HP 3000 Computer System

AID Diagnostic Language

Reference Manual

HEWLETT PACKARD

COMPUTER SYSTEMS DIVISION, 19447 PRUNERIDGE AVE., CUPERTINO, CALIF. 95014

Part No. 30341-90006
Index No. 3HDWR.070.30341-90006

Printed in U.S.A 5/81

LIST OF EFFECTIVE PAGES

The List of Effective Pages gives the date of the current edition and of any pages changed in updates to that edition. Within the manual, any page changed since the last edition is indicated by printing the date the changes were made on the bottom of the page. Changes are marked with a vertical bar in the margin. If an update is incorporated when an edition is reprinted, these bars are removed but the dates remain.

PRINTING HISTORY

New editions are complete revisions of the manual. Update packages, which are issued between editions, contain additional and replacement pages to be merged into the manual by the customer. The date of the title page of the manual changes only when a new edition is published. When an edition is reprinted, all the prior updates to the edition are incorporated.

CONTENTS

SECTION I - GENERAL INFORMATION

Paragraph Page

SECTION II - ESSENTIALS OF AID

Paragraph Page

CONTENTS (Continued)

SECTION III - AID COMMANDS

SECTION IV - AID STATEMENTS (NON I/O)

CONTENTS (Continued)

SECTION IV (Con't)

SECTION V - SPECIAL CHARACTERS

CONTENTS (Continued)

SECTION V (Con't)

SECTION VI - OPERATORS

SECTION VII - RESERVED VARIABLES

CONTENTS (Continued)

SECTION VII (Con't)

Paragraph

Page

SECTION VIII - AID STATEMENTS (I/O - NON CHANNEL PROGRAM)

Paragraph

Page

CONTENTS (Continued)

SECTION IX - AID STATEMENTS (CHANNEL PROGRAM TYPE)

SECTION X - FUNCTION STATEMENTS

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 HP 3000 HP-IB version computer systems containing at least 256K bytes of memory, with a device to load AID and a keyboard console for operator interaction.

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

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

For documentation purposes, throughout this manual, characters outputed 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 manual makes reference to the Diagnostic/Utility System III which is documented in the Diagnostic/Utility System III Reference Manual, part no. 30341-90005 of this diagnostic manual set.

1.1 SPECIAL KEYS

RETURN	Must be pressed after every command and or statement. It terminates the line and causes the Console to return to the first print position.
linefeed	Advances the Console one line.
CTRL	When pressed simultaneously with another key, converts that key to a control character that is usually non-printing.

CTRL H (Bs) or BACKSPACE — Deletes the previous character in a line. The cursor is moved one space to the left.

CTRL X (Cn) or DELETE ENTRY — Cancels the line currently being typed. Three exclamation marks, a Return and Linefeed are issued to the Console (Note - May not apply to all Console types).

CTRL Y (Em)
or
ATTENTION — Suspends AID program execution, reports the statement number currently executing and prompts (>). See the PAUSE command for further action. CTRL Y has no significance in the entry mode except during LISTing where it causes the listing to terminate.

1.2 PROMPT CHARACTERS

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

> The prompt character for AID; an AID command or statement is expected.

? User input is expected during execution of an INPUT(B) statement.

?? Further input is expected during execution of an INPUT statement.

!!! A full line has been deleted with CTRL X (Note- May not apply to all Console types).

1.3 LOADING THE AID DIAGNOSTIC PROGRAM

(1) Bring up the Diagnostic/Utility System III (DUSIII) from a DUSIII Tape.

(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, refer to Section III.

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, RETURN 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. (Refer to paragraph 2.10.)

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, refer to Sections IV, VIII, IX, and X.

1.4.3 Changing or Deleting a Statement

If an error is made before RETURN is pressed, the error can be corrected with CTRL H, (Hc) or the line may be concelled with CRTL X (Xc). Refer to paragraph 1.1. After RETURN 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.

Refer to the LIST Command (paragraph 3.13) 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
----
```

Refer to the RUN Command (paragraph 3.21) 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 S:=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. Refer to EP Command (paragraph 3.7) 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 as are needed can be entered.

The previous sample program to average 5 numbers can be documented with several comments by using the insert line function:

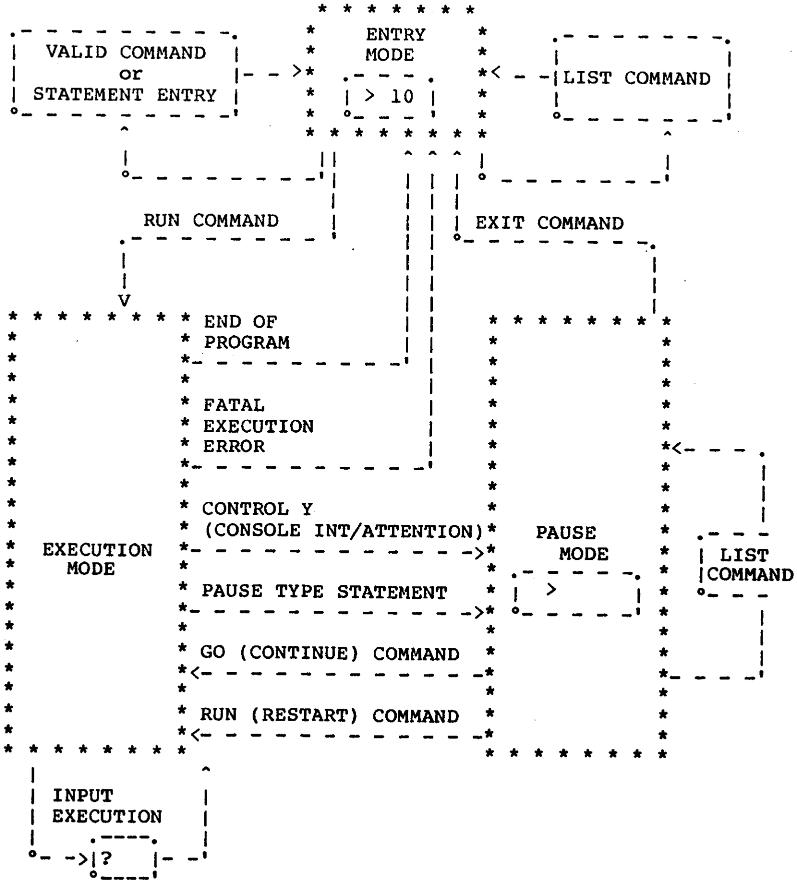
```
> 40   5. THIS PROGRAM AVERAGES
----
> 40   7. 5 NUMBERS
----
> 40   10 INPUT A,B,C,D,E  .GET VALUES
----
> 40   25.S CONTAINS THE AVERAGE.
----
```

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 (paragraph 4.4) or the period (.) Special Character (paragraph 5.1) for further details.

1.10 AID OPERATOR MODE STATE DIAGRAM



ESSENTIALS OF AID | SECTION II

2.0 INTRODUCTION

This section explains some of the ground rules for handling constants, variables, and strings. Discussions are also included 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 CONTSTANTS

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   |
|           |           |
o-----------------------'
```

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 binary 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 binary 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"
```

Note that using relational operators does not work in this type application. However, consider the evaluation of special relational operators: (Refer to Special Relational Operators (Section VI) 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, refer to Section VI.

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 did not 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, refer to 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 - 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. (Refer to paragraph 2.8.3.)

(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, PURGE, INC and MODIFY. Program alteration is not allowed, but the operator may display any LIST data.

For further explanations, refer to the operator mode state diagram (paragraph 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, 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 paragraph 2.11.

If the program does not 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) 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 error is detected in a statement or command just inputed, 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 is not 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 better

```
> 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 words divided by 128 [,revision]

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 tape, a Data file "filename" which will be the "number of words long" for tape. Refer to the DUSIII Reference Manual, part no. 30341-90005 for further details.

EXAMPLE(S):

```
> 10 CREATE TEST,4  (creates the Data file TEST
----                 with a length of 512 words.
```

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
-----
(Control Y)

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
-----
(Control Y)

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
-----
(Control Y)

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
----
(Control Y)

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 does not 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 DUSIII.

ALLOWED IN: Pause Mode or Entry Mode

EXAMPLE(S):

```
> 50  RUN
----
(Control Y)

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 directory. For further information, refer to the DUSIII Reference Manual, part no. 30341-90005.

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

EXAMPLE(S):

```
> 10 LF     (Refer to DUSIII Reference Manual 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.

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
-----

(Control Y)

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
-----

(Control Y)

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 in-
                  crements 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
-----                          TAPE VIA DUSIII)

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 TAPE 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.

```
(Control Y)
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 DUSIII directory. Refer to the DUSIII Reference Manual 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
-----

(Control Y)

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 tape via DUSIII (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 tape for the file, the message "End od Tape" is displayed. Since going to DUSIII will cause the current AID program to be lost, follow this recovery procedure:

(1) Insert another Diagnostic/Utility tape that has more space

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

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

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
-----
(Control Y)

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
-----
(Control Y)
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 ENTR
----            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
-----
(Control Y)

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
-----
(Control Y)

Break in Statement 40
---------------------
> SNPS
-
```

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

3.28 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, in this section. The format for each statement explanation is:

OPERATION NAME: General phrase of what the statement does.

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

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

EXAMPLES: One or more examples using the statement.

4.1 ASSIGN

OPERATION NAME: Assign Data to Buffer

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

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 in data buffer (element+N-1). If repeat factor is included, the data pattern is repeated repeated factor times. Datal 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 did not compare (see INDEX under Reserved Variables).

The length of the compare is in words (limit 32,767) if comparing data buffers and in 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 consumming 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 (always 0 for DUSIII tape) is the sector number (for DUSIII disc) 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, 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 Media being used and must be a valid filename as specified by the DUSIII Reference Manual, part no. 30341-90005.

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)] variablel [,variableN]

DESCRIPTION: Generates random integers (-37,768 to 32,767) from an argument (optional) and stores them into variables specified (variabll 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 clock's resolution 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 files may be written into by this statement. (Refer to the DUSIII Reference Manual, part no. 30341-90005 for additional 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),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%007657004839
```

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
              ----
              > 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
-----

(Control Y)
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

OPERATION NAME: Suspend Execution

SYMBOL: Control Y(Em)

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

AID Diagnostic Language

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. (Also see the spe- inclusion character (/).)

```
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  |
                o---------------'

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

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         |
          o-------------------------'

          .-------------------------.
          |     | 15 bits shifted   |  Bits Lost
          |  *  |                   |-------------> ASR
          |     |      Right        |
          o-------------------------'
     * 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   |  |
 |   o-----------------'  |
 o-----------"-----------'

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

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.
```

RESERVED VARIABLES | SECTION VII

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  =Type of the file
Bit 12/15 =Class of the file
```

(Refer to the DUSIII Reference Manual.)

INITIALIZED TO: Zero

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

```
10 DB &AA<10,"XYZ "
--
20 FILENAME &AA(0)
--
30 LET A:=FILEINFO AND %100000 LSR 15
--
40 LET B:=FILEINFO AND %360 LSR 4
--
50 LET C:=FILEINFO AND %17
--
60 PRINT &AA(0,2);" file ","PROTECT BIT=";A;2;
--
70 PRINT "Class=";B;2;"Type=";C;2;"Length=";FILELEN
--
80 RUN
--
XYZ file
PROTECT BIT=1  Class=1  Type=1  Length=256
```

7.6 FILELEN

OPERATION NAME: File Length

MNEMONIC: FILELEN

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

INITIALIZED TO: Zero

EXAMPLE(S): See FILEINFO Reserved Variable example.

7.7 GOPARAM1/GOPARAM2/GOPARAM3

OPERATION NAME: Go Parameters

MNEMONIC: GOPARAM1/GOPARAM2/GOPARAM3

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

INITIALIZED TO: Zero

EXAMPLE(S):

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

        -or-

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

7.8 INDEX

OPERATION NAME: Buffer Compare Indicator

MNEMONIC: INDEX

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

INITIALIZED TO: Zero

EXAMPLE(S):

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

7.9 INPUTLEN

OPERATION NAME: Last Input character Length

MNEMONIC: INPUTLEN

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

INITIALIZED TO: Zero

EXAMPLE(S):

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

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

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

7.10 MAXMEMORY

OPERATION NAME: Maximum Buffer Area

MNEMONIC: MAXMEMORY

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

INITIALIZED TO: Memory space available prior to RUN time

EXAMPLE(S):

```
> 20   IF MAXMEMORY < 4000 THEN 50
----
> 30   DB AA, 4000
----
> 40   GOTO 60
----
> 50   DB AA, 2000
----
(IF THE DB AT 30 WAS EXECUTED THEN MAXMEMORY
 WOULD THEN EQUAL MAXMEMORY - 4000)
           ---------
```

7.11 NEWTEST

OPERATION NAME: Test Command Indicator

MNEMONIC: NEWTEST

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

INITIALIZED TO: Not altered at RUN time

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

```
> 10   IF NEWTEST THEN 30
----
> 20   LET SECTIONS 1:=!FFDF .CLEAR SECTION 11
       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 |   |   |   |   |   |          |        |
o-----------------------------------------------'
```

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. 10 sec.)
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 10 seconds), the user program may set TIMEOUT equal to -1. To increase the default timeout by N times 10 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.

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>l)

.
.
.

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 %13).

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 %13).

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 may be 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 %13), 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.

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.

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.

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 does not 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.

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 at 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.

9.16 WDMAR

OPERATION NAME: Write DMA Record

MNEMONIC: WDMAR XX(N), BC[,R]

DESCRIPTION: This statement executes the Write DMA Record channel program instruction. The parameters are the same as WR except the modifier and DC are deleted.

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.

```
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            |
o--------------------------------------'
```

```
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 ensure all parameter types are matched. Any parameter may be defaulted i.e., excluded, except the NAME type when it is 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.

(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: VALUE1,VALUE2,NAME1,VALUE3,NAME2,VALUE4,NAME3,NAME4

** See the respective examples on the following pages which display rule usage.

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 re-
                          stored  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)
```

NOTES