECIAL CHARACTERS

SECTION V

5.0 INTRODUCTIONS

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:	CNTRL H (Bs) or BACKSPACE
DESCRIPTION:	Allows the operator to backspace to the last character entered by pressing the CNTRL 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

```
CRT Example
-----------

> 10  LES
----    -
```

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

5.3 CONTROL X

OPERATION NAME: Delete Existing Line Input

SYMBOL: CNTRL 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 ON
----
> 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 (>) is printed in the 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 Designtion

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;
----

> 20  PRINT A;" DAYS"
----
> 30  PRINT "CALL " ;A
----
> 40  PRINT ";"
----
> 50  PRINT A;5;A;4;A,A;5;A
----
> 60  RUN
----
```

The results of the above statements are as follows:

```
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 (>) 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-linefeed in concatenated print strings (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
-
,
-
```

-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)
```

OPERATORS

SECTION VI

6.0 INTRODUCTION

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 ASSIGNMENT (:=)

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 INTEGER MULTIPLY (*)

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 INTEGER DIVIDE (?)

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 INTEGER ADD (+)

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 INTEGER SUBTRACT (–)

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 EQUAL (=)

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 NOT EQUAL TO (<>)

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 GREATER OR LESS THAN (< 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 LOGICAL 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 LOGICAL 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 THEN
----                   .TESTED FOR TRUTH (NON-ZERO)
```

6.12 EXCLUSIVE OR

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 MODULO OPERATION

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 LOGICAL SHIFT OPERATIONS

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).

```
                 .---------------.
     Bits        | All 16 bits   |
<----------------|               |--<--  0's In    LSL
     Lost        | shifted left  |
                 `---------------'

                 .----------------.
                 | All 16 bit     | Bits
0's In-->--------|                |-----------> LSR
                 | shifted right  | Lost
                 `----------------'
```

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 ARITHMETIC SHIFT OPERATIONS

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).

```
  Bits Lost
<------------------.
                   |
          .-----------------------.
          |     | 15 bits shifted  |  0's  IN     ASL
  Sign    |     |                  |<-------------
  Unchanged|    |      Left        |
          `-----------------------'

          .-----------------------.
          |     | 15 bits shifted  | Bits Lost
          |  *  |                  |-------------> ASR
          |     |      Right       |
          `-----------------------'
     * 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 CIRCULAR SHIFT OPERATIONS

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.

```
    .----------------.
    |   All 16 bits  |
 .<-|                |-<.          CSL
 |  |  shifted left  |  |
 |  `----------------'  |
 `-----------><----------'

    .----------------.
    |   All 16 bits  |
 .>-|                |->.          CSR
 |  |  shifted right |  |
 |  `----------------'  |
 `-----------<----------'
```

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.

SECTION VII

RESERVED VARIABLES

7.0 INTRODUCTION

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 program 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
```

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
      INDICATOR
----
> 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

```
 0  1  2  3  4  5  6  7  8  9       12  13   15
 ---------------------------------------------------.
|  | B|B  |NO|I  |>  |T  |D  |<  |  4 BIT   | 3 BIT  |
|  | A|A  |H |N  |   |O  |S  |   |  CHANNEL | DEVICE |
|  | D|D  |I |T  |   |   |   |   |          |        |
|  |PT|IN |O |S  |   |   |   |   |          |        |
|  |  |   |P |   |   |   |   |   |          |        |
`---------------------------------------------------'
```

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
----
```

7.15 PASSCOUNT

OPERATION NAME: Execution Pass Counter

MNEMONIC: PASSCOUNT

DESCRIPTION: May be used to maintain a program passcount. Each time a BUMP statement is executed PASSCOUNT is incremented (see BUMP statement)

INITIALIZED TO: Zero

EXAMPLE(S):

```
.
.
.
> 200  .END OF PROGRAM
-----
> 210  BUMP   .INCREMENT PASSCOUNT AND PRINT IT
-----
> 220  GOSUB 500   .GO CHECK FOR LOOP
-----
.
.

          -or-

>290 .Display PASSCOUNT
----
>300 LET PASSCOUNT:=PASSCOUNT+1
----
>310 PRINT "End of pass ";PASSCOUNT
----
```

7.16 RUNPARAM1/RUNPARAM2/RUNPARAM3

OPERATION NAME: Run Parameters

MNEMONIC: RUNPARAM1/RUNPARAM2/RUNPARAM3

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

INITIALIZED TO: Parameters input with the RUN Command

EXAMPLE(S):

```
> 10  IF RUNPARAM2=2 THEN 50
----                  .If the second parameter in
                      .the RUN command was 2 then
                      .go to 50
```

or

```
> 10 RUN 2,,4  (RUNPARAM1=2, RUNPARAM2=0, RUNPARAM3=4)
----
```

7.17 SECTION

OPERATION NAME: Section Number

MNEMONIC: SECTION

DESCRIPTION: During program execution, any SECTION statement* will alter the SECTION Reserved Variable to the current section number if the section is executed.

INITIALIZED TO: Zero

EXAMPLE(S):

(Assume TEST 10 was entered prior to execution)

```
> 100  SECTION 10,300   .SECTION RESERVED VARIABLE SET TO 10
-----
> 300  SECTION 11,400   (SECTION IS UNCHANGED BECAUSE
-----                    SECTION 11 WILL NOT BE EXECUTED)
```

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

7.18 SECTIONS1/SECTIONS2/SECTIONS3

OPERATION NAME: Section Execution Indicators

MNEMONIC: SECTIONS1/SECTIONS2/SECTIONS3

DESCRIPTION: During a SECTION statement execution the bit in the Reserved Variable SECTIONS1, SECTIONS2 or SECTIONS3 correlating to the SECTION statement number is extracted, and, if it's a logical "1", the next sequential statement(s) will be executed otherwise control is transferred to the statement number in the SECTION statement. The format is:

```
Bit  0                                        15
    ---------------------------------------------
     1  2 . . . . . . . . . . . . . . . . . . 16    SECTIONS1
    ---------------------------------------------
     17 18 . . . . . . . . . . . . . . . . . . 32   SECTIONS2
    ---------------------------------------------
     33 34 . . . . . . . . . . . . . . . . . . 48   SECTIONS3
    ---------------------------------------------
```

These variables are altered by the TEST command or, if no TEST has been entered, at RUN time where they are stored with all "ones".

INITIALIZED TO: Minus one if no TEST Command (without parameters) was entered otherwise not altered.

EXAMPLE(S):

```
> TEST 1,17,33 (Bit 0 of SECTIONS1/3 are set to "1" and
-                   the rest are set to "0" meaning only
                    SECTIONS 1, 17 and 33 may be
                    executed.)

       -or-

> 10  LET SECTIONS1:=SECTIONS2:=SECTIONS3:=!8000
----                      (Yields the same result as the
                           TEST command above when executed)
```

7.19 STEP

OPERATION NAME: Step Number

MNEMONIC: STEP

DESCRIPTION: STEP is provided so that the user's current STEP number may be available to AID or the user program. A postive and non-zero value in STEP will cause PPRINT and EPRINT Statement messages to be preceded by a header message indicating the program is in that STEP.

INITIALIZED TO: Zero

EXAMPLE(S):

```
> 5    .START STEP 1 TO CHECK XYZ
----
> 10   LET STEP:=1
----
.              .A FAILURE ANYWHERE MAY DESIGNATE
.              .THE STEP NUMBER.
> 1000 .END OF STEP 1
------
```

-or-

```
> 10  .START STEP 2 TO CHECK ABC
----
> 20  LET STEP:=2
----
> 30  PPRINT*"HELLO"
----
> 40  EPRINT*"ERROR"
----
> 50  RUN
----

Step 2: HELLO
-------------
Error in Step 2: ERROR
----------------------
End of AID user program
-----------------------
```

7.20 TIMEOUT

OPERATION NAME: Channel Program Timeout Flag

MNEMONIC: TIMEOUT

DESCRIPTION: To disable the software timer (default approximately 5 seconds), the user program may set TIMEOUT equal to -1. To increase the default timeout by N times 5 seconds, the user may set TIMEOUT to N in an assignment statement.

INITIALIZED TO: Zero

EXAMPLE(S):

```
> 10 .SET UP FOR SCOPE LOOP
----
> 20 LET CHANNEL:=2
----
> 30 TIMEOUT:=-1 .DISABLE I/O TIMEOUTS
----
> 40 DB CC,3,!1400 .READ DISC ADDRESS
----
> 50 BSIO AA
----
> 60 WR 8,CC(0),2
----
> 70 RR 8,CC(1),4
----
> 80 JUMP 60
----
> 90  RSIO
----
> 100 RUN
-----
```

7.21 TRUE OR FALSE

OPERATION NAME: Truth Assignment

MNEMONIC: TRUE or FALSE

DESCRIPTION: Allows the programmer the ability to manipulate or assign variables as Boolean Values (even though they are really manipulated arithmetically internally).

INITIALIZED TO: TRUE is set to -1 and FALSE is set to 0

EXAMPLE(S):

```
> 10  LET A:=FALSE      .A=0
----
> 20  LET B:=TRUE       .B = -1
----
```

AID STATEMENTS (I/O — NON CHANNEL PROGRAM)

SECTION VIII

8.0 INTRODUCTION

The AID I/O Statements that do not reside within the BSIO-ESIO instructions are listed, in detail, in this section. 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.

8.1 ADDRESSOFF/ADDRESSON

OPERATION NAME: Prevent address increment

MNEMONIC: ADDRESSOFF/ADDRESSON

DESCRIPTION: Prevent (ADDRESSOFF) or allow (ADDRESSON which is the default) channel program data buffer address from updating after each byte transfer. These indicators determine the state of Bit 4 of Word 4 of Read/Write Channel instuctions (See Amigo I/O ERS).

8.2 BSIO

OPERATION NAME: Begin Channel Program

MNEMONIC: BSIO XX[,C]

DESCRIPTION: This statement is used to mark the start of the definition of a Channel program. During user program execution, the Channel Program is completely defined when the ESIO or RSIO statement is reached. No direct I/O or DB statements may be placed within a BSIO-ESIO pair.

The Channel program is stored in buffer XX. Any previous definition of XX is purged. C is the number of copies to make (1<=C<=32). Default for C is 1. XX has the following format when the definition is complete:

Word(s)	Definition
0	Length (quantity n*) of Channel program.
1 (bits 0-7)	Number of words (quantity s*) to save after channel program executes. Examples of cases where needed are RREG and DSJ.
1 (bits 8-15)	Number of copies minus one.
2	Dirty** copy mask where bit0-bit15 indicate status of copies 1-16(dirty=Bit set).
3	Dirty** copy mask where bit0-bit15 indicate status of copies 17-32(dirty=Bit set).
4	SPARE
5 to n + 4	Master copy of Channel program.

* The quantities n and s are used in formulas under the WORD(S) heading.

**Dirty implies already executed (therefore needing recopying before another execution is attempted).

n+5 to n+4+(2*s)	Two word pairs for saving words after the channel program executes. First word=relative location within Channel program. Second word= relative location of variable.
n+5+(2*s) to 2n+4+(2*s)	Place to put first copy of Channel program. (First copy is copy 0.)
2n+5+(2*s)to 3n+4+(2*s)	Place to put second copy of Channel program. (If c>1)
	.
	.
	.
8n+5+(2*s) to 9n+4+(2*s)	Place to put eighth copy of Channel program. (If c>7)

.
.
.

EXAMPLE(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 PROGRAM
----
> 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.

EXAMPLE(S):

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

8.5 ESIO

OPERATION NAME: End Channel Program Definition

MNEMONIC: ESIO

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

EXAMPLE(S): See BSIO

8.6 HIOP

OPERATION NAME: Halt Channel Program

MNEMONIC: 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 present
----
> 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.

AID STATEMENTS (CHANNEL PROGRAM TYPE)

SECTION IX

9.0 INTRODUCTION

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 DOWNLOAD
----
> 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 WEMAR

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
-----
```

FUNCTION STATEMENTS

SECTION X

10.0 INTRODUCTION

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) into A
120 GETNAMEDATA NAME1,-3,B .Store contents of AA(7) into B
   .
   .
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 in 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:

```
 0   1                 8                 15
.-------------------------------------------.
|type |                 |  name              |
`-------------------------------------------'
```

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)
```

-hp-

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

Computer Museum

Sleuth Simulator Diagnostic Language
Reference Manual

HEWLETT PACKARD

Manual Part No. 30070-90018

Print Date: Sept. 1978
Change 1 March 1979

CONTENTS

GENERAL INFORMATION

SECTION I

1.0 INTRODUCTION

The Sleuth Simulator language simulates the HP 3000 Series II and III Sleuth programming language. The purpose of the Sleuth Simulator is to provide as many of the HP 3000 Series II/III statements as possible to a user of an HP 3000/33 system.

The simulator is writen in HP AID, a lower level language, and AID is written in SPL II. The simulator is actually a series of AID functions, which are a series of HP AID statements, and simulate each particular Sleuth statement defined in this manual. The simulator will maintain Sleuth's ability to run up to eight devices of various types concurrently.

Note that not all Sleuth commands and statements, as indicated in the Sleuth Manual for HP 3000 Series II/III. Subsection 1.3 lists those commands and statements not included and the reason why.

1.1 HARDWARE REQUIREMENTS

The Sleuth Simulator can run on an HP 3000/33 computing system with the following minimum equipment:

- Memory - 128K bytes
- System Console
- Flexible Disc Drive (7902) for cold loading DUS system.

1.2 SOFTWARE REQUIREMENTS

- Diagnostic Utility System that includes AID and Sleuth Simulator.
- AID and Sleuth Simulator manuals

1.3 SLEUTH SIMULATOR LIMITATIONS

The Sleuth Simulator is a separate program written in the HP AID language. When you enter a Sleuth program, the Sleuth Simulator becomes a part of this program. With the Simulator becoming part of a user's program, the variables and word buffers normally available to an HP AID user program have been limited as follows:

Variables A through N are available

Word Buffers AA through NN are available

String Buffers &AA through &VV are available

All Reserved Variables are available

If you use any of the simulation variables, word buffers, or string buffers, an error may be reported. If not, the operation of your program will be adversely affected.

HP 3000 Series II/III Sleuth statements and commands that will not be simulated fall into four catagories:

- Statements and commands that have the same mnemonic as an HP AID statement, command, reserved variable or buffer name.
- Statements not required under HP AID and commands that perform the same or similar function as HP AID commands.
- Statements pertaining to the items that are unique to the HP 3000 Series II/III system.
- Statements and commands that can only be partially simulated under HP AID.

1.30 Sleuth/AID Duplicate Mnemonic Limitations

An AID function cannot be created in HP AID for any HP 3000 Series II/III Sleuth statment or command that falls into this catagory. The items listed below, therefore, will not be simulated. The equivalent AID format is shown next to the Sleuth item not to be simulated. The AID statements and commands have been included in this manual and in the AID manual for ease of reference.

Sleuth Statements	HP AID Statement Format
Bump - Bump pass counter	Bump [;] [H]
For - For initial value to final value	For variable:=initial value to final value (see ERS for additional form)
*Go - Conditional branch	Go - HP AID command.
Goto - Goto step #	Goto statement #

Sleuth Statements	HP AID Statement Format
If - If exp then statement #	If exp [special op exp] [special op exp] then statement #
Let - Let var = exp	Let var:= exp or string
*Loop - Loop to statement #	Loopto statement # Loop command must be executed first
Next - Next variable	Next statement #
Proc - Proceed	Proc [N]
* These statements cannot be implemented as functions under any name. HP AID does not allow variable branching which would be needed to implement these statements as functions. The HP AID format shown for these statements does not imply they will work the same as these Sleuth statements.	

Sleuth Commands	HP AID Command Format
EP - Erase program	EP
List - List variable (only displays contents of variables)	List [printer][data type] [statement #] (Displays contents of buffer, variables, or program)
Ren - Renumber (Increments vary between 3 & 10)	Ren [C] (Increments [C] are entered by the user)
Run - Run	Run [P1],[[P2] [,P3]]

1.31 Statements Not Required/Command Functions Duplicated in AID

The second category petains to Sleuth statements that are not required when using HP AID. It also includes those Sleuth commands replaced by HP AID commands performing the same or similar functions. The statements and commands listed below fall into this category. Refer to the Command Section 3 for a description of all commands (including applicable AID commands) used by a Sleuth program.

Sleuth Mnemonic	HP AID Format
Auto - Auto numbering resumed	Automatic numbering mode is on at all times, including editing of programs.
*BA - Batch in tests from the Mag Tape	Load filename
Dump - Lists buffers,variables and programs	L[ist][p[rinter]][data type] [statement #]
End - Terminates a program (Required in all programs)	End statement not required in HP AID. HP AID provides an implied END after each program. In AID, END statement will end user program
*Makt - Make test tape-Mag tape	Save filename
	D[elete] [range] Deletes a range of text or program lines already entered
*Media used for storing and retreiving programs on a Series 33 system is a flexible disc.	

1.32 HP 3000 Series II/III Unique Statements

The third category of statements not available for simulation are those that pertain to the items that are unique to the HP 3000 Series II/III systems. This includes all Sleuth statements associated with testing the I/O structure, I/O PC boards and unique peripheral devices for the series II/III. These items are the ISS Disc, 7900A disc, Fixed Head Disc, Paper Tape Reader, Asynchronous Multiplexor, Universal Interface, Plotter, Paper Tape Punch, Card Reader, Selector Channel, all Direct I/O statements, all SIO order statements and the Set Bank (SBNK) General statement used with SIO orders. The Read Clock (RCLK) and Set Clock (SLCK) statements, which were used for reading and setting the process clock in the Series II/III CPU, will not be simulated because this feature is not available on the Series 33 system.

1.33 Partial Simulation Limitation

The fourth category pertains to Sleuth statements that can be converted into simple HP AID functions or that can only be partially simulated by HP AID (i.e., the remaining portion of the statement can only be controlled by an AID command). For these situations it is better for the user to become familiar with the capabilities of HP AID with regards to these statements. The Sleuth statements that fall into this category are listed below with the HP AID statement or command that will provide equivalent capabilities. Those commands and statements listed below have been included in this manual for ease of reference and use and can also be found in the AID manual.

Sleuth Statement	HP AID Statement	HP AID Command
ACB - Access single word element in buf.	LET - Makes an assignment to any element of buffer.	
PUT - Prints a message on the console.	PRINT - Prints a message on the console.	
Halt - Halt computer (Causes a halt %17)	Pause - Creates an unconditional pause in the execution of a HP AID program	
Nopr - No print (Turns off all messages except user Sleuth dialogue)	Suppress - Suppress all error messages	Sepr - Suppress error printout -or- Snpr - Suppress non-error print
Pr - Print (Causes resumption of all printing)	Enable - Enable error reporting	Eepr - Enable error printout -or- Enpr - Enable non error printout
ZBUF (buf) Zero defined buffer	DB Buf,length,0	
DELY ^Delays the software timer in increments of .1sec	TIMEOUT - delays the software timer in increments of 5 secs.	

1.34 Disc Limitations

The Sleuth default mode, for the file mask (13037 controller.), in the HP 3000 Series II/III is cylinder mode. For Series 33 it will be "surface mode". This limitation is created by AID's inability to distinguish a difference between a parameter of zero and an omitted parameter (both appear as zero). For example: if the following statement is entered

```
RDI 0,AA(0),0
```

The simulator will set the file mask to zero. If the last zero is not entered at all, AID will still pass the simulator a zero. Therefore, the simulator cannot distinguish between an omitted parameter and a zero (0).

OPERATING INSTRUCTIONS

SECTION II

2.0 INTRODUCTION

The Sleuth Simulator program (written in AID language) is physically located on the Diagnostic Utility System cold load diskette under the file name SLEUTHSM. To execute the Sleuth Simulator program, perform the following procedure:

1. Cold load the Diagnostic Utility System (DUS) diskette.
2. Once the DUS program has output its title message and prompt (:) enter "AID".
3. AID will respond with a prompt character (>) and line number:

   >10

4. Enter "LOAD SLEUTHSM". The Sleuth Simulator is now loaded and you may enter your program statements or use the available commands.

2.1 ENTERING A SLEUTH PROGRAM

User programs are entered at the first available AID line number after the simulator program. Note that the simulator will become part of the user program entered.

2.2 DELETING A SLEUTH PROGRAM

To erase the lines of code generated by your entries, the delete command must be used as it will erase only the lines specified:

```
D(elete) 5000/5100
```

To erase both the Sleuth Simulator and your program use the EP command. If this occurs inadvertantly, you can load the simulator again by entering "LOAD SLEUTHSM".

All commands and statement descriptions can be found in Section 3 of this manual.

2.3 PROCEED MODE

The Sleuth Simulator does not turn off the proceed mode at any time. A user should use this HP AID statement with caution. Refer to the HP AID manual for more information on this statement.

STATEMENT/COMMAND DESCRIPTIONS

SECTION III

3.0 INTRODUCTION

The following pages in this section will describe the capabilities of each simulated sleuth statement. Statements that have the same mnemonic as an HP AID statement, command, reserved variable, or buffer name (AA-ZZ), that are being simulated, will have an S preceeding the original mnemonic (i.e., compare buffer (CB) will become (SCB).

NOTE

All buffers and variables must be in upper case. The simulator will not recognize lower case letters.

Functions that differ from the original Sleuth statement will either describe the difference or refer to an HP AID equivalent statement that will perform that specific task.

The syntax for each of the following statements defines what the parameters of the statement are. If a parameter is optional it will be enclosed by brackets (i.e., SEEK lun [,cylinder,head,sector]). If SEEK 3 is entered in a user program, then a seek for logical unit 3 to cylinder 0, head 0, sector 0 would be issued. The parameters that are not enclosed by brackets are required inputs. If any parameter is not entered, the default for that particular parameter is 0. This implies that a SEEK statement, by itself, will issue a seek for logical unit 0, to cylinder 0, head 0, sector 0.

The statement descriptions on the following pages, are subdivided into the following categories and are presented in alphabetical order within the subdivision.

- General Statements
- Disc I/O Statements
- Line Printer I/O Statements
- Magnetic Tape I/O Statements

3.1 STATEMENT/COMMAND SECTION INDEX

Statement	Description	GEN.	Disc	LP	MT
ID	Initialize data		53		
IDI	Initialize data immediate		55		
IF-THEN	Conditional program branch true	29			
IFN-THEN	Conditional program branch false	30			
INPUT	Input data	31			
IS	Incremental Seek		57		
IT	Increment track		58		
LET	Assignment of variables	32			
LOOPTO	Conditional Loop Branch	33			
MC	Master Clear	34			
NEXT	End of For-Step-Until-Next loop	35			
PAUSE	No-error Pause	36			
PE	Pause on Error	37			
POLL	Resume polling devices		60		
PRINT	Print to console without pause	38			
PROC	Proceed without end of Chan Prog	39			
RAND	Randomize	40			
RC	Recalibrate		61		
RD	Read data		62		104
RDA	Request Disc Address		64		
RDB	Randomize Define Buffer	41			
RDI	Read Data Immediate		65		
REW	Rewind tape				105
REWOFF	Rewind Offline				106

Statement	Description	Page Numbers Gen.	Disc	LP	MT
RFS	Read Full sector		67		
RFSI	Read full sector immediate		68		
RP	Ripple Print			95	
RQST	Request status		69		
RS	Random Seek		70		
RSA	Request Sector address		71		
RSYN	Request Syndrome		72		
RWO	Read with offset		74		
RWOI	Read with offset immediate		76		
RWV	Read without Verify		77		
RWVI	Read without Verify immediate		79		
SCB	Simulated compare buffer	42			
SEEK	Seek		81		
SELU	Select Unit				107
SFM	Set File Mask		82		
SKRD	Seek and Read data		83		
SKWD	Seek and Write data		85		
SOUT	Switch output	43			
SST	Suppress Status	44			
STAT	Status Dump	45			
TIMEOUT	Channel Program Timeout flag	46			

Statement	Description	Page Numbers: Gen.	Disc	LP	MT
VER	Verify disc		87		
VERI	Verify disc immediate		88		
WD	Write Data		89	96	108
WDI	Write Data Immediate		91		
WFM	Write File Mark				109
WFS	Write Full Sector		92		
WFSI	Write Full Sector Immediate		93		

3.2 GENERAL COMMANDS (AID/SLEUTH)

Refer to the AID Manual for commands as all AID commands are valid for Sleuth programs.

3.3 GENERAL STATEMENTS

General Statements control system oriented data manipulation. Each statement description contains the formal name, the function name or mnemonic, the syntax of the statement, a parameter description, a description of the statments operation, and an example of the statement usage.

ASSIGN

==

FORMAL NAME: Assign Data to Buffer

FUNCTION NAME: ASSIGN data buffer(element)[,(repeat factor)],datal[,da

DESCRIPTION: Stores data into a data buffer. The word datal 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 If repeat factor is included the data pattern is repeated repeat factor times. Datal through datan must be numeric.

EXAMPLE(S):

```
>5000 DB AA,100,%55          .INITIALIZE AA TO %55
----
>5010 ASSIGN AA(50),5,10,15,20,25,30,35
----  (AA(50)=5, AA(51)=10, . . . AA(56)=35)

>5010 ASSIGN AA(10),(10),!FF
----  (AA(10) THROUGH AA(19))=!FF)

>5010 ASSIGN AA(80),(5),3,7
----  (AA(80)=3, AA(81)=7, AA(82)=3, AA(83)=7 etc.)

>5010 LET A:=80,F:=5
----
>5020 ASSIGN AA(A),(F),3,7
----   (Same as ASSIGN statement 5010 above)
```

BUMP

==

FORMAL NAME: Bump Pass Counter

FUNCTION NAME: 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 PASSCO initialized to zero whenever a RUN Command is issued. Printing may be suppressed by a SNPR Command and, if th optional semi-colon follows BUMP, no return-line feed will be issued after the pass counter value is printed.

```
>5000 BUMP H
>5010 RUN
```

System outputs "END OF PASS 0". Note that passcount is still 0 after the print because of the H parm.

```
>5000 BUMP;
>5010 PRINT "FOUND A BUG!!"
>5020 RUN
```

System outputs "END OF PASS 1 FOUND A BUG!!".

CB

==

FORMAL NAME: Compare Buffers

FUNCTION NAME: 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):

```
>5000 CB AA(10),BB(10),10
----  (COMPARE BUFFER AA WITH BB FOR 10 ELEMENTS
----  STARTING WITH ELEMENT 10 IN EACH BUFFER)

>5010 IF INDEX <> -1 THEN 200  . REPORT ERROR
      ROUTINE A
----
>5000 CB &CC(5), &DD(10), 6
----  (COMPARE BYTES 5-10 OF &CC TO BYTES
----  10-15 OF &DD)

>5010 IF INDEX = -1 THEN 100
----  (IF INDEX = -1 THEN COMPARE 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)).

CHB

==

FORMAL NAME: Change Buffer

FUNCTION NAME: CHB

SYNTAX: >CHB buf(0),type

PARAMETERS: buf - Buffer to be changed.
This parameter must be any buffer AA(0)-NN(0) where AA-NN define buffer name and (0) sets an HP AID pointer to the first element in the buffer.

type - Type of change.

TYPE	FUNCTION
A	Fill with address
R	Randomize
I	Increment
D	Decrement
S	Circular shift left (shifts bits within each element 1 place to left)
W	Circular word shift (shifts words within buffer 1st to last and all else moves down one position)

OPERATION: The CHB command will change the contents of the specified buffer.

EXAMPLE:

```
000 DEV 0,2,7,20,0
005 DB AA,4096
010 ASSIGN AA(0),(1024),%52525,%125252,%66666,%33333
020 DB BB,4096,0
030 FOR C:= 1 UNTIL 100
040 FOR I:=0 TO 410
050 WD 0,AA(0),7,I,0,0
060 RD 0,BB(0),7,I,0,0
065 SCB 0,AA(0),BB(0),5
070 NEXT 5040
080 CHB AA(0),R
090 NEXT 5030
100 RUN
```

This example uses the CHB function to randomize the data buffer AA. It writes the preassigned buffer AA on the first 32 sectors of surface 0 (head 0), reads and compares the data. The buffer is then randomized and the process is repeated 100 times. NOTE: Buffer manipulation with this function is slow.

CORB

==

FORMAL NAME: Correct Buffer

FUNCTION NAME: CORB

SYNTAX: >CORB lun,buf(x)

PARAMETERS: lun - Logical unit number; must be an HP Disc using the HP 13037 controller.

buf(x) - Buffer to be corrected. This parameter must be any buffer AA(x)-NN(x) where AA-NN define buffer name and (x) sets an HP AID pointer to the first element in the buffer specified by the user.

OPERATION: This statement will correct the data buffer specified by the buf parameter according to the last syndrome requested for the logical unit designated.

EXAMPLE:

```
>5000 DEV 0,2,7,10,0
>5010 DB AA,6144,%66666
>5020 DB BB,6144,0
>5030 FOR A:= 0 TO 822
>5040 SEEK 0,A,1,0
>5050 WDI 0,AA(0)        (Note disc is in surface
                          mode)
>5060 RD 0,BB(0),1,A,1,0
>5070 IF SS(0)=%7400 THEN 5110
                         (disc status word 1 and
                         2 is stored in SS(0) and
                         SS(1))
>5080 SCB 0,AA(0),BB(0),5
>5090 NEXT 5030
>5100 END
>5110 RSYN 0
>5120 CORB 0,BB(0)
>5130 GOTO 5090
>5140 RUN
```

This program writes one track of data on surface 1, reads and checks for possible correctable errors and then compares buffers. If a possible correctable data error occurs, the data buffer (BB) will be corrected if the request syndrome (RSYN) indicates that the data is correctable.

DB

===

FORMAL NAME: Define Buffer

FUNCTION NAME: 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 b is stored with that string numeric or variable value. Dynamic allocation of buffers is allowed, but may cause large overhead in execution time since existing buffers "packed" to allow room for a new buffer. Dynamic allocation will leave the existing element values unchanged.

EXAMPLE(S):

```
>5000 DB AA,100 (Declares the buffer AA as 100 words
                 long)

>5000 DB &AA,10 (Declares the string buffer &AA as
                 10 bytes long (note aa and &AA are
                 separate buffers))

>5000 DB &CC,100,"START" (Each sequential 5 byte set
                          of &CC contains "START")

>5000 DB CC,100,0 (Stores 0 in all 100 elements of
                   CC)

>5000 DB CC,110  (Reallocate CC to 110 words (first
                  100 elements intact))

>5000 DB CC,0  (Deletes buffer CC)
```

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

DEV

FORMAL NAME: Device

FUNCTION NAME: DEV

SYNTAX: >DEV lun,chan,dev,errs,unit

PARAMETERS: lun - Logical unit number (0 to 7).

chan - Channel number device connected to (0 to 15).

dev - Device number (0 to 7).

errs - Maximum error count the device is allowed (1 to 999).

unit - Device unit number (0 to 7).

OPERATION: The Device statement allows the user to define the characteristics of a particular device and to assign an error count and logical unit number to that device. This function will test for boundaries on all parameters, see if the entered channel and device are present, identify the device, obtain the device type for the 79XX discs (13037 controlled) and store these parameters in buffer ZZ for future use. The DEV function buffer (ZZ) is structured as described in Appendix A. If any of these parameters exceed the boundaries or if a non-existent channel or device has been entered an error message is output to the console and the program ends.

EXAMPLE:

```
>5000 DEV 1,3,2,10,0
        -OR-
>5000 DEV 0,2,7,25,3
```

ES

FORMAL NAME: Enable Status

FUNCTION NAME: ES

SYNTAX: >ES

OPERATION: Enable Status will enable automatic checking of device status when utilizing Sleuth simulated statements (HP AID Functions).

EXAMPLE:

```
>5000 DEV 0,2,7,15,2
>5005 RDB AA(0),128
>5010 DB BB,128,0
>5015 FOR A:= 0 TO 99
>5018 FOR B:= 0 TO 822
>5020 WD 0,AA(0),1,A,0,63
>5060 IFN B=822 THEN 5080
>5070 SST                (suppress status)
>5080 RD 0,BB(0),1,A,0,63
>5090 SCB 0,AA(0),BB(0),4
>5100 DB BB,128,0 (zero out buffer BB)
>5110 ES
>5120 NEXT 5040
>5130 NEXT 5050
>RUN
```

This example will test the last sector (63) on surface zero for a 7925A disc. The disc file mask (1) is set (line 5020) to allow the unit to increment beyond the end of cylinder. A test is made for cylinder 822. When cylinder 822 is reached, the status is suppressed for the read operation. This is required to prevent a seek check status error that will occur because of the buffering scheme of the 12745A disc interface. Refer to the RDA function for further information. Status checking is then enabled and this process is repeated for 99 more times.

ESTA

===

FORMAL NAME: Expected Status

FUNCTION NAME: ESTA

SYNTAX: >ESTA [status1[,mask[,status2[,mask]]

PARAMETERS: status1 - First status word for discs, line printers or first two bytes of status for Mag Tape.

mask - A word of don´t care bits. A 1 in the mask corresponds to a don´t care bit in the status

status2 - Second status word for discs or third byte of status for Mag Tape. Mag Tape status byte is left justified (bits 0-7).

NOTE

If either the status or "mask" parameters are omitted, status and mask equal 0.

OPERATION: This statement changes the expected status of the next statement which utilizes status checking.

EXAMPLE:

```
>5000 DEV 2,2,7,10,0
>5010 ESTA !1300,!8604,16100
>5020 SEEK 2,150,3,0
>5030 GOTO 5010
>5040 RUN
```

In this example a 7925A disc will continue to seek to cylinder 150, head 3, sector0. The expected status is set for a status 2 error of seek check. If the seek actually completes, then the following message will appear on the console:

```
79XX DISC STATUS WORD 1                          WORD 2
                 ------                          ------
STATUS IS   0 0 0 11111 0000 0000 / 0 00 0011 0 0 0 0 0 0 0 0 0
SHOULD BE   0 0 0 10011 0000 0000 / 1 XX 0011 X 0 0 0 0 0 1 0 0

CYLINDER = 0, SECTOR = 0, HEAD = 0
```

FOR-STEP-UNTIL

==

FORMAL NAME: For-Step-Until

FUNCTION NAME: F[OR] 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 buffer element). The STEP parameter is an optional increment the FOR variable with a default of 1. The FORNEXT sequence repeated until the terminator expression value is exceeded 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):

```
>5000 FOR I:=5 TO 50
       .
       .
>5060 NEXT 5000
```

This for statement will execute the statements between 10 and 100 (46 times) with I=5 through 50 stepping one at a time.

```
>5000 FOR I:=5 STEP 8 UNTIL 50
       .
       .
>5060 NEXT 10
```

This FOR statement will execute the statement between 10 and 100 (6 times) with I=5,13,21,29 37,45)

```
>5000 for i:=5 step B:=8 until C:=50
       .
       .
>5060 NEXT 10
```

This statement sequence provides the same sequence of the above statments.

```
>5000 FOR AA(2):=-5 TO 50
       .
       .
>5060 NEXT 10
```

Buffer element AA(2) will step -5,-4,-3,-2,-1, 0,1,......50.

*If the STEP value is negative the sequence will repeat until the FOR value is less then the UNTIL value.

GET

===

FORMAL NAME: Get (Obtain logical unit information)

FUNCTION NAME: GET

SYNTAX: >Get lun,C or D or E or U

PARAMETERS: lun - Logical unit number.

C = Channel number
D = Device number
E = Error count
U = Unit number

OPERATION: This statement will read from the console these parameters only. The HP AID statement (INPUT) provides the standard capability of receiving operator input from the console.

NOTE: AID statement INPUT can be interspersed with Sleuth Simulator statements as with most AID statements.

EXAMPLE:

```
>5000 PRINT "ENTER THE CHANNEL NUMBER"
>5010 GET 0,C
>5020 PRINT "NUMBER OF ERRORS?"
>5030 GET 0,E
>5040 PRINT "NUMBER OF PASSES?"
>5050 INPUT A
>5060 FOR I:=1 TO A
>5070 PRINT "PASS NUMBER";I
>5080 NEXT 5060
>5090 RUN

ENTER THE CHANNEL NUMBER
?3
--
NUMBER OF ERRORS?
?10
---
NUMBER OF PASSES
?2
--
PASS NUMBER 1
PASS NUMBER 2
PASS NUMBER 3

END OF AID USER PROGRAM
```

This example shows how the GET statement may be used to dynamically obtain information from the operator. Note that the operator input is underlined.

GOTO

==

FORMAL NAME: GO TO (Unconditional Branch)

FUNCTION NAME: GOTO Statement Number

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

EXAMPLE: >5000 GOTO 50

The above statement transfers control to statement 50.

IF THEN

==

FORMAL NAME: If-Then Control

FUNCTION NAME: IF exp [[SPECIAL OPERATOR exp][SPECIAL OPERATOR exp]] THEN statement number

DESCRIPTION: Allows the executing program to evaluate "exp" and if it is 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 seperated 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)

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):

```
>5000 IF AA(2) THEN 50 (If AA(2) is true (non-zero)
                        go to 50)

>5000 IF A OR B THEN 30 (The expression "A or B" is
                         evaluated)

>5000 IF 14 LE A:=A+1 LE 20 THEN 120
                       (Test if A+1 is between 14
                          and 20 INCLUSIVE)

>5000 IF A:=A+1 GE B:=B+1 GE C:=C+1 THEN 200
                     (Test IF (A+1)>=(B+1)>=(C+1))

>5000 IF 1 LT B LT 100 THEN 20
                 .TEST IF B IS BETWEEN 1 AND 100**.
```

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

IFN THEN

==

FORMAL NAME: IF-NOT-THEN

FUNCTION NAME: 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.

EXAMPLES:

```
>5000 IF 1 LE A LE 14 THEN 20
                    (If a is between 1 and 14
                    go to 20)

>5000 IFN 1 LE A LE 14 THEN 20
                     (If A is "NOT" between 1
                     and 14 go to 20)
>5000 IF A THEN 20  (If A<>0 go to 20)

>5000 IFN A THEN 20 (If A=0 go to 20)
```

INPUT

FORMAL NAME: Input Data

FUNCTION NAME: 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. See the Reserved Variable INPUTLEN for determining the character length of the input.

EXAMPLES:

```
>5000 INPUT A (value input from console is
               interpreted and then stored in A)

>5000 INPUT AA(2)
 (the console input will be stored in AA(2))

>5000 INPUT &BB(2,6)
 (5 characters are accepted from console and stored
 in string buffer BB starting at element 2 - string
 buffers must be used to contain ASCII characters)

>5000 INPUT A,B,C
 (3 numeric values, separated by commas are
 accepted from the console and stored in variables
 A, B, and C respectively)

>5000 INPUT A
 (1 numeric value is accepted from the console)
```

NOTE: If you fail to enter the correct amount of input parameters at the console, the INPUT function will output a double ?? until all parameters, called for by the INPUT statement, have been entered.

LET

OPERATION NAME: Assignment

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

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

EXAMPLE(S):

```
>5000 LET A:=10    .A IS ASSIGNED THE VALUE 10
----
>5000 LET C:=D+E   .C IS ASSIGNED THE SUM OF D+E.
----
>5000 LET AA(2):=!F   .ELEMENT 2 OF THE BUFFER AA IS
----                  .THE HEXADECIMAL VALUE F.

>5000 LET A:=C:=4   .MULTIPLE VARIABLE ASSIGNMENTS
----
>5000 LET A:=4,B:=7 .MULTIPLE EXPRESSION ASSIGNMENT
----                  ALLOWED.
>5000 LET AA(4):=B .ELEMENT 4 OF BUFFER AA IS ASSIGN
----               .THE VALUE OF THE B VARIABLE.

>5000 LET &AA(5,9):="HELLO"
----          .&AA(5,6)=HE, &AA(7,8)=LL,
              &AA(),10)=OO
>5000 A:=10        .IDENTICAL TO FIRST EXAMPLE
----
>5000 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.

LOOPTO

===

OPERATION NAME: Conditional Loop Branch

MNEMONIC: LOOPTO label

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

EXAMPLE(S):

```
> 100 SECTION 1,200
-----
      .
      .
> 200 SECTION 2,500
-----
      .
      .
> 500 LOOPTO 100 . Go to 100 if LOOP flag is set.
-----
```

MC

==

FORMAL NAME: Master Clear

FUNCTION NAME: MC

SYNTAX: >MC lun

PARAMETER: lun - Logical unit number.

OPERATION: This function will clear the specified unit by issuing a device clear. For the 2608A printer, a master clear will be sent and for the 2631A printer a clear 1 (device clear with parity enable) is issued.

EXAMPLE:

```
>5000 DEV 4,2,3,10,5
>5010 SST
>5020 SEEK 4,823  (Create a seek check)
>5030 ES
>5040 MC 4        (Clear device and status)
>5050 SEEK 4      (Seek to zero (0))
>5060 GOTO 5010
>5070 RUN
```

This example will continually loop forcing a seek check error on a 7920A disc. The status is suppressed during the error condition, enabled afterword and cleared out by the master clear (MC). The heads are then repositioned to cylinder 0, head 0, sector 0.

NOTE: Recalibrate will not work in place of the seek in line 5050 because the disc address (cyl. 823) is beyond the maximum limit.

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.

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 onl 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)
-
```

PE

==

FORMAL NAME: Pause On Error

FUNCTION NAME: PE

SYNTAX: >PE lun

PARAMETER: lun - Logical unit number

OPERATION: This function will notify the user that an error has occurred and stop the execution of the users program.

Once the function has been executed it can only be defeated by an AID suppress non-error pause command (SNPS). Program can also be continued by typing GO.

EXAMPLE:

```
>5000 DEV 0,2,7,15,0
>5010 PE 0
>5020 FOR C:=1 UNTIL 4000
>5030 RS 0
>5040 NEXT 5020
>5050 RUN
```

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 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.

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 state

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

```
> 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
------
```

RAND

==

FORMAL NAME: Randomize

FUNCTION: RAND

SYNTAX: >RAND var

PARAMETER: var - A variable designated by a letter A thru N.

OPERATION: This function generates a positive random number and places it in the designated variable.

EXAMPLE:

```
>5000 RAND A
>5010 RAND B
>5020 RAND C
>5030 RUN
```

This example places a random number in the variables A,B & C.

RDB

===

FORMAL NAME: Randomize Data Buffer

FUNCTION NAME: RDB

SYNTAX: > RDB name,length

PARAMETERS: name - Two letter buffer name in quotes.

length - Number of words allocated to buffer.

OPERATION: This function defines randomized data buffers only. Note: The HP AID statements (DB and ASSIGN) should be used for string buffers. This function is the same as the Sleuth DB statement for randomizing buffers. It does not provide the other features of the Sleuth DB statement. They can be implemented with AID commands. For example:

Function	HP AID Format
Define a data buffer (AA) and fill it with minus 1.	DB AA,100,-1
Define a string buffer (BB), 10 elements long	DB &BB,10
Define a data buffer (CC) with alternating data patterns of %33333 and %66666 for 100 words.	DB CC,100 ASSIGN CC(0),(50), %33333,%66666

Note: This function operates slowly (approx. 21 secs for 3972 words) due to the overhead required to access and modify a buffer.

EXAMPLE:

```
>5000 DEV 0,2,7,10,0
>5010 RDB AA(0),6144
>5020 FOR D:=0 UNTIL 822
>5030 SEEK 0,D,2,0
>5040 WDI 0,AA(0)
>5050 NEXT 5020
>5060 RUN
```

SCB

==

FORMAL NAME: Simulated Compare Buffer

FUNCTION NAME: SCB

SYNTAX: >SCB lun,bufl(0),buf2(0),errcount[,maxcount]

PARAMETERS: lun - Logical unit number of device being tested.

bufl(0),buf2(0) - Buffers of which the contents are to be word by word compared.

errcount - Maximum number of errors to be displayed for each execution.

maxcount - Maximum number of words to be compared. Uses smallest buffer length if defaulted.

OPERATION: The Compare Buffer command will compare word by word each element of buffers 1 and 2.

EXAMPLE:

```
>5000 DEV 0,2,7,10,0
>5005 DB AA,6144
>5010 ASSIGN AA(0),(1536),%125252,%52525,!FFFF,!AAAA
>5025 FOR C:=0 UNTIL 822
>5030 IT 0
>5040 WDI 0,AA(0),7
>5045 IT 0
>5050 VERI 0,48
>5060 DB BB,6144,0
>5065 IT 0
>5070 RDI 0,BB(0),7
>5080 SCB 0,AA(0),BB(0),4
>5090 NEXT 5025
>5100 RUN
```

The above example indicates how a compare buffer operation may be used to help evaluate the results of an operation. One track of information is written on a 7920A disc. It is then verified and compared until the entire disc is checked.

SOUT

FORMAL NAME: Switch Output

FUNCTION NAME: SOUT

SYNTAX: >SOUT

OPERATION: Switch output will output error messages to the lineprinter or the system console. Initially error messages will be output to the system console. Each SOUT statement will alternate the output device for error messages.

EXAMPLE:

```
>5000 DEV 0,2,7,10,1
>5010 DB AA,128,%052525
>5020 SOUT
>5030 FOR B:=0 TO 410
>5040 SEEK 0,B,0,0
>5050 WD 0,AA(0),7,B,0,0
>5070 NEXT 5030
>5080 RUN
```

The above example will switch the reporting of error messages from the console to the lineprinter. Note if a lineprinter is not connected to the system, the output will default to the console.

SST

===

FORMAL NAME: Suppress Status

FUNCTION NAME: SST

SYNTAX: >SST

OPERATION: This function will disable status checking fcr all statements following this function statement. Status checking can be re-enabled by using the Enable Status function (ES) of AID.

EXAMPLE:

```
>5000 DEV 5,4,1,10,3
>5010 DB AA,128,0
>5020 SST
>5030 RD 5,AA(0),1,822,8,47
>5040 PRINT "BUFFER AA(4)=",AA(4)
>5050 RUN
```

This example suppresses the status error that would normally occur when trying to read the last sector of a 7925A disc. Refer to the RDA function for more information on 12745A interface operation during a read. The fifth word of the sector is then displayed on the console.

STAT

FORMAL NAME: Status Dump

FUNCTION NAME: STAT

SYNTAX: >STAT lun[,C or ,D]

PARAMETERS: lun - logical unit number

C - Obtains channel registers (0-F) status

D - Obtains device status

OPERATION: This statement will obtain status from the channel or device specified whether an error has occurred or not and print it on the console.

EXAMPLE:

```
>5000 DEV 0,2,7,10,0
>5020 STAT D
>5030 RUN
```

The above example will print the device status of the disc. In this case status words 1 and 2 will be displayed on the console.

TIMEOUT

==

OPERATION NAME: Channel Program Timeout Flag

MNEMONIC: TIMEOUT

DESCRIPTION: To disable the software timer, the user program may set TIMEOUT equal to -1. To increase the time allowed by N times, the user may set TIME-OUT to N. The timeout period is approximately 3 seconds.

INITIALIZED TO: Zero

EXAMPLE(S):

```
> 10  .SET UP FOR SCOPE LOOP
----
> 20  LET CHANNEL:=2
----
> 30  TIMEOUT:=-1 .DISABLE I/O TIMEOUTS
----
> 40  DB CC,3,!1400     .READ DISC ADDRESS
----
> 50  BSIO AA
----
> 60  WR 8,CC(0),2
----
> 70  RR 8,CC(1),4
----
> 80  JUMP 60
----
> 90  RSIO
----
> 100 RUN
-----
```

3.4 DISC I/O STATEMENTS

The following statement descriptions pertain strictly to I/O operations concerning the disc subsystem.

The status for each device is stored in buffer SS as follows:

SS(0) 79XX disc 1st status word

SS(1) 79XX disc 2nd status word

AR

==

FORMAL NAME: Address Record

FUNCTION NAME: AR

SYNTAX: >AR lun[,cylinder,head,sector]

PARAMETERS: lun - Logical unit number.

cylinder - cylinder address

head - head address

sector - sector address

OPERATION: Sets logical address specified in the cylinder, head and sector parameters into the disc HP 13037 controller only.

EXAMPLE:

```
>5000 DEV 0,2,7,10,0
>5010 AR 0,4,2,3
>5020 RDA 0
>5030 DISP 0,D
>5040 RUN
```

The above example uses the Address Record function to set the logical disc address into the disc controller. The Request Disc Address function and the Display function obtain and display the address.

CL

FORMAL NAME: Clear

FUNCTION NAME: CL

SYNTAX: >CL lun

PARAMETER: lun - Logical unit number.

OPERATION: The clear function pertains to disc drives only. It will clear any clock offset, clear status, clear the interface busy bit and wait for a new command.

EXAMPLE:

```
>5000 DEV 0,2,7,10,0
>5010 CL 0
>5020 RUN
```

The above example issues a clear to a disc connected to channel 2, device 7.

DS

===

FORMAL NAME: Decremental Seek

FUNCTION NAME: DS

SYNTAX: >DS lun[,cylinder,head,sector]

PARAMETERS: lun - Logical unit number.

cylinder - cylinder address

head - head address

sector - sector address

This function will do an initial seek to the location specified by the cylinder, head, sector parameters, default is 0,0,0. Each time the instruction is executed the cylinder will be decremented by 1 until it reaches cylinder 0. When this occurs the disc will seek to the maximum cylinder. This function updates the internal disc address.

NOTE: This function only operates on discs controlled by the 13037 controller.

This function does not decrement a common cylinder table. It sets up a cylinder table based on the statement number making this function call. Everytime that statement # makes a call to this function it will decrement its unique table. NOTE: When using this function remember that all read, write, and verify operations update the 7906/20/25A discs internal address. If a write operation of 128 words started at cylinder 100,0,0 and you issued a read command following it, the read would begin at cylinder 100,0,1 If a decremental seek was issued before the write operation, another decremental seek would be issued before the read to properly position the heads.

The maximum number of DS function calls (separate entries) allowed for each program is ten (10).

EXAMPLE:

```
>5000 DEV 0,2,7,10,0
>5010 DS 0,822,0,0
>5020 GOTO 5020
>5030 RUN
```

DISP

==

FORMAL NAME: Display

FUNCTION NAME: DISP

SYNTAX: >DISP lun,type

PARAMETERS: lun - Logical unit number.

type	function
D	Disc Address
R	Requested Status
S	Sector Address
Y	Last Syndrome

OPERATION: This function will display the item specified in the type parameter for the lun indicated.

EXAMPLE:

```
>5000 DEV 1,2,7,10,0     (7920 disc)
>5010 RS 1
>5020 RQST 1
>5030 DISP 1,R
>5040 RUN
```

This program will issue a random seek to a 7920A disc, request the status after the seek completes and print the status on the console.

FMT

==

FORMAL NAME: Format

FUNCTION NAME: FMT

SYNTAX: >FMT lun

PARAMETER: lun - Logical unit number.

OPERATION: This function will format a moving head disc (HP 7902, 7905,7906,7920 & 7925). It will also verify each track.

EXAMPLE:

```
>5000 DEV 0,2,7,10,0
>5010 FMT 0
>5020 RUN
```

When the program begins execution the following is output to the console

```
 Begin Format

*End Format

 End of AID user program
```

* For a 7902 disc, the message "Begin Verifying Formatted Disc" will also appear on the console.

>5020

ID

FORMAL NAME: Initialize Data

FUNCTION NAME: ID

SYNTAX: >ID lun,buf(0)[,mask[,flag[,cylinder,head,sector]]]

PARAMETERS: lun - Logical unit number.

buf - Buffer into which data is read. This parameter must be any buffer AA(0)-NN(0) where AA-NN define buffer name and (0) sets an HP AID pointer to the first element in the buffer.

Note: For 7902 discs, buffer size should not exceed one sector (128 words).

cylinder - Disc parameters indicating starting location
head - of initialization operation. The heads are
sector - assumed to be positioned over the correct cylinder.

flag - Flags the track as:
- S - Spare
- P - Protected
- D - Defective
- N - Non-flagged

mask - Loads file mask on the 13037 controller only. The mask bits are:

Bits	Function
12	Incremental/decremental seek. If set and bit 15 is a 1, the cylinder address will be decremented when End-of-Cylinder; otherwise, incremented.
13	Allow sparing
14	Cylinder/surface mode. If set, a cylinder consists of all available surfaces; End-of-Cylinder is set when the last sector of the last surface has been transferred. In surface mode, End-of-Cylinder is set when the last sector of any surface has been transferred.

Bits	Function
15	Allow incremental/decremental seek.

Default mask is surface mode. Default flag is non-flagged. Default cylinder, head and sector is 0,0,0.

OPERATION: Initialize Data function will perform an initialize operation on all 79XX disc drives. The initialize operation will begin at the cylinder, head and sector designated and will continue until the word count of the buffer is exhausted. The designation of the cylinder, head and sector parameters will be accomplished in this function by an Address Record command to the disc controller.

EXAMPLE:

```
>5000 DEV 0,2,7,10,0
>5010 DB AA,6144,0
>5020 SEEK 0,10,0,0
>5030 ID 0,AA(0),3,D,815,0,0
>5040 SEEK 0,815,0,0
>5050 ID 0,AA(0),3,S,10,0,0
>5060 SEEK 0,10,0,0
>5070 RDI 0,AA(0),7
>5080 GOTO 5060
>5090 RUN
```

In this example a 7920A disc will seek to cylinder 10, head 0, sector 0, flag the entire track defective and place the address for its spare in the address field of each sector. It will then seek to a spare track (815) and flag it as a spare and write the address of the defective track in its address field. A loop is then set up to test the sparing feature.

NOTE: A 7902 disc drive does not have spare tracks but a defective track can be made invisible to the controller by flagging it defective and formatting the diskette. This process reduces the total number of available tracks on that surface.

IDI

===

FORMAL NAME: Initialize Data Immediate

FUNCTION NAME: IDI

SYNTAX: >IDI lun,buf(0)[,mask[,flag]]

PARAMETERS: lun - Logical unit number

buf - Buffer into which data is read. Buffer length determines word count of the write. This parameter must be any buffer AA(0)-NN(0) where AA-NN define buffer name and (0) sets an HP AID pointer to the first element in the buffer.

flag - Flags the track as:
S - Spare
P - Protected
D - Defective
N - Non-flagged

mask - Loads file mask on the 13037 controller only. The mask bits are:

Bits	Function
12	Incremental/decremental seek. If set and bit 15 is a 1, the cylinder address will be decremented when End-of-Cylinder; otherwise, incremented.
13	Allow sparing
14	Cylinder/surface mode. If set, a cylinder consists of all available surfaces; End-of-Cylinder is set when the last sector of the last surface has been transferred. In surface mode, End-of-Cylinder is set when the last sector of any surface has been transferred.
15	Allow incremental/decremental seek.

Default flag is non-flagged. Default cylinder, head, sector is 0,0,0. Default mask is surface mode.

OPERATION: This function will perform an initialize operation on a moving head disc. The internal disc address will be used as the starting point of the write.

EXAMPLE:

```
>5000 DEV 0,2,7,10,0
>5010 DB AA,6144,0
>5020 FOR C:=0 UNTIL 410
>5030 IS 0
>5040 IDI 0,AA(0)
>5050 IS 0,0,1,0
>5060 IDI 0,AA(0)
>5070 NEXT 5020
>5080 RUN
```

The above example formats the upper cartridge on a HP7905A disc.

IS

==

FORMAL NAME: Incremental Seek

FUNCTION NAME: IS

SYNTAX: >IS lun[,cylinder,head,sector]

PARAMETERS: lun - Logical unit number

cylinder - cylinder address

head - head address

sector - sector address

OPERATION: This function will do an initial seek to the address specified in the cylinder, head and sector parameters. Each time this command is executed it will increment the cylinder address. This function updates the internal disc address. Default cylinder, head and sector is 0,0,0.

NOTE: This function is only valid when used for operations on disc controlled by the 13037 controller.

This function does not increment a common cylinder table. It sets up a cylinder table based on the statment number making this function call. Only ten (10) IS function calls are allowed per program.

NOTE: All read, write, and verify operations update the 7906/20/25A disc internal address. Multiple use of this function can be used to position the 7906/20/25A discs before write and read operations. See example for the IT function for more information.

EXAMPLE:
```
>5000 DEV 0,3,2,15,0
>5010 FOR D:=1 UNTIL 2000
>5020 IS 0
>5030 NEXT 5010
>5040 RUN
```

The above example causes a moving head disc to execute one cylinder incremental seeks.

IT

FORMAL NAME: Increment Track

FUNCTION NAME: IT

SYNTAX: >IT lun[,cylinder,head,sector]

PARAMETERS: lun - Logical unit number

cylinder - cylinder address

head - head address

sector - sector address

OPERATION: This function will do an initial seek to the location specified in the cylinder, head and sector parameters. It will increment the head address by one each time this function is called. After the last head has been selected the next increment will proceed to the next cylinder. If the cylinder equals the last cylinder in the disc the function will seek to 0 and start over. This function updates the internal disc address.

This function is only valid when used in operations on disc controlled by the 13037 controller.

This function does not increment a common track table. It sets up a track table based on the statement number making this function call. A maximum of ten (10) IT function calls program are allowed.

EXAMPLE:

```
>5000 DEV 0,2,7,10,0
>5005 DB AA,6144
>5010 ASSIGN AA(0),(1536),%125252,%66666,%33333,%52525
>5020 DB BB,6144
>5030 FOR D:=1 UNTIL 4115
>5040 IT 0
>5050 WDI 0,AA(0)
>5060 DB BB,0
>5065 IT 0
>5070 RDI 0,BB(0),1
>5080 SCB 0,AA(0),BB(0),5
>5090 NEXT 5030
>5100 RUN
```

The above example indicates how this function may be used to test all surfaces on an HP 7920 disc. The increment track (IT) utilizes a separate counter for the write and read operation, thus assuring proper position of the heads before each write and read operation.

POLL

FORMAL NAME: Poll

FUNCTION NAME: POLL

SYNTAX: >POLL lun

PARAMETERS: lun - Logical unit number.

OPERATION: This function causes the HP13037 disc controller to resume polling.

EXAMPLE:

```
>5000 DEV 0,2,7,10,0
>5010 POLL 0
>5020 RUN
```

RC

==

FORMAL NAME: Recalibrate

FUNCTION NAME: RC

SYNTAX: >RC lun

PARAMETER: lun - Logical unit number.

OPERATION: This function performs a recalibrate operation on a 7906/20/25A moving head disc. At the completion of this operation the heads are located at cylinder 0.

EXAMPLE:

```
>5000 DEV 2,3,5,10,0
>5010 FOR A:=1 UNTIL 50
>5020 SEEK 2,822,0,0
>5030 RC 2
>5040 NEXT 5010
>5050 RUN
```

The above example will seek a HP7920A disc to cylinder 822 then recalibrate. This process is repeated fifty times.

RD

==

FORMAL NAME: Read Data

FUNCTION NAME: RD

SYNTAX: >RD lun,buf(0)[,mask[,cylinder,head,sector]]

PARAMETERS: lun - Logical unit number.

buf - Buffer into which data is read. This parameter must be any buffer AA(0)-NN(0) where AA-NN define buffer name and (0) sets an HP AID pointer to the first element in the buffer.

cylinder - starting cylinder address
head - starting head address
sector - startin section address

mask - Loads file mask on the 13037 controller only. The mask bits are:

Bits	Function
12	Incremental/decremental seek. If set and bit 15 is a 1, the cylinder address will be decremented when End-of-Cylinder; otherwise, incremented.
13	Allow sparing
14	Cylinder/surface mode. If set, a cylinder consists of all available surfaces; End-of-Cylinder is set when the last sector of the last surface has been transferred. In surface mode, End-of-Cylinder is set when the last sector of any surface has been transferred.
15	Allow incremental/decremental seek.

Mask default is surface mode. For the cylinder, head and sector parameter default is 0,0,0.

OPERATION: This function will perform a read operaton on the logical unit number indicated. This includes an Address Record command for 7906/20/25A discs and a Seek command for 7902 discs to pass the cylinder, head, and sector parameters.

At the completion of a 7906/20/25A disc read operation the internal disc address will be four sectors beyond the end of the last read operation. The buffering scheme of the 12745A interface, which has two 128 word buffers, reads three sectors worth of information before receiving an end of data from the CPU, which terminates the transfer. By the time the 12745A notifies the 13037A disc controller to stop the read, another sector has begun to be read. The disc has now read three sectors that were not transferred. The internal disc address is updated to point to the next sector. If a one sector read at cylinder 100, head 0, sector 0 were performed, the internal disc address will indicate cylinder 100, head 0, sector 4.

EXAMPLE:

```
>5000 DEV 0,3,5,10,0
>5010 RDB AA(0),6144
>5025 FOR C:=0 UNTIL 822
>5030 DB BB,6144,0
>5040 WD 0,AA(0),,C
>5050 RD 0,BB(0),1,C
>5060 SCB 0,AA(0),BB(0),4
>5075 NEXT 5025
>5080 RUN
```

This example writes, reads, and compares buffers of a random data pattern on surface zero of a 7920A disc.

RDA

FORMAL NAME: Request Disc Address

FUNCTION NAME: RDA

SYNTAX: >RDA lun

PARAMETER: lun - Logical unit number.

OPERATION: This function will return the current disc address stored in the controller. If a data error occurred it contains the address of the current sector; if not, it contains the address of the next logical sector. This function may be used to determine where an error occurred during a verify or any other function which terminates with a error. The address can be displayed on the console with DISP function.

At the completion of a 7906/20/25A disc read operation the internal disc address will be four sectors beyond the end of the last read operation. The buffering scheme of the 12745A interface, which has two 128 word buffers, reads three sectors worth of information before receiving an end of data from the CPU, which terminates the transfer. By the time the 12745A notifies the 13037A disc controller to stop the read, another sector has begun to be read. The disc has now read three sectors that were not transferred. The internal disc address is updated to point to the next sector. If a one sector read at cylinder 100, head 0, sector 0 were performed, the internal disc address will indicate cylinder 100, head 0, sector 4.

EXAMPLE:

```
>5000 DEV 0,2,7,10,0
>5010 RS 0
>5020 RDA 0
>5030 DISP 0,D
>5040 RUN
```

This example utilizes the RDA function to obtain the last address from a moving head disc.

RDI

==

FORMAL NAME: Read Data Immediate

FUNCTION NAME: RDI

SYNTAX: >RDI lun,buf(0)[,mask]

PARAMETERS: lun - Logical unit number.

buf - Buffer into which data is read. Length determines word count of read. This parameter must be any buffer AA(0)-NN(0) where AA-NN define buffer name and (0) sets an HP AID pointer to the first element in the buffer.

mask - Loads file mask on the 13037 controller only. The mask bits are:

Bits	Function
12	Incremental/decremental seek. If set and bit 15 is a 1, the cylinder address will be decremented when End-of-Cylinder; otherwise, incremented.
13	Allow sparing
14	Cylinder/surface mode. If set, a cylinder consists of all available surfaces; End-of-Cylinder is set when the last sector of the last surface has been transferred. In surface mode, End-of-Cylinder is set when the last sector of any surface has been transferred.
15	Allow incremental/decremental seek.

Default mask is surface mode.

OPERATION: This function will perform a read operation with the internal disc address designating the starting point of the read. This function updates the internal address.

EXAMPLE:

```
>5000 DEV 0,2,7,10,0
>5010 DB AA,128,0
>5020 RDI BB(0),128
>5040 WD 0,BB(0),7,120,2,0
>5045 SEEK 0,120,2,0
>5050 RDI 0,AA(0),7
>5060 SCB 0,BB(0),AA(0),3
>5070 GOTO 5020
>5080 RUN
```

This program writes, reads and compares the continually changing data for cylinder 120, head 2, sector 0.

RFS

==

FORMAL NAME: Read Full Sector

FUNCTION NAME: RFS

SYNTAX: >RFS lun,buf(0)[,cylinder,head,sector]

PARAMETERS: lun - Logical unit number.

buf - Buffer into which data is read. Buffer length determines word count of read. This parameter must be any buffer AA(0)-NN(0) where AA-NN define buffer name and (0) sets an HP AID pointer to the first element in the buffer.

cylinder - cylinder address
head - head address
sector - sector address

OPERATION: This function will execute a full sector read operation on a 7906/20/25A moving head disc. The heads are assumed to be positioned over the correct cylinder. The default cylinder, head, sector is 0,0,0. The length of the buffer determines the the word count of the read.

EXAMPLE:

```
>5000 DEV 0,2,7,10,0
>5010 DB AA,138,%52525
>5015 LET AA(0):=!80FE        (SYNC WORD)
>5020 DB BB,138,0
>5030 SEEK 0,123,0,0
>5040 WFS 0,AA(0),123,0,0
>5050 RFS 0,BB(0),123,0,0
>5060 SCB 0,AA(0),BB(0),3
>5070 RUN
```

In this example cylinder 123, head 0, sector 0 has had its address field written over by buffer AA. This track should be reformatted before proceeding.

RFSI

FORMAL NAME: Read Full Sector Immediate

FUNCTION NAME: RFSI

SYNTAX: >RFSI lun,buf(0)

PARAMETER: lun - Logical unit number.

buf - Buffer into which data is read. Buffer length determines word count of read. This parameter must be any buffer AA(0)-NN(0) where AA-NN defined as buffer name and (0) sets an HP AID pointer to the first element in the buffer.

This function will perform a full sector read operation on a 7906/20/25A moving head disc. The length of the buffer determines the word count of the read. The internal disc address will be used for the starting point.

EXAMPLE:

```
>5000 DEV 0,2,7,10,0
>5010 DB AA,138,%125252
>5015 LET AA(0):=!80FE          (SYNC WORD)
>5020 DB BB,138,0
>5050 WFS 0,AA(0),150,1,5
>5055 SEEK 0,150,1,5
>5060 RFSI 0,BB(0)
>5070 SCB 0,AA(0),BB(0),5
>5080 RUN
```

This example issues an address record (from WFS function) to address 150,1,5 on disc, writes and reads full sectors, then compares the data. The disc will require formatting after the use of the WFSI function.

RQST

===

FORMAL NAME: Request Status

FUNCTION NAME: RQST

SYNTAX: >RQST lun

PARAMETER: lun - Logical unit number.

OPERATION: This function will return two words of status from the disc controllers (status words 1 & 2). This status may be displayed using the DISP function. The status will be stored in buffer SS(0) AND SS(1) for any disc function error. This may be useful for user programs.

EXAMPLE:

```
>5000 DEV 0,3,0,10,0
>5010 SEEK 0,10,0,0
>5020 RQST 0
>5030 DISP 0,R
>5040 RUN
```

RS

==

FORMAT NAME: Random Seek

FUNCTION NAME: RS

SYNTAX: >RS lun

PARAMETER: lun - logical unit number.

OPERATION: This function will cause a moving head disc to seek randomly. This function will update the internal disc address.

EXAMPLE:

```
>5000 DEV 1,3,0,15,0
>5010 RS 0
>5020 GOTO 5010
>5030 RUN
```

RSA

FORMAL NAME: Request Sector Address

FUNCTION NAME: RSA

SYNTAX: >RSA lun

PARAMETER: lun - Logical unit number.

OPERATION: This function will return the logical sector address of the sector currently under the heads. This address may be displayed using the DISP function.

EXAMPLE:

```
>5000 DEV 0,2,7,10,0
>5010 RS 0
>5020 RSA 0
>5030 DISP 0,S
>5040 RUN
```

The following messages are output as a result of the above program executing:

```
Requested Sector Address for Logical Unit 0 is: 33

End of AID user program

>5040
```

RSYN

FORMAL NAME: Request Syndrome

FUNCTION NAME: RSYN

SYNTAX: >RSYN lun

PARAMETER: lun - Logical unit number.

OPERATION: This function will obtain a seven word syndrome from the HP13037 (disc controller). A request syndrome operation may be issued after any read or verify operation which terminates with a possible correctable data error. The seven words of information will be read into an internal buffer which may be displayed with the DISP function. The format of the syndrome returned is as follows:

WORD	DEFINITION
1	Status
2	Cylinder
3	Head/Sector
4	Displacement
5	Pattern 1
6	Pattern 2
7	Pattern 3

EXAMPLE:

```
>5000 DEV 0,2,7,15,0
>5010 DB AA,6144,%133333
>5020 DB BB,6144,0
>5030 FOR C:=1 UNTIL 5000
>5040 IT 0
>5050 WDI 0,AA(0),2
>5055 IT 0
>5060 RDI 0,BB(0),3
>5070 IF SS(0) = %7400 THEN 5090 (unit # in status word)
>5080 NEXT 5030
>5085 END            .Terminates program
>5090 RSYN 0
>5100 DISP 0,Y
>5110 RUN
```

NOTE: When attempting to use an entire cylinder on a read operation, the controller will attempt to read beyond the end of cylinder because of the buffering in the controller. The file mask must therefore be set to increment (file mask=1, 3, or 7).

This example writes and reads data on a HP7920 disc. If a correctable data error is detected, the syndrome is requested and displayed.

RWO

==

FORMAL NAME: Read With Offset

FUNCTION NAME: RWO

SYNTAX: >RWO lun,buf(0),mask,offset[,cylinder,head,sector]

PARAMETER: lun - Logical unit number.

buf - Buffer into which data is read. Buffer length determines word count of read. This parameter must be any buffer AA(0)-NN(0) where AA-NN define buffer name and (0) sets an HP AID pointer to the first element in the buffer.

mask - Loads file mask on the 13037 controller only. The mask bits are:

Bits	Function
12	Incremental/decremental seek. If set and bit 15 is a 1, the cylinder address will be decremented when End-of-Cylinder; otherwise, incremented.
13	Allow sparing
14	Cylinder/surface mode. If set, a cylinder consists of all available surfaces; End-of-Cylinder is set when the last sector of the last surface has been transferred. In surface mode, End-of-Cylinder is set when the last sector of any surface has been transferred.
15	Allow incremental/decremental seek.

offset - Contains cylinder offset and the separator clock information.

OFFSET WORD FORMAT

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
- - - - - - A D S - [   CYL OFFSET   ]
```

- = Don't care
A = Advance separator clock by 10 nanoseconds.
D = Delay separator clock by 10 nanoseconds.
S = Sign bit (offset direction)
CYL OFFSET = Range of offset (+63 to -63 moves heads from track center). Increment depends on drive type

cylinder - cylinder address

head - head address

sector - sector address

NOTE: Default for cylinder,head,sector is 0,0,0.

OPERATION: This function operates like a normal read except an offset word is transmitted to the drive before executing. The cylinder, head and sector parameters are passed to the disc controller with an Address Record command.

NOTE: This function is valid for 7906/20/25A moving head discs only. This function cannot read a spare track with offset. The offset information is lost when the controller seeks from a flagged defective track to its spare. This is a feature of the of the controller.

RWOI

==

FORMAL NAME: Read With Offset Immediate

FUNCTION NAME: RWOI

SYNTAX: >RWOI lun,buf(0),mask,offset

PARAMETER: lun - Logical unit number.

buf - Buffer into which data is read. Buffer length determines word count of read. This parameter must be any buffer AA(0)-NN(0) where AA-NN define buffer name and (0) sets an HP AID pointer to the first element in the buffer.

offset - Contains cylinder offset and the separator clock information.

```
                 OFFSET WORD FORMAT
                 ------------------
  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
  - - - - - - A D S - [   CYL OFFSET   ]


                 - = Don´t care
                 A = Advance separator clock by
                     10 nanoseconds.
                 D = Delay separator clock by
                     10 nanoseconds.
                 S = Sign bit (offset direction)
        CYL OFFSET = Range of offset (+63 to -63
                     moves heads from track center).
                     Increment depends on drive type
```

OPERATION: This function executes like a normal read except offset word is sent to the disc before executing. Heads are assumed to be positioned when using this function.

NOTE: This function is valid for a 7906/20/25A moving head disc only and it cannot read a spare track with offset. The offset information is lost when the controller seeks from a flagged defective track to its spare. This is a feature of the controller.

RWV

FORMAL NAME: Read Without Verify

FUNCTION NAME: RWV

SYNTAX: >RWV lun,buf(0)[,mask[,cylinder,head,sector]]

PARAMETER: lun - Logical unit number.

buf - Buffer into which data is read. Buffer length determines word count of read. This parameter must be any buffer AA(0)-NN(0) where AA-NN define buffer name and (0) sets an HP AID pointer to the first element in the buffer.

mask - Loads file mask on the 13037 controller only. The mask bits are:

Bits	Function
12	Incremental/decremental seek. If set and bit 15 is a 1, the cylinder address will be decremented when End-of-Cylinder; otherwise, incremented.
13	Allow sparing
14	Cylinder/surface mode. If set, a cylinder consists of all available surfaces; End-of-Cylinder is set when the last sector of the last surface has been transferred. In surface mode, End-of-Cylinder is set when the last sector of any surface has been transferred.
15	Allow incremental/decremental seek.

cylinder - cylinder address

head - head address

sector - sector address

Note: Default for cylinder,head,sector is 0,0,0.
Mask default is surface mode.

OPERATION: This function operates like a normal read function but does not verify the preceding sector. No address checking or sparing operations occur unless a track boundary is crossed during the operation.

NOTE: This function is valid only for 7906/20/25A moving head discs only.

EXAMPLE:

```
>5000 DEV 0,2,7,10,0
>5010 DB AA,6144,0
>5020 FOR C:=0 TO 410
>5030 RWV 0,AA(0),2,C,0,0
>5040 NEXT 5020
>5050 RUN
```

This example uses the RWV function to read one entire surface from an HP 7906A disc.

RWVI

FORMAL NAME: Read Without Verify Immediate

FUNCTION NAME: RWVI

SYNTAX: >RWVI lun,buf(0)[,mask]

PARAMETERS: lun - Logical unit number.

buf - Buffer into which data is read. Buffer length determines word count of read. This parameter must be any buffer AA(0)-NN(0) where AA-NN define buffer name and (0) sets an HP AID pointer to the first element in the buffer.

mask - Loads file mask on the 13037 controller only. The mask bits are:

Bits	Function
12	Incremental/decremental seek. If set and bit 15 is a 1, the cylinder address will be decremented when End-of-Cylinder; otherwise, incremented.
13	Allow sparing
14	Cylinder/surface mode. If set, a cylinder consists of all available surfaces; End-of-Cylinder is set when the last sector of the last surface has been transferred. In surface mode, End-of-Cylinder is set when the last sector of any surface has been transferred.
15	Allow incremental/decremental seek.

NOTE: Default mask is surface mode.

OPERATION: This function operates like a normal read operation but does not verify the preceding sector. No address checking or sparing operations occur unless a track boundary is crossed during the operation. The starting point of the read will be taken from the internal disc address.

EXAMPLE:

```
>5000 DEV 0,2,7,10,0
>5010 DB AA,128,0
>5020 RS 0
>5030 RWVI 0,AA(0)
>5040 GOTO 5020
>5050 RUN
```

This example randomly seeks and uses the RWVI function to read one sector of information.

SEEK

==

FORMAL NAME: Seek

FUNCTION NAME: SEEK

SYNTAX: >SEEK lun[,cylinder,head,sector]

PARAMETERS: lun - Logical unit number.

cylinder - Disc parameters

head - for the moving

sector - head discs. Default is 0,0,0.

NOTE: Default cylinder,head,sector is 0,0,0.

OPERATIONS: This function will cause a disc to position its heads over the specified cylinder. This function also updates the internal disc address.

EXAMPLE:

```
>5000 DEV 0,2,7,10,0
>5010 SEEK 0
>5020 SEEK 0,200,3,23
>5030 SEEK 0,405,1,5
>5040 GOTO 5010
>5050 RUN
```

This example executes seek operations on a HP7906/20/25 disc.

SFM

FORMAL NAME: Set File Mask

FUNCTION NAME: SFM

SYNTAX: >SFM lun,mask

PARAMETERS: lun - Logical unit number.

mask - Loads file mask on the 13037 controller only. The mask bits are:

Bits	Function
12	Incremental/decremental seek. If set and bit 15 is a 1, the cylinder address will be decremented when End-of-Cylinder; otherwise, incremented.
13	Allow sparing
14	Cylinder/surface mode. If set, a cylinder consists of all available surfaces; End-of-Cylinder is set when the last sector of the last surface has been transferred. In surface mode, End-of-Cylinder is set when the last sector of any surface has been transferred.
15	Allow incremental/decremental seek.

OPERATION: This function will set the file mask on the HP13037 disc controller from bits 8-15 of the mask parameter. When the disc controller is first powered up the mask is set to surface mode, where no sparing or automatic seeking is performed.

NOTE: Default for all functions using the file mask parameter is 0 (surface mode).

EXAMPLE:

```
>5000 DEV 2,2,7,15,3
>5010 SFM 2,7
>5020 RUN
```

This example loads the file mask on the disc controller with a value of 7.

SKRD

==

FORMAL NAME: Seek Read Data

FUNCTION NAME: SKRD

SYNTAX: >SKRD lun,buf(0)[,mask[,cylinder,head,sector]]

PARAMETERS: lun - Logical unit number.

buf - Buffer into which data is read. Buffer length determines word count of read. This parameter must be any buffer AA(0)-NN(0) where AA-NN define buffer name and (0) sets an HP AID pointer to the first element in the buffer.

mask - Loads file mask on the 13037 controller only. The mask bits are:

Bits	Function
12	Incremental/decremental seek. If set and bit 15 is a 1, the cylinder address will be decremented when End-of-Cylinder; otherwise, incremented.
13	Allow sparing
14	Cylinder/surface mode. If set, a cylinder consists of all available surfaces; End-of-Cylinder is set when the last sector of the last surface has been transferred. In surface mode, End-of-Cylinder is set when the last sector of any surface has been transferred.
15	Allow incremental/decremental seek.

cylinder - cylinder address

head - head address

sector - section address

NOTE: Default for cylinder,head,sector is 0,0,0.

OPERATION: This function will perform a seek to the specified location and read that data into the specified buffer. The internal disc address is updated by this function.

EXAMPLE:

```
>5000 DEV 0,2,7,10,2
>5010 DB AA,128,0
>5020 SKRD 0,AA(0),2,10,1,2
>5030 RUN
```

SKWD

FORMAL NAME: Seek write Data

FUNCTION NAME: SKWD

SYNTAX: >SKWD lun,buf(0)[,mask[,cylinder,head,sector]]

PARAMETERS: lun - Logical unit number.

but - Buffer into which data is read. Buffer length determines word count of read. This parameter must be any buffer AA(0)-NN(0) where AA-NN define buffer name and (0) sets an HP AID pointer to the first element in the buffer.

mask - Loads file mask on the 13037 controller only. The mask bits are:

Bits	Function
12	Incremental/decremental seek. If set and bit 15 is a 1, the cylinder address will be decremented when End-of-Cylinder; otherwise, incremented.
13	Allow sparing
14	Cylinder/surface mode. If set, a cylinder consists of all available surfaces; End-of-Cylinder is set when the last sector of the last surface has been transferred. In surface mode, End-of-Cylinder is set when the last sector of any surface has been transferred.
15	Allow incremental/decremental seek.

cylinder - cylinder address

head - head address

sector - sector address

NOTE: Default for cylinder,head,sector is 0,0,0.

OPERATION: This function will issue a seek to the specified location and read the data into the specified buffer. The internal disc address is updated.

NOTE: This function is only valid for 79XX discs that are controlled by the 13037 controller.

EXAMPLE:

```
>5000 DEV 0,2,7,10,0
>5010 RDB AA(0),4096
>5020 DB BB,4096,0
>5030 FOR C:=1 UNTIL 100
>5040 RAND I
>5050 LET A:=I MOD 411
>5060 LET B:=I MOD 4
>5070 LET C:=I MOD 48
>5080 SKWD 0,AA(0),7,A,B,C
>5090 SKRD 0,BB(0),7,A,B,C
>5100 IF SS(0) = %7400 THEN 5120
>5105 SCB 0,AA(0),BB(0),3
>5110 NEXT 5030
>5115 END                 .Terminates program
>5120 RSYN 0
>5130 DISP 0,Y
>5150 RUN
```

This example uses the SKWD and SKRD functions to test a HP7906 disc.

VER

==

FORMAL NAME: Verify

FUNCTION NAME: VER

SYNTAX: >VER lun,secount[,cylinder,head,sector]

PARAMETERS: lun - Logical unit number.

secount - Numbers of sectors to be verified.

cylinder - cylinder address
head - starting head address
sector - starting sector address

NOTE: Default for cylinder,head,sector is 0,0,0.

OPERATION: This function will verify the data on a number of sectors on a moving head disc.

EXAMPLE:

```
>5000 DEV 1,2,7,10,3
>5010 SFM 1,7
>5020 FOR I:=0 TO 410
>5030 SEEK 1,I,0,0
>5040 VER 1,192,I,0,0
>5050 NEXT 5020
>5060 RUN
```

This example verifies one cylinder at a time until the entire 7906 disc is checked.

VERI

==

FORMAL NAME: Verify Immediate

FUNCTION NAME: VERI

SYNTAX: >VERI lun,secount

PARAMETERS: lun - Logical unit number.

secount - Number of sectors to be verified.

OPERATION: This function will verify the data on a number of sectors on a moving head disc. The starting point will be the internal disc address.

EXAMPLE:

```
>5000 DEV 4,2,7,15,2
>5010 DB AA,128,%155555
>5020 RS 4
>5030 WDI 4,AA(0)
>5040 VERI 4,1
>5050 GOTO 5050
>5060 RUN
```

This example seeks to random locations, writes and verifies one sector.

WD

==

FORMAL NAME: Write Data

FUNCTION NAME: WD

SYNTAX: >WD lun,buf(0)[,mask[,cylinder,head,sector]]

PARAMETERS: lun - Logical unit number.

buf - Buffer into which data is read. Buffer length determines word count of read. This parameter must be any buffer AA(0)-NN(0) where AA-NN define buffer name and (0) sets an HP AID pointer to the first element in the buffer.

mask - Loads file mask on the 13037 controller only. The mask bits are:

Bits	Function
12	Incremental/decremental seek. If set and bit 15 is a 1, the cylinder address will be decremented when End-of-Cylinder; otherwise, incremented.
13	Allow sparing
14	Cylinder/surface mode. If set, a cylinder consists of all available surfaces; End-of-Cylinder is set when the last sector of the last surface has been transferred. In surface mode, End-of-Cylinder is set when the last sector of any surface has been transferred.
15	Allow incremental/decremental seek.

cylinder - cylinder address

head - head address

sector - sector address

NOTE: Default for cylinder,head,sector is 0,0,0.

OPERATION: This function will write the data specified by the buf parameter beginning at the location specified by the cylinder, head, sector parameters. An Address Record command will be issued to the disc controller to pass the cylinder, head and sector parameters. This is required to simulate the Sleuth format. This function updates disc internal address.

EXAMPLE:

```
>5000 DEV 0,3,2,10,0
>5010 RDB AA(0),6144
>5020 FOR I:=0 TO 410
>5040 WD 0,AA(0),7,I,0,0
>5050 NEXT 5020
>5060 RUN
```

This example fills one surface of a HP7906 disc with random data.

WDI

FORMAL NAME: Write Data Immediate

FUNCTION NAME: WDI

SYNTAX: >WDI lun,buf(0)[,mask]

PARAMETERS: lun - Logical unit number.

buf - Buffer into which data is read. Buffer length determines word count of read. This parameter must be any buffer AA(0)-NN(0) where AA-NN define buffer name and (0) sets an HP AID pointer to the first element in the buffer.

mask - Loads file mask on the 13037 controller only. The mask bits are:

Bits	Function
12	Incremental/decremental seek. If set and bit 15 is a 1, the cylinder address will be decremented when End-of-Cylinder; otherwise, incremented.
13	Allow sparing
14	Cylinder/surface mode. If set, a cylinder consists of all available surfaces; End-of-Cylinder is set when the last sector of the last surface has been transferred. In surface mode, End-of-Cylinder is set when the last sector of any surface has been transferred.
15	Allow incremental/decremental seek.

OPERATION: This function will write data on a moving head disc. The internal disc address will designate the starting point of the write operation. This function updates the internal disc address.

EXAMPLE:

```
>5000 DEV 0,2,7,10,0
>5010 DB AA,128,%155555
>5020 RS 0
>5030 WDI 0,AA(0),7
>5040 GOTO 5020
>5050 RUN
```

WFS

==

FORMAL NAME: Write Full Sector

FUNCTION NAME: WFS

SYNTAX: >WFS lun,buf(0)[,cylinder,head,sector]

PARAMETERS: lun - Logical unit number.

buf - Buffer into which data is read. Buffer length determines word count of read. This parameter must be any buffer AA(0)-NN(0) where AA-NN define buffer name and (0) sets an HP AID pointer to the first element in the buffer.

cylinder - starting cylinder address

head - starting head address

sector - starting sector address

NOTE: Default for cylinder,head,sector is 0,0,0.

OPERATION: This function will write a full sector on a moving head disc. Note the disc should be formatted after this operation. This function writes over the address field.

NOTE: This function is only valid for 79XX discs that are controlled by the 13037 controller and it updates the internal address of the disc.

EXAMPLE:

```
 >5000 DEV 1,3,7,10,2
 >5010 DB AA,138,%125252
*>5015 LET AA(0):=!80FE,AA(1):=400,AA(2):=!305
 >5020 DB BB,138,0
 >5040 WFS 1,AA(0),400,1,5
 >5050 RFS 1,BB(0),400,1,5
 >5060 SCB 1,AA(0),BB(0),5
 >5070 RUN
```

* AA(0) equals the sync word.

This example performs a single write full sector and a read full sector on a HP7906/20/25A disc. The buffers are then checked to verify the data.

WFSI

==

FORMAL NAME: Write Full Sector Immediate

FUNCTION NAME: WFSI

SYNTAX: >WFSI lun,buf(0)

PARAMETERS: lun - Logical unit number.

OPERATION: Buffer into which data is read. Buffer length determines word count of read. This parameter must be any buffer AA(0)-NN(0) where AA-NN define buffer name and (0) sets an HP AID pointer to the first element in the buffer.

OPERATION: This function will perform a full sector write operation on a 7906/20/25A moving head disc only. The internal disc address will be used and updated as the starting point.

EXAMPLE:

```
>5000 DEV 0,2,7,10,0
>5010 RDB AA(0),138
>5020 DB BB,138,0
>5030 SEEK 0
>5034 RFSI 0,AA(0)
>5037 LET AA(5):= -AA(5)
>5040 WFSI 0,AA(0)
>5050 RDI 0,BB(0)
>5060 RUN
```

This example uses the WFSI function to force a possible correctable data error negating the sixth word of buffer AA and using the original CRC value.

3.5 LINE PRINTER I/O STATEMENTS

The following statement descriptions pertain strictly to I/O operations concerning the line printer (2608 or 2631 specifically).

The status for each device is stored in buffer SS as follows:

SS(7) Line printer status (I/O and operator)

RP

==

FORMAL NAME: Ripple Print

FUNCTION NAME: RP

SYNTAX: >RP lun,linelength

PARAMETERS: lun - Logical unit number.

linelength - Number of columns defining the area of ripple print.

OPERATION: This function will write write a ripple pattern on the lun indicated and continue until stopped with CNTRL Y or until 32767 lines have been printed.

EXAMPLE:

```
>5000 DEV 0,3,4,10,0
>5010 RP 0,132
>5020 RUN
```

This is an example of a 132 column ripple print on a lineprinter.

WD

FORMAL NAME: Write Data

FUNCTION NAME: WD

SYNTAX: >WD lun,buf(0),mode,linelength

PARAMETERS: lun - Logical unit number

buf - Buffer containing write data. This parameter must be any buffer AA(0)-NN(0) where AA-NN define buffer name and (0) sets an HP AID pointer to the first element in the buffer.

linelength - Length of each line.

mode - Format character as indicated below:

CODE	BIT 9	10	11	12	13	14	15	COMMAND
0	0	0	0	0	0	0	0	SUPPRESS SPACE **
1	0	0	0	0	0	0	1	SINGLE SPACE
2	0	0	0	0	0	1	0	DOUBLE SPACE
63	0	1	1	1	1	1	1	63 SPACES
64	1	0	0	0	0	0	0	Chan 1(Top of form)*
65	1	0	0	0	0	0	1	Chan 2(bottom of form)*
66	1	0	0	0	0	1	0	Chan 3(Single space forms step-over)*
67	1	0	0	0	0	1	1	Chan 4(Double space forms step-over)
68	1	0	0	0	1	0	0	Triple space forms step-over)*
69	1	0	0	0	1	0	1	Next one-half page*
70	1	0	0	0	1	1	0	Next one-fourth page*
71	1	0	0	0	1	1	1	Next one-sixth page*

*Assigned according to HP programming standards.

** Not allowed for 2608A. Results are indeterminate.

OPERATION: This function will write data from the specified buffer, perform the indicated mode command over the length specified by the linelength parameter and transmit this data to the lun specified.

EXAMPLE:

```
>5000 DEV 0,3,2,10,0
>5010 DB AA,66,12345
>5015 FOR C:= 1 UNTIL 100
>5020 WD 0,AA(0),1,132
>5030 NEXT 5015
>5040 RUN
```

3.6 MAGNETIC TAPE I/O STATEMENTS

The following statement descriptions pertain strictly to I/O operations concerning the 7970E Magnetic Tape.

The status for each device is stored in buffer SS as follows:

SS(8) Magnetic Tape status (byte 1 and 2)

SS(9) Magnetic Tape status (byte 3)

BSF

FORMAL NAME: Backspace File

FUNCTION NAME: BSF

SYNTAX: >BSF lun

PARAMETER: lun - Logical unit number

OPERATION: This function issues a backspace file to a magnetic tape unit.

EXAMPLE:

```
>5000 DEV 1,2,4,10,0
>5005 FOR B:=0 UNTIL 10
>5010 GAP 1
>5020 WFM 1
>5030 NEXT 5005
>5035 FOR C:=0 UNTIL 9
>5040 BSF 1
>5050 NEXT 5035
>5060 REW 1
>5070 RUN
```

This example demonstrates how a BSF function might be utilized in a user program. Eleven file marks are written on the tape then the tape unit is backspaced 10 file marks.

BSR

FORMAL NAME: Backspace Record

FUNCTION NAME: BSR

SYNTAX: >BSR lun

PARAMETER: lun - Logical unit number

OPERATION: This function will cause the magnetic tape unit to backspace one record from its present position.

EXAMPLE:

```
>5000 DEV 0,2,4,10,0
>5010 RDB AA(0),128
>5015 FOR C:=1 UNTIL 10
>5020 WD 0,AA(0)
>5030 NEXT 5015
>5035 FOR D:=1 UNTIL 9
>5040 BSR 0
>5050 NEXT 5035
>5060 REW 0
>5070 RUN
```

This example writes 128 records then backspaces through nine of them.

FSF

==

FORMAL NAME: Forward Space File

FUNCTION NAME: FSF

SYNTAX: >FSF lun

PARAMETER: lun - Logical unit number

OPERATION: This function will move the tape forward to the next file on the tape.

EXAMPLE:

```
>5000 DEV 0,2,3,15,0
>5010 RDB AA(0),4000
>5015 FOR C:=0 UNTIL 10
>5020 WD 0,AA(0)
>5030 WFM 0
>5040 NEXT 5015
>5050 REW 0
>5055 FOR D:=0 UNTIL 9
>5060 FSF 0
>5070 NEXT 5045
>5080 RUN
```

This example writes 11 records of random data with a file mark after each, rewinds the tape, then forward spaces to 10 of them.

FSR

FORMAL NAME: Forward Space Record

FUNCTION NAME: FSR

SYNTAX: >FSR lun

PARAMETER: lun - Logical unit number

OPERATION: This function will move the tape forward one record.

EXAMPLE:

```
>5000 DEV 0,3,3,15,2
>5010 RDB AA(0),8000
>5015 FOR C:=1 UNTIL 10
>5020 WD 0,AA(0)
>5030 NEXT 5015
>5040 REW 0
>5045 FOR D:=1 UNTIL 8
>5050 FSR 0
>5060 NEXT 5045
>5070 REW 0
>5080 RUN
```

This example writes 11 records of random data on tape, rewinds the tape, then forward spaces to 9 of them.

GAP

FORMAL NAME: Gap

FUNCTION NAME: GAP

SYNTAX: >GAP lun

PARAMETER: lun - Logical unit number

OPERATION: This function will write a gap on the specified magnetic tape unit.

EXAMPLE:

```
>5000 DEV 1,2,3,20,2
>5010 GAP 1
>5015 REW 0
>5020 RUN
```

This example erases a 3 inch portion of magnetic tape and rewinds.

RD

FORMAL NAME: Read Data

FUNCTION NAME: RD

SYNTAX: >RD lun,buf(0)

PARAMETER: lun - Logical unit number

buf - Buffer into which data will be read. This parameter must be any buffer AA(0)-NN(0) where AA-NN define buffer name and (0) sets an HP AID pointer to the first element in the buffer.

OPERATION: This function will perform a read operation on the lun specified.

EXAMPLE:

```
>5000 DEV 0,2,4,10,1
>5010 RDB AA(0),4000
>5020 DB BB,4000,0
>5030 WD 0,AA(0)
>5040 REW 0
>5050 RD 0,BB(0)
>5060 SCB 0,AA(0),BB(0),3
>5070 REW 0
>5080 RUN
```

This example indicates how a read data operation may be performed on magnetic tape. This program writes one 4000 word record on magnetic tape then reads and checks the data.

REW

==

FORMAL NAME: Rewind

FUNCTION NAME: REW

SYNTAX: >REW lun

PARAMETER: lun - Logical unit number

OPERATION: This function will issue a rewind command to the magnetic tape unit specified.

EXAMPLE:

```
>5000 DEV 0,4,2,15,0
>5005 DB &AA,100
>5010 ASSIGN &AA(0),(32),%123,%377,%345,0
>5015 FOR C:=1 UNTIL 10
>5020 WD 0,&AA(0)
>5030 NEXT 5015
>5040 REW 0
>5050 RUN
```

This example writes eleven 128 word records of data then rewinds the tape unit with the REW function.

REWOFF

==

FORMAL NAME: Rewind And Reset

FUNCTION NAME: REWOFF

SYNTAX: >RST lun

PARAMETER: lun - Logical unit number

OPERATION: This function will rewind and reset the specified magnetic tape unit.

EXAMPLE:

```
>5000 DEV 0,3,3,15,0
>5010 REWOFF 0
>5020 RUN
```

This example places mag tape unit 0 offline.

RRB

==

FORMAL NAME: Read Record Backward

FUNCTION NAME: RRB

SYNTAX: >RRB lun, buf(0)

PARAMETER: lun - logical unit number

buf - Buffer into which data will be read. This buffer must be any buffer AA(0) - NN(0) where AA-NN defines buffer name and (0) sets an HP AID pointer to the first element in the buffer.

OPERATION: This function will read from the last element (byte) in the record toward the first. The bits within the bytes will remain the same if the record is read backward or forward.

SELU

===

FORMAL NAME: Select Unit

FUNCTION NAME: SELU

SYNTAX: >SELU lun,unit

PARAMETER: lun - Logical unit number

unit - Temporary unit select in the range of 0 to 3. Does not affect the logical unit number. The unit does not have to be on-line.

OPERATION: This function will select the unit specified in the unit parameter.

EXAMPLE:

```
>5000 DEV 3,2,4,10,3
>5010 SELU 3,1
>5020 RUN
```

This example defines the mag tape unit as 3 but it selects unit 1.

WD

==

FORMAL NAME: Write Data

FUNCTION NAME: WD

SYNTAX: >WD lun,buf(0)

PARAMETER: lun - Logical unit number

buf - Buffer that contains the write data. This parameter must be any buffer AA(0)-NN(0) where AA-NN define buffer name and (0) sets an HP AID pointer to the first element in the buffer.

OPERATION: This function will execute a write operation on the specified unit.

EXAMPLE:

```
>5000 DEV 0,3,4,10,0
>5010 RDB AA(0),8000 (statement takes approx. 45 sec)
>5015 FOR C:= 1 UNTIL 50
>5020 WD 0,AA(0)
>5030 NEXT 5015
>5040 REW 0
>5050 RUN
```

This example writes records of 8000 random words of data 50 times on mag tape unit 0.

WFM

==

FORMAL NAME: Write File Mark

FUNCTION NAME: WFM

SYNTAX: >WFM lun

PARAMETER: lun - Logical unit number

OPERATION: This function will write a file mark on the specified unit.

EXAMPLE:

```
 >5000 DEV 0,2,3,10,0
 >5010 DB FF,6000,%22222
*>5020 WD 0,FF(0)
 >5030 WFM 0
 >5040 REW 0
 >5050 FSF 0
 >5060 REWOFF 0
 >5070 RUN
```

This example writes a file mark on mag tape, rewinds and then forward spaces to the file mark.

* This parameter must be any buffer AA(0) through NN(0) where AA through NN define buffer name and (0) sets an HP AID pointer to the first element in the buffer.

RESERVED BUFFERS AND VARIABLES

APPENDIX A

The following lists those buffers and variables reserved for use by the Sleuth Simulator functions along with a brief explanation of assignment.

BUFFER	USAGE
OO	RESERVED
PP	RESERVED
QQ	RESERVED
RR	Contains Magnetic tape and line printer commands.
SS	Contains status command and information as follows: Last status in 0 and 1 Status request command in 2 Expected status in 3 and 4 Don't care masks in 5 and 6 Printer status in 7 Magnetic Tape status in 8 and 9
TT	Contains disc syndrome information
UU	Channel program buffer for general usage
VV	Channel program buffer primarily for obtaining disc address and other general usage.
WW	Used for passing commands and information - usage is as follows: 0 = command 1-3, 6-8, and 16 = command or information 4 = disc cylinder information 5 = Head and sector information 9 = counter 10 = DSJ information 11 = Sleuthsm variable usage indicator

12 = Suppress status flag

13-15 = disc parameters (cyl,hd,sect)

17 = SCB error count

18 = suppress status flag

19 = Pause on error flag

20 = DSJ information

21 = # of sect/cyl

22 = command

23 = cyl information

24 = head and sector information

25 = Head count for verify error

26 Sector count for verify error

27 = Next sector after verify error

28 and 29 = internal disc address (head & sector)

30-32 = beginning cylinder, head, and sector address for a read or verify

33 = final head address

34 = final sector address

35 = SOUT counter

40-49 = IS function line number

50-59 = IS function cylinder number

60-69 = DS function line number

70-79 = DS function cylinder number

80-89 = IT function line number

90-99 = IT function head number

100-109 = IT Function cylinder number

XX Channel program buffer that is variable in length and is built every time a channel program is executed.

YY	CPVA buffer (See Appendix B in the 7906,7920,7925 Verifier Manual for more information).
ZZ	This buffer contains DEV function parameters as follows:

0-7 = Logical unit number

8 = Ripple print counter

9 = Printer page length counter

10-17 = Channel number

18 = Identify code of executing device

19 = FMT cyl counter

20-27 = device number

28 = number of bytes per page (WD function)

29 = Data overrun counter for 12745A

30-37 = number of errors

38 = Top of form indicator (WD function)

40-47 = Unit number

50-57 = EXP status 2 work

60-67 = Mask words

70-77 = 13037 disc type

80-87 = device indentification code

STRING BUFFER	USAGE
&WW	RESERVED
&XX	RESERVED
&YY	Print buffer
&ZZ	Contains information for user error reporting as follows:

0 = variable name

1-2 = buffer name

3 = 0, 1, or X for printing status

VARIABLE	USAGE
O - Z	Initially these variables act as pointers to set up the logical unit table. This table contains all necessary parameters for a particular device usage and is located in buffer ZZ. Once the variable information stored in buffer ZZ, all are then available for general usage by the Sleuth simulator functions. NOTE: Attempted use of these variables by a user will adversely affect the users program.

IOMAP Diagnostic

IOMAP
Reference Manual

HEWLETT hp PACKARD

Manual Part No. 30070-90041 Print Date: Sept 1978

CONTENTS

GENERAL INFORMATION

SECTION I

1.0 INTRODUCTION

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. Table 6-1 provides a list of those identify codes that are recognized by IOMAP.

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.

1.1 REQUIRED HARDWARE

Standard HP3000/33 system with 128KB.

1.2 REQUIRED SOFTWARE

Diagnostic/Utility disc

1.3 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 which this involves circuitry in every device connected to the bus; not just the controller and device being identified.

OPERATING INSTRUCTIONS

SECTION II

2.0 INTRODUCTION

The following paragraphs describe the method used to load and execute the standard operation for IOMAP. Also described is the optional operating modes of the IOMAP program.

2.1 STANDARD MODE OF OPERATIONS

Perform the following steps to obtain a listing of the current I/O configuration of the system.

1. Load the Diagnostic/Utility System flexible disc per the procedure found in the Diagnostic/Utility System Manual.

2. Once the DUS has displayed its title message and prompt, enter IOMAP and then press RETURN. IOMAP will respond with its title message and prompt as shown below:

```
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)
>
```

3. Enter 'GO' or 'GO,1' and the IOMAP program will do an identify to all devices and then display the system I/O configuration table, as shown below, and then return control to the DUS.

2.2 OPTIONAL MODE OF OPERATIONS

In addition to the I/O configuration display, IOMAP has three selectable test secions available to the operator when in the optional mode.

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.

To enter the optional mode, enter 'GO 1' or 'GO 1,1'. In response, the system displays the following message:

```
           IOMAP                  Optional Test Sections

      Test Section 2                   Identify
      Test Section 3                   Self-Test
      Test Section 4                   Loopback

Enter Desired Test Section(s)
   ('LC' to list commands)

>
```

At this point you enter the term 'TEST' and then the test section number that you want to execute. For example: Entering 'TEST 2' will execute the Identify portion of the IOMAP program.

NOTE: To exit the Optional Mode of operation, enter 'TEST' without a test section number and the 'GO' selections will be displayed again.

2.3 MESSAGES

Several kinds of messages may be displayed by IOMAP:

General messages:	Request action from the operator or report results of commands.
	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).

Sample I/O table:

```
                 SYSTEM I/O CONFIGURATION

-----------------------------------------------------------------
>Control panel switch settings: Channel=2  Device=6
>System console is device 2 on channel 3
-----------------------------------------------------------------
Channel 2     ID=!0      General I/O Channel  (GIC)
 Device 3     ID=!1234    XXX Device
 Device 5     ID=!4321    YYY Device
-----------------------------------------------------------------
Channel 4      ID=!1     Async. Data Comm. Channel (ADCC)
 Devices 0-3  ID=!4080  ADCC MAIN   (Code=1,2)
 Devices 4-7  ID=!4080  ADCC EXTEND (Code=1,2)
-----------------------------------------------------------------
Channel 7 (GIC)
 Device 0     ID code=!9999 Device responds but ID code undefined*
-----------------------------------------------------------------


    Note: The devices on the ADCC MAIN and ADCC EXTEND are
          not individually identifiable. The ADCC responds
          to the IDENTIFY command with !4080.


    Code: 1 Implies -- No Loopback
          2 Implies -- No Self-Test
```

End of pass n

where "n" indicates the number of passes that have been made to this point.

TEST DESCRIPTIONS

SECTION III

3.0 INTRODUCTION

The following paragraphs describe each test section operation and possible error situations and messages.

3.1 TEST SECTION 1 — I/O Configuration 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.

3.2 TEST 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 enter -2 to EXIT this test section.

?
```

The AID prompt character ´?´, awaits the operator´s response:

Upon entry of a legal channel and device number the test executes

Upon entry of ´-2´ the utility returns the operator to:

```
     "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)
>
```

At this time the operator may enter the 'TEST' and RETURN to exit the IOMAP utility, select another test section, or enter 'GO' to continue.

3.3 TEST 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 channel and device number separated by a comma,
           or enter -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
```

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.

3.4 TEST SECTION 4 — HP-IB 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, Enter -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 HAS 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.

ERROR INTERPRETATION

SECTION IV

4.0 INTRODUCTION

The following paragraphs discuss possible error situations and corrective action.

4.1 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 6.0; check for stuck bits.

(2) Device is not supported on system. Check current 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 6.0; check for stuck bits.

(3) Channel is newly supported on system and IOMAP copy is not up-to-date.

4.2 OPTIONAL MODE ERRORS

"Device X does not exist on Channel Y."
"Enter a 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 (Manual Section 3).

Possible causes:

(1) Operator Error

(2) Channel or Device is intermittently failing to Identify.

REFERENCE TABLES

SECTION V

5.0 INTRODUCTION

The following manual section may be used as a quick reference for identify codes recognized and supported by either an HP 300 or HP 3000 Series 33 system.

5.1 SUPPORTED CHANNELS

IOMAP currently recognizes the following channels:

GIC
ADCC MAIN
ADCC MAIN with EXTENDER

Figure 5.1 shows the Channel ID Codes that are recognized as devices.

5.2 SUPPORTED DEVICES

IOMAP currently recognizes the following devices:

```
ID code           DEVICE
-----------------------------------------------------------------
!0081....7902 Flexible Disc Drive
!0001....7910 Fixed Disc
!0082....12745 HP-IB Adapter for 13037 Disc Controller
!2001....2608 Line Printer
!2002....2631 Serial Printer
!2080....Integrated Display System (IDS)
!4080....ADCC
!6000....GIC as device
!8000....PMPI
```

Figure 5.1 - Device ID Codes Recognized

-hp-