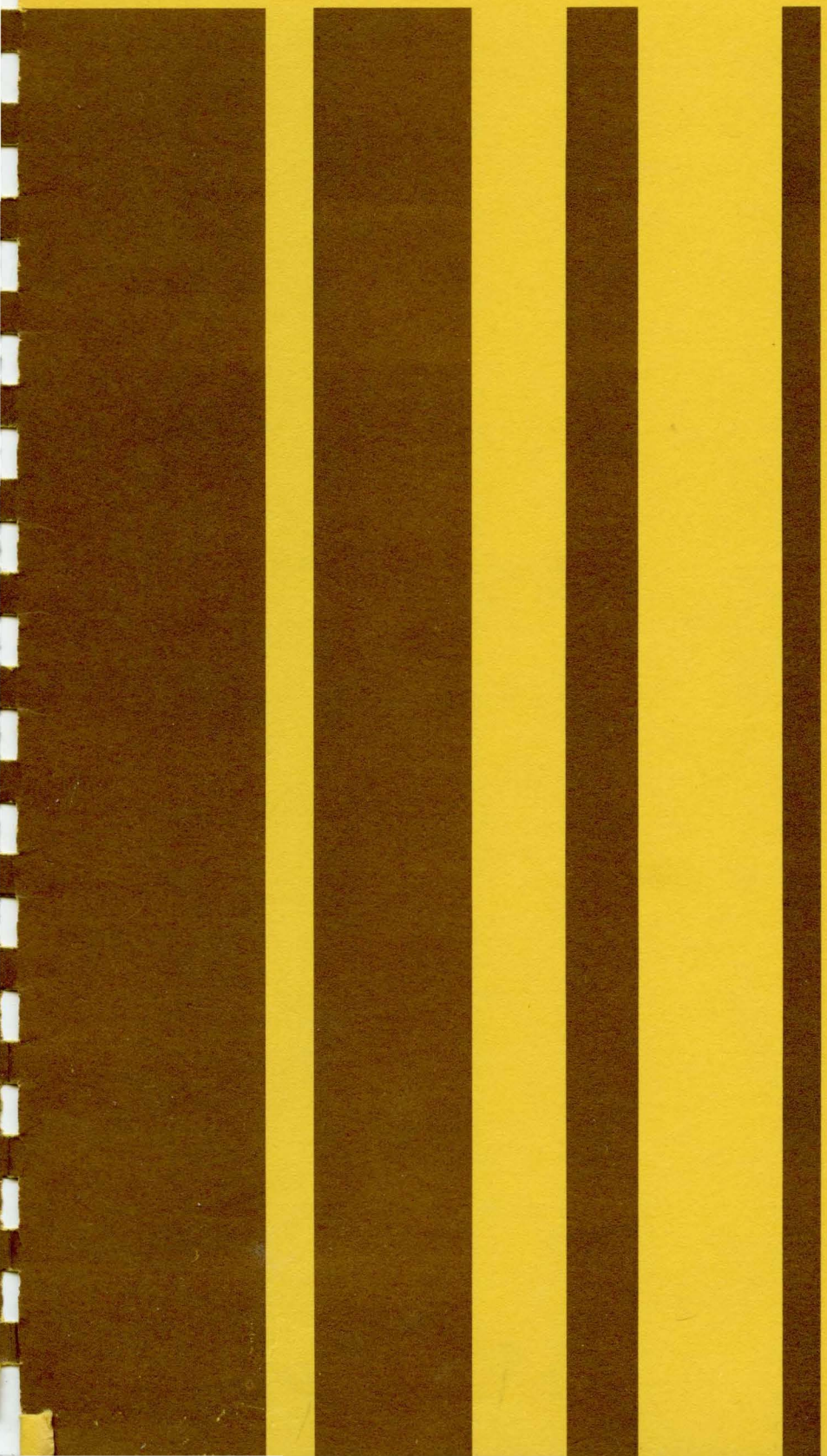
PROGRAMMER'S REFERENCE MANUAL
ADAGE GRAPHICS TERMINAL
VOL. I (SOFTWARE)



adage

adage

ADAGE GRAPHICS TERMINAL

PROGRAMMER'S REFERENCE MANUAL

Volume I

ADAGE, INC.
1079 Commonwealth Avenue
Boston, Massachusetts 02215

FOREWORD

This manual is provided to the prospective Adage Graphics Terminal user to provide detailed information about the system software. It is the first volume of a two-volume set which comprises the Programmer's Reference Manual. Descriptions contained herein are subject to change without notice.

The actual manual from which this material was obtained is loose-leaf and updated on a regular basis.

CONTENTS

PROGRAMMING REFERENCE MANUALS

GENERAL

This routine converts a real or integer number to its absolute value. If the argument is of type real, the function ABS is of type real; if the argument is of type integer, the function IABS is of type integer. The argument is specified in the parameter statement immediately following the ABS or IABS call. The result is in the AR register when control is returned to the user program at the instruction immediately following the parameter statement.

Name:

ABS

Purpose:

Convert the real argument A to the absolute value.

Calling Sequence:

```
JPSR    ABS
 Ø       A
```

Name:

IABS

Purpose:

Convert the integer argument I to absolute value.

Calling Sequence:

```
JPSR    IABS
 Ø       I
```

CORE REQUIREMENTS

10_3 words

EXECUTION TIME

50 μs

adage

ADEPT

ADAGE EXTENDABLE PROGRAM TRANSLATOR

Programmer's Reference Manual

Revision G

July 1969

TABLE OF CONTENTS

INTRODUCTION

This manual is intended to be a reference guide for programmers using the ADEPT symbolic programming language. The manual contains descriptions of statement formats and the initial set of action operators for composing statements in the ADEPT language.

The manual is organized to introduce the topics of symbolic input format and action operators. The section on symbolic input format summarizes the concepts of statements, comments, and statement evaluation. The following section analyzes action operators by function and usage in output generation, value generation, and control and definition. Since all processing by the ADEPT translator is governed by action operators, the remainder of the manual is devoted to detailed expansion of this paragraph (ACTION OPERATORS).

The user may wish to consult the following documents for further information:

1. ADEPT PROGRAMMING INSTRUCTION MANUAL
2. ADEPT SOFTWARE MAINTENANCE MANUAL
3. ADEPT PROGRAM LISTING

HARDWARE REQUIREMENTS

AMBILOG 200 subsystems and software required by basic ADEPT are as follows:

Version 1 - (SCU1-P1, ACC1-P1, OPC1-P1, DME1-P8 and AMRM versions 1 or 2) or (DPR1-P2 and AMRM versions 3, 4, 5 or 6).

Version 2 - (SCU1-P1, ACC1-P1, OPC1-P1, DME1-P16 or P32 and AMRM versions 1 or 2) or (DPR1-P3 or P4 and AMRM versions 3, 4, 5 or 6).

Version 4 - (DPR2-P3 or P4 and AMRM versions 3 or 4).

Version 5 - (AGT with memory size 16K or 32K and AMRM versions 11 or 12).

NOTE: The capability of extending the assembler to include user-defined action operators is not available on configurations with DME1-P8, DPR1-P2, or DPR2-P2.

SYMBOLIC INPUT FORMAT

A. Statements

Input to ADEPT is nearly format-free, consisting of a series of "statements", each of which is terminated by a "statement terminator" character. When first loaded into memory, ADEPT treats semicolons and carriage return characters as "statement terminators". However, for convenience, the option of using or not using the carriage return character as a "statement terminator" is provided by two ADEPT "action operators". (See NOCARRET and CARRET descriptions in section on CONTROL AND DEFINITIONS). There is no necessary correspondence between statements and physical lines; both multiple-statement lines and multiple-line statements are acceptable in ADEPT input.

The ADEPT user is free to use any convenient tabular or columnar statement format, such as the following:

```
TAG:        MDAR'X        COUNT        [COMMENT
```

where space or tab characters separate the location tag, instruction, address, and comment. The superficial appearance of specific fields for the four above-named items on the line is, however, purely artificial.

B. Comments

Within a statement, the string of input characters is scanned from left to right and processed on a symbol-by-symbol basis. Comments may be placed anywhere in the input text to increase the readability of the source program. These comments are deleted at the lowest scanning level and are otherwise ignored by the translator. Each such comment begins with the character "[" (left bracket), and is terminated by another "[" or by a "statement terminator" character, whichever comes sooner. The "[" characters are part of the comment and are deleted from the input text, hence cannot be used for any other purpose.

C. Statement Evaluation

Each statement, after being stripped of comments, is scanned from left to right by the translator and evaluated.

In scanning a statement the translator replaces all symbols which represent character strings by their value texts. Prior to extension these consist of Macro names, dummy arguments, and certain action operators.

The action performed by the translator in evaluating a scanned statement is specified by any operators it contains, as modified by the arguments (symbols, values, or character strings) of these operators.

> NOTE: In this manual the OPC characters °, ¢, ±, and BKSP are equivalent to the TTY characters [,], ↑ and ← respectively.

ACTION OPERATORS

The actions specified by operators fall into the following general classes:

A. Output Generation

Operators are initially provided for generating two forms of output during the translation.

1. Message Outputs

Message outputs are character strings to be output on the assigned system control unit.

2. Object Program Outputs

Object Program Outputs are either 3ø-bit machine words and their address relocation information, or control commands to the loader specifying any loading, allocating, naming, referencing, or linking actions required prior to object program execution.

B. Value Generation

These operators generate either character strings for further scanning or address and word values for use as "terms" in expression evaluation.

C. Control/Definition

These operators control subsequent translation by setting flags, adding code to the translator, or making entries in the translator's tables.

A. Translation Time Console Text Input

The operator

TYPEIN

will cause ADEPT to accept and process a string of characters input from the console typewriter until a carriage return character is typed. ADEPT will then process the string (not including the carriage return) as if it had occured in the input text at the point of the TYPEIN operator. It is up to the source program to indicate to the console operator that a type-in is required before using the TYPEIN operator.

B. Translation Time Message Outputs

The input sequence

TYPEOUT CHAR STRING CHAR

where CHAR is any character except "\" (back-slash), " " " (double quote), or " # " (number sign) selected by the programmer, causes the STRING to be typed out on the console typewriter. Tabs, spaces, and C/R (in NOCARRET mode) are ignored when they occur between TYPEOUT and the first occurrence of CHAR. The STRING is scanned for substitution and expansion during output. CHAR may not occur within STRING.

The statement

TYPEOCT EXPRESSION

causes the value of the EXPRESSION to be typed out as an octal integer. (See section on VALUE GENERATION for Expression Evaluation). If the value of EXPRESSION is negative, a single zero will be typed.

The statement

TYPEDEC EXPRESSION

causes the value of the EXPRESSION to be typed out as a decimal integer.

C. Text Output to Object Program

The operator TEXT followed by a string "quoted" (i.e., preceded and followed) by a character causes the AMOS code representation of the string to be inserted into successive words of the output program. If the last word of the output is not filled, the characters will be left-justified and the word filled with zeros. Macro names will cause the strings to be expanded unless themselves quoted (see String Quotes).

The form of this input is

TEXT CHAR Character String CHAR

where CHAR is any character except "\" (back-slash), " " " (double quote), or "#" (number sign) selected by the programmer. Tabs, spaces, and C/R (in NOCARRET mode) are ignored when they occur between TEXT and the first occurrence of CHAR. CHAR may not occur within the Character String.

The statement

STRING CHAR Character String CHAR

has the same function as the TEXT statement described above except that it forces the Character String inserted into the object program to be terminated by a null character ($\emptyset\emptyset_8$) (i.e., if the length of the string is an integral multiple of five characters, an additional 3Ø-bit word containing zeros will be inserted after the text). The same restrictions on CHAR also apply in this statement.

The operator ASCII (Version 5 only) followed by a string "quoted" by a character, causes assembly of a packed USASCII string in the format used by the LCG1 Character Generator. The form of this input is:

ASCII CHAR Character String CHAR

where CHAR is any character except " " (back-slash), " " " (double quote), "#" (number sign), or "@" (at sign) selected by the programmer. Tabs, spaces, and C/R (in NOCARRET mode) are ignored when they occur between ASCII and the first occurrence of CHAR. CHAR may not occur within the Character String unless used as a special character generator after "@" (see below).

As the USASCII character set includes 128_{10} characters, the characters from the AMOS ATEXT set are insufficient in number to generate all ASCII characters. Provision is made to generate any ASCII character by the following sequence:

@nnn

where nnn is a three-digit octal integer indicating the desired ASCII character code ($\emptyset\emptyset\emptyset$ through 177). Any of the 128_{10} ASCII characters may be generated in this manner.

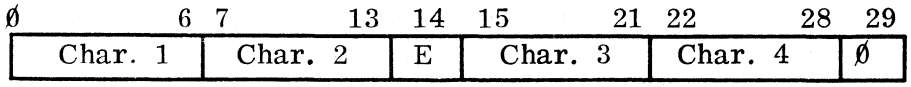
In order to provide a shorter sequence for commonly used characters, all the standard AMOS ATEXT characters may be used in the string with the exception of "\" (back-slash), " " " (double quote), "#" (number sign), "@" (at sign), and "[" (left bracket). These characters and other ASCII characters used by the LCG1 Character Generator may be generated by the following sequences:

"		@'
#		@+
@		@@
<		@(
>		@)
[		@]
^	(circumflex)	@V

_ (underline)	@ –
\ (back-slash)	@/
Raise line (GS)	@R
Lower line (FS)	@L
Brighter (US)	@B
Dimmer (RS)	@D
Expand size (DC3)	@E
Shrink size (DC2)	@S
Italics control (BS)	@I
CR (Carriage Return)	@C
LF (Line Feed)	@F
HT (Position "X")	@X
VT (Position "Y")	@Y
NUL (Null character)	@N

A sequence of "@" (at sign) followed by a character which is not found in the list above will be ignored by the ASCII operator.

The words inserted into the output program by the ASCII operator contain four characters, each in the following format:



Bit 14, the End code, is set to "1" on the last word generated by the string and is "Ø" on all other words. If the last character generated for the string does not completely fill the last word, the characters are left justified and NUL characters (code $ØØØ_8$) are inserted in the empty character positions.

D. Control Output to Object Program

Any assembly control operations which affect the loading or execution-time allocating, naming, relocating, referencing, or linking of programs will automatically generate any object-program control words needed.

The control operators and statements are individually described in the section on CONTROL AND DEFINITIONS.

E. Value Output to Object Program

Any expression whose value is left after a statement has been fully evaluated is used to generate a word for the object program output, together with any modifying codes to properly relocate its address at load-time. After any output of object code, the value of the "current location" counter is appropriately incremented.

WORD-VALUE GENERATION

A. Expressions

Expressions are composed of operators and their operands. The operators substitute word or address-values for character strings and symbols (terms), and operate on these values to yield word values. An expression is terminated by a comma (,) or if at the end of a statement, by a "statement terminator".

B. Expression Evaluation Operators

The value of an expression is obtained by a left-to-right scan in which terms are evaluated, and then combined with the previous value of the expression according to the expression's "action operators". The operators initially provided by the ADEPT translator for expression evaluation are:

+	arithmetic plus
-	arithmetic minus
'	logical exclusive "OR"
+	logical inclusive "OR"
&	logical product "AND"
*	integer multiplication (versions 4 and 5 only)
/	integer division (versions 4 and 5 only)
//	integer division remainder (versions 4 and 5 only)
space	is ignored unless it is the only separator between two terms, when it acts as an arithmetic plus
tab	same as space
!B	rotates the expression value 1 bit left and then acts as a space
!K	rotates the expression value 6 bits left and then acts as a space
!H	rotates the expression value 15 bits left and then acts as a space

C. Operator Terms

The character "." (period) is an operator yielding the value of the current location counter when used as a term in an expression.

D. Numerical Terms

Each numerical term is either an octal integer or a decimal integer. Each octal integer is a string of up to ten digits from the set $\emptyset, 1, 2, \ldots 7$. Each decimal integer is a string of up to nine digits from the set $\emptyset, 1, 2, \ldots 9$, whose value must be less in magnitude than $2^{29} = 536,87\emptyset,912_{10}$. Leading zeros may be omitted from both octal and decimal integers.

The choice of radix for numerical terms is controlled by three ADEPT operators. (See OCTAL & DECIMAL in section on CONTROL AND DEFINITIONS).

E. Symbolic Terms

Each symbol consists of a string of up to ten characters from the set ., A, B, C, . . . , Z, $\emptyset$, 1, 2, . . . 9, of which at least one character must be from the set A, B, C, . . . Z. During expression evaluation, word and address values for the symbolic terms are found by table look-up, and these values are used in obtaining the value of the expression.

F. Subexpression Terms

An unterminated expression enclosed in parentheses is called a subexpression, and may be used as a term in another expression or subexpression. Up to 31_{10} levels of nested subexpressions are permitted in ADEPT input statements.

CHARACTER-STRING-VALUE GENERATION

A. String Substitution

When a symbol defined as the name of a "Macro" is encountered during statement scanning, it is replaced by the character string constituting the "body" of the Macro definition with substitutions from the argument list which follows the Macro name. When a symbol is encountered which is currently a "dummy" argument of the Macro body being expanded or of the indefinite repeat being repeated, the symbol is replaced by its corresponding actual argument (character string).

B. Macro Call

The character string

NAME (STR1, STR2, STR3,...) or
NAME STR1, STR2, STR3,... ;

where NAME is the name of a defined Macro, is replaced during scanning with the STRING corresponding to the body of Macro NAME, but with the "place markers" replaced by the "actual argument" strings STR1, STR2, STR3, etc., inserted instead of the "place markers" having the same ordinal numbers. (See "Macro Definition" under CONTROL AND DEFINITIONS).

Any "place markers" defined in the Macro body for which no substrings are given in the Macro call are simply deleted.

If the Macro argument list is enclosed by parentheses, these outer parentheses are removed before substitution. Argument strings (STRn) may themselves contain argument lists with commas, and matched parentheses.

If the Macro call terminates a statement (i.e., is followed immediately by a "statement terminator" character), the parentheses enclosing the argument list may be omitted:

NAME STR1, STR2, STR3,... ;

If the first character (other than SPACE or TAB) following the Macro NAME is not left parenthesis, then the argument string will consist of all characters up to but not including the next end-of-statement character.

If a Macro is called which had no dummy arguments in its definition form, no argument list is required and the body of the called Macro will simply replace the Macro name.

C. Nested Macro Calls

A Macro call may appear within a Macro definition. Such a nested call may even be a call to the Macro being defined, so long as the rule for self-nesting is followed (i.e., there must be at least one path through the Macro that does not call itself). If arguments are handed down (by using a dummy argument in a Macro call imbedded in a Macro definition) the rules for argument substitution again apply.

D. Symbol Definition within Macros

Care must be taken in the defining of address tags within Macros to ensure that multiple symbol definitions do not arise when these Macros are expanded. Concatenation and numerical text generation may be used for this purpose.

E. Numerical Text Generation

The operator "#" (number sign) causes the symbol which immediately follows to be replaced by a string of lead-zero suppressed octal digits corresponding to the value of the symbol. If the symbol is undefined or has the value zero, a single zero character will be inserted. The operator "##" (double number sign) causes the following symbol to be replaced by a string (signed if negative) of decimal digits corresponding to the value of the symbol. These operators can be used to generate unique symbols for address tags within Macros, etc. The symbol following the "#" or "##" can not be generated by Macro or argument substitution, concatenation, nor by subsequent "#" or "##" operators.

F. String Quotes

A string in an argument list or a text-generating statement is taken literally (i.e., without expansion) if it is preceded and followed by a " " " (double quote) character.

G. Concatenation

The character "\" (back-slash) is used to concatenate two character strings — i.e., to place them end-to-end without an intervening character. If one such string is replaced by another during input scanning, the result will be a new string which may include a new symbol at the position where the "\" character originally appeared.

H. Literal Characters

Certain characters in the argument string of a Macro call (such as a semi-colon, comma, left and right parentheses) have definite meanings in the format of the string. In order to include these characters in an argument, the character "%" (per cent sign) causes the next character following it to be scanned without considering any possible format implications. To include the character "%" in the argument, one must include it twice in the input -- "%%".

I. Definite Repeat

A portion of the input character string will be repeated a specified number of times if the following sequence is encountered:

```
REPEAT EXPRESSION
STRING
ENDR
```

The EXPRESSION will be evaluated to produce a count and the STRING repeated the corresponding number of times. If the count is zero or negative, the STRING will be ignored.

The STRING may, of course, include symbols, statements, expressions, etc., subject to the same limitations with regard to symbol definitions and special characters as in the body of a Macro definition.

J. Indefinite Repeat

An alternate method of repeating an input string is provided by the indefinite repeat, which has the form:

```
IREPEAT DUMMY, (ARG1,ARG2,...) or
  IREPEAT DUMMY, ARG1,... ;
STRING
ENDI
```

When the IREPEAT operator is encountered, the following STRING is repeated once for each argument in the list ARG1, ARG2, etc. In each repetition, the corresponding "actual argument" string ARGn is inserted in place of the "dummy argument" string DUMMY wherever it occurs within the repeated STRING. The argument string following DUMMY is of the same form as the argument string for a Macro call.

K. Conditional Translation

Conditional translation is provided in basic ADEPT (in addition to the indefinite repeat) by the following sequences:

```
IFZERO EXPRESSION
STRING
ENDC
```

When the above sequence is encountered, the expression is evaluated, and if the value is logical (positive) zero, then the following STRING is scanned. Otherwise, the STRING is ignored.

```
IFNEGATIVE EXPRESSION
STRING
ENDC
```

When the above sequence is encountered, the expression is evaluated, and if the value is negative, then the following STRING is scanned. Otherwise, the STRING is ignored.

```
IFPASS1
STRING
ENDC
```

The above sequence causes the STRING to be scanned in the first translation pass and ignored in the second.

```
IFSAME          (ARG1)  (ARG2)
STRING
ENDC
        or
IFSAME      ARG1; ARG2;
STRING
ENDC
```

When either of the above sequences is encountered, the two arguments, ARG1 and ARG2 (possibly null), are compared on a character-by-character basis and if equal, the STRING is scanned. Otherwise the STRING IS ignored. Each argument must have the same form as a Macro argument list and must be either bracketed by parentheses or terminated by a statement terminator. Spaces or tabs between the arguments are ignored.

```
IFZERO      EXPRESSION
STRING1
ELSE        STRING2
ENDC
```

The operator ELSE occurring in any conditional statement causes the scanning or ignoring of the string to be reversed. In the above form, either STRING1 or STRING2 will be scanned, depending on whether the value of EXPRESSION is zero or non-zero, respectively. This operator may be placed in any conditional statement.

Conditional statements may be nested to any depth permitted by the available storage at assembly time.

Other conditional translation statements can easily be built into ADEPT using "action operator" definitions.

L. Termination of String Substitution

STOP

This operator terminates the current string substitution operation (Macro call, repeat or indefinite repeat) in which it occurs.

M. Nesting Limit

Macros, repeats, and argument substitutions may be nested in a depth of 63_{10}.

CONTROL AND DEFINITIONS

The following ADEPT inputs control various aspects of the object program: assembly, definition and/or loading.

A. Radix Control

OCTAL

This operator causes the following numerical terms to be taken as octal, until the appearance of a DECIMAL operator.

DECIMAL

This operator causes the following numerical terms to be taken as decimal, until the appearance of an OCTAL operator.

NOTE: The use of a period (". ") immediately following a numeric form causes that term to be regarded as decimal no matter what the current radix. The current radix for subsequent numeric terms is not changed.

B. Statement Scan Control

NOCARRET

This operator causes following carriage return characters to be treated as space or tab characters.

CARRET

This operator causes following carriage return characters to act as "statement terminator" characters.

C. Object Program Controls

TITLE NAME

This operator causes the alphanumeric symbol NAME (limited to no more than five characters) to be designated as the name of the object program file.

LOC EXPR

This operator sets the current location counter to the value of EXPR and can be used in both machine coding output and action operator definitions. At the start of translation, the location counter is initially set to zero and the output is relocatable and relative to location zero.

AVAILABLE

This operator sets the current storage pointer to the first available storage location in memory. It may be used only during action operator definition, to facilitate the assignment of storage for coding added to the assembler. This operator is not available in version 1 ADEPT.

RELOCATE

This operator places the translator in relocatable assembly mode. In this mode symbolic address tags are designated relative to the origin or base address of the program and machine words created from relocatable expressions (expressions containing one relocatable term) are subsequently relocated at load time. Address tags defined under relocatable assembly mode are still relocatable if referenced under absolute assembly mode. The translator is initially in relocatable assembly mode at the start of each pass. Each assembly mode has its own location counter, and when a switch from one mode to the other occurs, the assembly location counter is set accordingly. The generation of absolute coding does not affect the relocatable location counter.

ABSOLUTE

This operator places the translator in the absolute assembly mode. Symbolic address tags defined under the absolute mode are given the absolute value of the location counter at the time they were defined, and these tags are not relocated at load time whether referenced from a relocatable or an absolute section of program coding. As in RELOCATE, the absolute assembly mode has its own assembly location counter similarly not affected by the generation of relocatable coding.

TERMINATE

This operator is normally used at the end of an ADEPT input text to bring the translation to an end.

D. External Symbol References and Definitions

ENTRY TAG1, TAG2, TAG3, TAG4,... ;

This statement causes the symbols TAG1, TAG2, etc., (each limited to no more than five characters), to be specified as entry points in the object program. That is, they may be referenced by other programs, and by the AMOS Monitor. Entry points are either absolute or relocatable depending on the current assembly mode.

$NAME

The dollar sign character preceding a symbolic NAME of no more than five characters causes that NAME to be taken as an external reference (reference to an entry point of another program or subprogram). The reference will be linked to the appropriate core memory location at load-time. The use of address arithmetic in a machine word expression containing an external symbolic reference is forbidden.

E. Symbol Definition

The symbolic address tags of ADEPT itself are initially available in the symbol table. When all action operator definitions have been made, the operator

EXPUNGE

may be used to remove ADEPT's own symbols to make room in memory for additional symbol storage. This operator should always appear before any Macro definitions or machine code generation whether or not there have been any action operator definitions. Symbol values may be defined during translation by one of the following:

SYM=EXPR

This statement causes the symbol SYM to be given the value of the expression EXPR. Symbols defined in this manner may be redefined at will. If the value of the expression is undefined on the first pass of the translator (i.e., the expression contains at least one undefined symbol), the symbol remains undefined. If the expression is undefined on the second pass of the translator, it will result in an error message. In subsequent uses, the symbol SYM will have the same relocation properties as the expression EXPR. If the symbol being defined is the symbol "VERSION", ADEPT will set the version number in the output object relocatable program to the value of the expression. This definition, if desired, should be used at the beginning of the program and should be scanned during

each pass of translation since ADEPT normally sets the version of the output file from the version number of the input source text. The revision level of the output file is set to that of the input source program file.

TAG:

The appearance of a symbol followed by a colon causes the symbol (e.g., TAG) to be given the value of the "current location" counter. Symbols defined in this manner may not be redefined. On the second pass thru the input, the value of such a symbol is compared with the "current location", and an error message results if they are not equal. When defined outside the scope of the ABSOLUTE operator, TAG symbols are relocatable.

COMMON SYM(LEN1),SYM2(LEN2),SYM3(LEN3),... ;

This statement causes the symbols SYM1, SYM2, SYM3, etc., to be specified as "common storage" references. The value of the expression LENn gives the size of the element in common storage. If "(LENn)" is omitted, the size of SYMn is assumed to be one word.

F. Macro Definition

The Macro definition consists of a heading statement, a body, and a terminator, in the format:

```
MACRO          NAME (ARG1, ARG2, ...);   (heading)
STRING                                   (body)
ENDM                                     (terminator)
```

If the character string comprising the body of the Macro contains the substrings ARG1, ARG2, etc., these "Dummy Arguments" are replaced by corresponding "place markers" during Macro definition. The body, containing these place markers, is stored during definition, and identified by the NAME. If the MACRO heading line contains no dummy arguments, no "place markers" will be defined in the Macro body and no argument list is required in the Macro call.

Macro definition may occur during pass 1 of the translator, during pass 2, or during both pass 1 and pass 2, according to the following operators:

MACRO

This operator causes a Macro definition during pass 1, and a redefinition during pass 2.

MACRO1

This oerator causes a Macro definition during pass 1 only, and should be used for normal Macros to avoid redefinition during the pass 2 evaluation scan.

MACRO2

This operator causes a Macro definition during pass 2 only.

G. Nested Macro Definition

A Macro definition may be included within the body of another Macro definition. The nested (inner) Macro is defined (or redefined) during the expansion of the encompassing Macro body. This takes place each time the encompassing Macro is called, and the resulting definition of the nested Macro remains valid until it is redefined, or the expansion of the encompassing Macro is completed.

H. Action Operator Definitions

Definition of new action operators is accomplished in response to the input sequence:

DEFINE EXPRESSION1 NAME ;

STRING

ENDD EXPRESSION2

where the value of EXPRESSION1 (which must be terminated by ",") is the "type" of action operator. NAME is the symbolic name of the action operator, and the STRING defines the coding to be added to the translator, with entry at EXPRESSION2.

An alternate form is:

DEFINE:

STRING

ENDD

which adds coding to the translator without defining a new operator.

If the type is "Ø" and no NAME is given, the coding generated by the STRING is added to ADEPT without defining an operator. In this case no EXPRESSION need be given.

The "type" of the action operator is an octal value from 1 to 177_8, specifying a number used for identification during scanning and evaluation. The ADEPT Maintenance Manual should be consulted for a list of types for existing operators and instructions for creating new ones.

All action operator definitions must occur before any other part of the input text which generates machine language output. The operator

ENDAO

must be used when all action operator definitions have been made. This statement sets the current output location to the value it had prior to processing definitions.

The Action Operator define facility and associated operators are available only in ADEPT versions 2, 4, and 5.

PROGRAM FORMATS

The following restrictions should be observed in preparing a source language program for input to ADEPT:

The operator

TITLE NAME

should be placed near the beginning of the program, before any machine coding.

Action operator definitions should come at the beginning of the program, before any Macro definitions and machine coding. These action operator definitions should be followed by the operators

ENDAO

EXPUNGE

Next follow Macro definitions and machine coding, in which all ENTRY statements must appear before any machine code-generating or location-setting statements and operators.

The COMMON statement may be placed anywhere in the program after EXPUNGE and before TERMINATE.

The operator

TERMINATE

should be placed at the physical end of the program. If no action operator definitions are being made, the "ENDAO" operator should be omitted. However, the "EXPUNGE" statement must be present except when terminating with the "SETTRAN" statement.

The statement

SETTRAN STRING "carriage return"

may be used instead of TERMINATE, in which case the current ADEPT translator in core memory containing the Action Operator definitions, Macro definitions and parameter assignments from the source input is written on the current selected system tape. When another translation is started, using this "extended" ADEPT, the STRING from the SETTRAN statement will be typed out to identify the version of ADEPT beind used. The SETTRAN operator is not available in version 1 ADEPT. Note: A program terminated by a SETTRAN statement does not generate any output code.

The operator ADEPTSYMS should not be used in the input source language program as this operator can only be used in the generation of ADEPT assemblies of ADEPT.

ERROR DIAGNOSTICS

The three forms of the error message are as follows:

Form I - Terminating error (termination of both scan and output)

```
TERnn A:SYM1  B:SYM2  C:SYM3  D:OLOC  E:SYMLOC
```

Form II - Output terminating error (scan continues)

```
*ERRnn  A:SYM1  B:SYM2  C:SYM3  D:OLOC  E:SYMLOC
```

Form III - Error indication (scan and output continue)

```
ERRnn  A:SYM1  B:SYM2  C:SYM3  D:OLOC  E:SYMLOC
```

In the above forms "nn" is the decimal error number, SYM1, SYM2, and SYM3 are three symbol buffers in the translator and depending on the particular error messages generated, will give meaning to the error, OLOC is the octal location counter value at the time of the error, and SYMLOC, if present, is the symbolic location of the error. In both Forms I and II, the output is terminated. In Form III the scan is terminated and control is returned from ADEPT.

ADDITIONAL DETAILS

A. Characters:

1. AMOS Character Set

Character Strings output by ADEPT (both message and object text) are in AMOS internal 6-bit character codes (single case), as follows:

Code	OPC Character	TTY Character	LNPR Character
ØØ	^ (null)	[	^ (null)
Ø1	%	%	%
Ø2	¢	]	¢
Ø3	!	!	!
Ø4	&	&	&
Ø5	*	*	*
Ø6	:	:	:
Ø7	_	\	_
1Ø	+	+	+
11	tab	tab (3 spaces)	<
12	?	?	?
13	"	"	"
14	'	'	'
15	carriage return	return - L. F.	>
16	(	(	(
17	)	)	)
2Ø	Ø	Ø	Ø
21	1	1	1
22	2	2	2
23	3	3	3
24	4	4	4
25	5	5	5
26	6	6	6
27	7	7	7
3Ø	8	8	8
31	9	9	9
32	;	;	;
33	=	=	=
34	.	.	,
35	-	-	-
36	.	.	.

Code	OPC Character	TTY Character	LNPR Character
37	/	/	/
4Ø	space	space	blank
41	A	A	A
42	B	B	B
43	C	C	C
44	D	D	D
45	E	E	E
46	F	F	F
47	G	G	G
5Ø	H	H	H
51	I	I	I
52	J	J	J
53	K	K	K
54	L	L	L
55	M	M	M
56	N	N	N
57	O	O	O
6Ø	P	P	P
61	Q	Q	Q
62	R	R	R
63	S	S	S
64	T	T	T
65	U	U	U
66	V	V	V
67	W	W	W
7Ø	X	X	X
71	Y	Y	Y
72	Z	Z	Z
73	$	$	$
74	#	#	#
75	@	@	@
76	±	↑	±
77	backspace	←	|

2. ASCII Character Set

Code	ASCII	LCG1	Code	ASCII	LCG1
ØØØ	NUL	NUL	Ø4Ø	space	space
ØØ1	SOH		Ø41	!	!
ØØ2	STX		Ø42	"	"
ØØ3	ETX		Ø43	#	#
ØØ4	EOT		Ø44	$	$
ØØ5	ENQ		Ø45	%	%
ØØ6	ACK		Ø46	&	&
ØØ7	BEL		Ø47	'	'
Ø1Ø	BS	italics control	Ø5Ø	(	(
Ø11	HT	position "X"	Ø51	)	)
Ø12	LF	line feed	Ø52	*	*
Ø13	VT	position "Y"	Ø53	+	+
Ø14	FF		Ø54	,	,
Ø15	CR	carriage return	Ø55	-	-
Ø16	SO		Ø56	.	.
Ø17	SI		Ø57	/	/
Ø2Ø	DLE		Ø6Ø	0	0
Ø21	DC1		Ø61	1	1
Ø22	DC2	shrink size	Ø62	2	2
Ø23	DC3	expand size	Ø63	3	3
Ø24	DC4		Ø64	4	4
Ø25	NAK		Ø65	5	5
Ø26	SYN		Ø66	6	6
Ø27	ETB		Ø67	7	7
Ø3Ø	CAN		Ø7Ø	8	8
Ø31	EM		Ø71	9	9
Ø32	SS		Ø72	:	:
Ø33	ESC		Ø73	;	;
Ø34	FS	lower line	Ø74	<	<
Ø35	GS	raise line	Ø75	=	=
Ø36	RS	dim intensity	Ø76	>	>
Ø37	US	brighten intensity	Ø77	?	?

Code	ASCII	LCG1	Code	ASCII	LCG1*
100	@	@	140	\	\
101	A	A	141	a	a
102	B	B	142	b	b
103	C	C	143	c	c
104	D	D	144	d	d
105	E	E	145	e	e
106	F	F	146	f	f
107	G	G	147	g	g
110	H	H	150	h	h
111	I	I	151	i	i
112	J	J	152	j	j
113	K	K	153	k	k
114	L	L	154	l	l
115	M	M	155	m	m
116	N	N	156	n	n
117	O	O	157	o	o
120	P	P	160	p	p
121	Q	Q	161	q	q
122	R	R	162	r	r
123	S	S	163	s	s
124	T	T	164	t	t
125	U	U	165	u	u
126	V	V	166	v	v
127	W	W	167	w	w
130	X	X	170	x	x
131	Y	Y	171	y	y
132	Z	Z	172	z	z
133	[	[	173	{	{
134	\	\	174	|	|
135	]	]	175	}	}
136	^	^	176	~	~
137	_	_	177	DEL	DEL

* NOTE: These LCG1 characters (140_8-177_8) are with the standard character set expansion option.

B. Symbols

ADEPT accepts a stream of characters until a "symbol" is recognized in the characters gathered. For this function characters are classified into seven types as follows:

Type	Characters
1	, $ " % ( )
2	A throuth Z, Ø through 9
3	; Carret (CARRET mode)
4	= :
5	Tab Space Carret (NOCARRET mode)
6	@] ? # + - / * ↑ & '
7	!

Characters of "odd" types form one-character symbols, otherwise all successive characters of the same type form single symbols.

After each symbol is obtained it is looked up in the translation tables to determine if it has a value or if it is an operator which names a routine to be performed.

C. Values

The values which ADEPT symbols may assume or generate as operators are of the following three types:

1. Strings

Values which are strings consist of a sequence of ADEPT characters of arbitrary length.

2. Words

Values which are words consist of a 3Ø-bit binary values.

3. Addresses

Values which are addresses consist of 15-bit binary fields which may be modified to designate addressable cells in core memory. Address values may be modified for the following four forms of designation.

Relative -

Relative address values refer to location local to the program being assembled.

Absolute -

Absolute address values refer to fixed locations in memory independent of where the current program will be loaded for execution.

Common -

Common address values refer to locations local to a block of storage common to all programs.

External -

External reference address values refer to global parameters and entry points to be defined by other programs when the current one is loaded for execution.

D. Operators

Operator symbols invoke the execution of translator routines which perform any or several of the following functions: generation of values, generation of output, entering definitions and controlling the translation process.

In the process of performing its function an operator may make use of a preceding value or of several following values, any of which may have been generated by other operators.

If the value left by an operator is of type string, it is then broken into symbols and scanned for operators to execute.

When the function of the operator is to generate output, either of two output facilities is used. One outputs messages at assembly time to the assigned control unit, and the other generates object code for the program being assembled with any necessary control codes for its proper loading and binding for execution.

The operators which make definitions or control the translation process do so by making entries in the translator's tables or setting flags which affect the execution of subsequent translating procedures.

E. Statements

The gathering of an input character string into symbols and scanning them for operators to perform (which may give further values upon which other operators perform) and repeating the entire procedure on any text strings generated in the process, is termed evaluation of the character string.

The unit of text string upon which the translator performs this evaluation is called a statement.

APPENDIX A

CONSOLE ERROR MESSAGES

See section on ERROR DIAGNOSTICS for message formats and codes.

*ERR1	Comma scanned on machine word expression evaluation. Could be misspelled Macro name or operator.
*ERR2	Address arithmetic in expression with external reference. Ex. - $SPTR+5
ERR3	Illegal relocation in machine word output. Illegal relocation occurs when two relocatable terms are added together. A relocatable term is subtracted from some expression or a relocatable expression is rotated.
*ERR4	Undefined symbol in common size declaration. (B:SYM is error symbol).
ERR5	Undefined symbol in conditional statement expression. (See last ERR2Ø message for undefined symbol).
ERR6	Character following ! is not B, K, or H.
ERR7	Comma scanned in parameter definition. Commas are illegal terminators for expressions giving parameter values.
ERR8	Symbol in parameter definition (before =) is previously defined not a parameter. A redefinition of a previously defined address tag or an initially defined instruction (predefined in ADEPT) could cause this.
ERR9	No symbol scanned for address tag definition. This is caused by the scanning of a colon (":") without an alphanumeric symbol preceding it (however, they can be separated by tabs or spaces).
ERR1Ø	A redefinition of an address tag was specified. (C:SYM is address tag). Multiply defined address tag (occurs on pass 1).
*ERR11	Tag value does not equal current location value (pass 2). (D:LOC is current location value). In pass 2, the current location when an address tag is scanned differs from the defined value in pass 1. This can usually be causes by some conditional assembly statements or Macro calls which generate an unequal number of instructions on pass 1 and pass 2.
ERR12	Symbol for address tag definition is previously defined not a tag. (C:SYM is symbol in error). This might be caused by trying to define an address tag which has already been defined as a parameter, instruction, etc.
ERR13	Symbol specified in entry point declaration undefined on pass 2. (B:SYM is symbol in error). The symbol was never defined as an address tag in pass 1.
*ERR14	Comma scanned in expression evaluation for AO definition machine word. Commas are illegal expression terminators in machine words for action operator definitions.

*ERR15	Undefined symbol in expression evaluation for AO definition machine word. (C:SYM is symbol in error)
ERR16	Period scanned not separated from previous symbol. (C:SYM is previous symbol). No operator between term and following period (not including tabs and spaces).
ERR17	Character "8" or "9" appears in octal input. (C:SYM is numeric string in error)
ERR18	Number overflow error (number greater than 2^{29-1}). (C:SYM is undefined symbol). Decimal string greater than 2^{29-1}.
ERR19	Common symbol predefined.
ERR2Ø	Undefined symbol in expression evaluation. (C:SYM is undefined symbol)
ERR21	Common symbol redefined.
ERR22	Undefined symbol in repeat count expression evaluation. (See last ERR2Ø message for symbol)
ERR23	Symbol consisting of greater than 1Ø characters scanned by the string expansion scanner. This is caused by the inputting from source text of a symbol consisting of more than ten characters (not including comments).
ERR24	Symbol consisting of greater than 1Ø characters scanned by the expression evaluator scanner. This is caused by the concatenation of two strings causing a symbol of more than ten characters to be formed.
*ERR25	Comma scanned in dispatch address expression following "ENDD". Commas are illegal expression terminators for the expression used to specify the dispatch address in an action operator definition.
ERR26	Undefined symbol in dispatch address expression following "ENDD". (See last ERR2Ø message for undefined symbol)
ERR28	Entry point same symbol as external reference. (B:SYM is symbol) Entry point cannot be the same as an external reference used in the same program.
*ERR29	Comma return in LOC statement. Commas cannot be used as the expression terminator for the expression computed in a "LOC" statement.
ERR30	Undefined symbol in LOC statement. (See last ERR2Ø message for undefined symbol)
TER1	Too many dummy arguments in a Macro definition (more than 62_{10}).
TER2	End of Macro scanned and Macro is not last entry in push-down list. Adept scanned an end of Macro indicator. (Should not occur)

NOTE: The errors listed as "should not occur" should not occur in normal operation of ADEPT. However, they are listed as they might occur as a result of incorrect action operator definition operations.

TER3	End of argument scanned and Macro or indefinite repeat argument is not last entry on list. Adept scanned an end of argument indicator. (Should not occur)
TER6	Macro dummy argument pointer scanned and no Macros on push-down list. (Should not occur)
TER7	"STOP" scanned and no Macros on push-down list. A "STOP" is placed erroneously in the program not within an encompassing MACRO, REPEAT, or IREPEAT.
TER8	"ENDR" scanned and "REPEAT" not last entry on push-down list. (Should not occur)
TER9	Illegal format of IREPEAT argument list. This message can occur if there is not an alphanumeric symbol followed by a comma after the space(s) or tab(s) in an IREPEAT statement.
TER1Ø	"DEFINE" scanned while in AO define mode. This will result from a DEFINE operator placed within an action operator definition.
TER11	Location not specified for AO definition. This occurs when a machine word has been evaluated to insert into memory for an action operator definition and the program has not told ADEPT where to put it by a LOC statement or the AVAILABLE operator.
TER12	"ENDI" scanned and "IREPEAT" not last entry on push-down list. (Should not occur)
TER13	Argument list storage table exceeded. This occurs when there are more than 400_8 entries in the argument list table (200_8 in version 1).
TER14	Dummy argument pointer scanned in argument list and no Macros on push-down list. (Should not occur)
TER15	Dummy argument pointer scanned in argument storage. (Should not occur)
TER16	"ENDD" scanned and not in AO definition mode. The symbol "ENDD" is placed erroneously in the program.
TER17	Too many arguments on argument list (more than 62_{10}). More than 62_{10} arguments appear in the call of a Macro or IREPEAT.
TER18	String Push-down List exceeded (more than 63_{10} entries). This occurs if the total nesting level of MACROS, REPEATS, IREPEATS, and arguments exceed 63_{10}. One common cause of this is a recursive Macro call which does not terminate.
TER19	Illegal entry type in call to string push-down subroutine. (Should not occur)

NOTE: The errors listed as "should not occur" should not occur in normal operation of ADEPT. However, they are listed as they might occur as a result of incorrect action operator definition operations.

TER2Ø	Illegal entry type in string pop-up. (Should not occur)
TER21	String Push-down List underflow. String pop-up subroutine called and level equals zero. (Should not occur)
TER22	String storage exceeded. This is caused by extensive Macro definitions and/or long repeated strings.
TER23	Symbol table length exceeded. Too many symbols defined in the input program. Adept cannot handle that program without larger memory.
TER24	Expression Evaluation Push-down List overflow. (more than 7 entries). This can occur when there is nesting of expression calls in machine expression evaluation, repeat counts, and conditional statements.
TER25	Subexpression Push-down List overflow. (more than 31_{10} entries). This is caused by too many subexpression levels in an expression or in nested expressions.
TER26	Subexpression Push-down List underflow. This is caused by too many right parentheses in an expression.
TER27	No symbol scanned for parameter definition. There was no alpha-numeric symbol before the "=".
TER28	Equal sign scanned while evaluating another parameter. Parameter definitions cannot be nested.
TER29	Parameter Storage exceeded. Too many parameters in the program.
TER30	Illegal symbol type-in Symbol Table. (Should not occur)
TER31	Illegal arithmetic or logical operator. (Should not occur)
TER32	Expression Evaluation Push-down List underflow. (Should not occur)
TER33	Parentheses nesting error in expression. Not the same number of right and left parentheses in an expression.
TER34	"AVAILABLE" scanned and not in define mode. "AVAILABLE" operator is placed erroneously in the program.
TER35	"AVAILABLE" scanned and already set. Two "AVAILABLE" operators in the same action operator definition not separated by a "LOC" statement.
TER36	"ENDM" scanned while not in Macro define mode. "ENDM" placed erroneously in the program.

NOTE: The errors listed as "should not occur" should not occur in normal operation of ADEPT. However, they are listed as they might occur as a result of incorrect action operator definition operations.

APPENDIX B

BASIC ADEPT ACTION OPERATORS AND INSTRUCTION CODES

The following action operator symbols and predefined instruction codes are part of the basic ADEPT Translator and may not be used for symbolic tags and value definitions.

String Action Operators

MACRO	ENDI	"	RELOCATE
MACRO1	IFZERO	#	ENDAO
MACRO2	IFNEGATIVE	##	EXPUNGE
STOP	IFPASS1	%	SETTRAN
ENDM	IFSAME	DEFINE	TERMINATE
REPEAT	ELSE	AVAILABLE	ADEPTSYMS
ENDR	ENDC	ENDD	
IREPEAT	\	ABSOLUTE	

Expression Action Operators

(TAB)	&	,	COMMON
(SPACE)	:	LOC	TITLE
(C/R)	=	OCTAL	TYPEOUT
;	$	DECIMAL	TYPEOCT
+	!	TEXT	TYPEDEC
-	.	CARRET	STRING
'	(	NOCARRET	ASCII (Version 5 only)
*	)	ENTRY	*
			/ } (Versions 4 and 5 only)
			// }

Pre-defined Instruction Codes

MDMD	ARMD	BRMD	ICMD
MDAR	ARAR	BRAR	ICAR
MDBR	ARBR	BRBR	ICBR
MDIC	ARIC	BRIC	ICIC
MDIR	ARIR	BRIR	ICIR
MDAS	ARAS	BRAS	ICAS
MDAE	ARAE	BRAE	ICAE
MDXO	ARXO	BRXO	ICXO
S4MD	S7AR	JSAN	ANAS
S4AR	S7BR	JPAN	ANAE
S4BR	S7IC	JSLS	ANXO
S4IC	S7IR	JPLS	LSMD
S4IR	B	SKAN	LSAR
S5MD	K	SKLS	LSBR
S5AR	H	SKUA	LSIC
S5BR	O	SKLA	LSIR
S5IC	A	ANMD	LSAS
S5IR	N	ANAR	LSAE
S6MD	X	ANBR	LSXO
S6AR	I	ANIC	CNVT
S6BR	L	ANIR	OPTY
S3IC	F		OPTI
S6IR	JPSR		OPCR
S7MD	JUMP		FPRI
			UPRI
			NOOP

Additional Instruction Codes (Versions 4 and 5 Only)

MPYL	DIVL	ARRS
MPYU	DIVU	ARLS
MPYI	DIVI	PINT
NORM	ERAR	

Display Image Codes (Version 5 Only)

MOVE	LDSCL	JMP	FLAG2
2DTBL	LDRX	JSR	PEN
DRAW	LDRY	LDMB	WBX1
2DTF	LDRZ	ORMB	WBY1
LABLM	LDRV	ANDMB	WBZ1
LABL	LDX	LDSN	WBX2
NUL	LDY	LDLS	WBY2
SCL	LDZ	CJMP	WBZ2
ROTX	LDV	CJSR	IFW
ROTY	LDI	WJMP	IRL
ROTZ	SAVT	WJSR	RHW
RXYZ	REST	LDW	RFW
DX	RET	MVW	RRL
DY	IMG	LWS	DIR
DZ	LOOP	DEPTH	STR
DV	ENDL	FLAG1	

adage

AFDSP

AFORT DISPLAY INTERFACE

Programmer's Reference Manual

Revision C

July 1969

CONTENTS

CONTENTS (Cont.)

INTRODUCTION

AFDSP provides a FORTRAN programmer with the necessary facilities to display two-dimensional pictures of three-dimensional images on the Graphics Display Scope. A selected image is displayed by means of a SHOW call, with an argument specifying the desired image. Images are kept in buffers, each buffer being a single-dimensioned FORTRAN array in which the description of an image has been created. The programmer may build up a description of an image by means of CALL statements, which add image defining items to its buffer.

For each separate image, the programmer must perform the following tasks:

1. Dimension a variable (integer or real) to be used as a buffer in which the image description is to be generated.
2. Empty and initialize the buffer; this is accomplished by means of a RSET call with arguments specifying which buffer and its dimension size.
3. Image-describing items can be added to the buffer. Two options are provided with respect to these items. Either they will remain unchanged throughout the display of the image, or they are dynamic and may be changed by subsequent computation or operator interaction.

Items may require arguments. For unchanging items requiring numerical values as arguments, these may be any fixed or floating point FORTRAN expression. For dynamic arguments requiring numerical values, they may be the name of any local or global (fixed or floating point) FORTRAN variable.

The call IMCON is provided for adding unchangable image items to a buffer, and IMVAR is used for adding dynamic image describing items.

DEFINING IMAGES

All images, when displayed, result in viewable lines or text strings suitably generated and placed in the viewing space.

The viewing space is a cube in which the definition of the shown image is specified. The orientation of the viewing space is that of a right-hand coordinate system with the y-axis vertical, the x-axis horizontal from left to right, and the z-axis horizontal coming out of the display screen towards the viewer.

The items defining an image specify:

1. Visible elements such as lines, text strings, and other images.
2. Placement of visible elements -- these may include changes in position, size, or orientation.
3. Subprograms to be executed for any required dynamic variation, geometrical constraints, or operator interaction.

Lines and text items have arguments giving their coordinates within the current definition space. For a shown image with no "Placement" items, the definition-space for all items corresponds to the viewing-space. The position in the definition space in which the next visible element will be generated is referred to as the "beam position." The generation of visual elements is done by a moving "beam" of light in the definition space coordinate system.

An image definition space is also a cube, its sides ranging from minimum to maximum values of the valid argument number ranges:

1. $-10,000 \leqq$ fixed point arguments $\leqq +10,000$
2. $-1. \leqq$ floating point arguments $\leqq +1.$

The arguments to PLACE items may specify a change in scale, rotation, and/or displacement of the image portion now described by subsequent items. The arguments for rotation range from $-\pi$ radians for the minimum numerical value, increase linearly through 0 radians for the zero value, and extend to $+\pi$ for the maximum numerical value.

All placement items affect the definition space. Items following a PLACE item have their coordinates measured with respect to a new coordinate system which has been turned, moved, or shrunk as specified by the PLACE item. The effect of a previous PLACE item can be stopped by a SPLC (stop place) item. All PLACE items are otherwise cumulative, i.e., each successive PLACE is performed with respect to the coordinate system resulting from the previous PLACE, not that of the image-start or viewer-space. In this way, PLACE-SPLC ranges may nest image describing segments.

Multiple instances of defined images may be placed about a definition space as valid elements in the definition of a new image. The IMCAL item is provided to implement such sub-image calls in a image's definition. The argument to an IMCAL item is the name of the buffer containing the sub-image definition.

Lines and text strings can be created, arguments specifying them altered, or arguments to items affecting their placement altered, by executing subprograms.

The CALL item in an image definition causes a subroutine to be executed.

This is a necessary element in the definition of dynamic images which are to be affected by computation or I/O. The arguments to a CALL item are the name of the subroutine to be executed, followed by its arguments (if any).

When numerical values for arguments to image describing items are to be varied during display, the item should be added to the image buffer via an IMVAR call. The variables used for arguments in the item may then have their values changed dynamically by CALL'ed subprograms during display. But, these dynamically assigned values must be transformed to machine oriented values prior to assignment. This is done automatically by AFDSP in all other cases.

The functions IMV and RMV are provided to convert their one argument, any valid fixed or floating point FORTRAN expression over the standard image argument ranges, into the proper hardware representation for assignment. IMV is used to assign values to integer variables, and RMV is used to assign values to real variables.

e.g.,: A = RMV (.3*B-C)

I = IMV (.25)

Similarly, the functions ISVA and RSVA are used to assign the address of an array name to a variable (integer or real, respectively).

e.g.,: A = RSVA (B), where B is a dimensioned array.

I = ISVA (IB), where IB is a dimensioned array.

All desired dynamic changes of displayed images cannot be effected through variations of numerical values of arguments to items. Some desired changes constitute removal or inclusion of image portions. Conditional and unconditional "jump-in-image" type items are provided for this purpose. The tested conditions provided for are: sign of any dynamic variable, and detection of operator light pen selection over any specified image portion.

To provide local references for image jumps, any item of an image buffer may be "labeled" by a variable.

Each call adding an item to an image's buffer provides a location value which may be used as a label referring to the next item. Jump items using this label may be added to the buffer before or after this point of assignment.

DESCRIPTION OF AFDSP CALLS

1. A selected dimensioned variable is cleared and initialized for use as a buffer into which an image description may be generated by the following call:

CALL RSET (Pic, n)

where: Pic = name of integer or real dimensioned variable.

n = size of dimensioned variable Pic.

2. The following CALL will cause the addition to the buffer Pic of an image item, Item, with argument references as current values of Arg1 and Arg2,...

CALL IMCON (Pic, Item (Arg1, Arg2,...), Loc)

where: Pic = name of dimensioned variable to be used as the buffer for the image item.

Item = operation to be added to image in Pic. A description of the available items is included in the next section.

Argn = Item arguments are local or global variables, literal constants, or valid AFORT expressions.

Loc = Variable (error flag) which is set to negative if Item could not be entered because there were no more elements of Pic available or, if positive, a label for the next image item in Pic.

3. The following CALL will cause the addition to the buffer Pic of an image item with argument references as running values of Arg1, Arg2,... .

CALL IMVAR (Pic, Item (Arg1, Arg2,...), Loc)

where: Pic = name of dimensioned variable, used as an image definition buffer which will contain the image Item.

Item = operation to be added to image in Pic. A description of the available items is included in the next section.

Argn = Item arguments can only be local or global variables for numeric arguments.

Loc = variable (error flag) which is set to negative if Item could not be entered because there were no more elements of Pic available or, if positive, a label for the next image item in Pic.

4. The following CALL will cause the image items described in buffer Pic to be displayed:

CALL SHOW (Pic, n)

where: Pic = name of dimensioned variable, the elements of which contain image items previously generated.

n = fixed point argument giving frame rate code as follows:

n	Rate (Frame/Second)
1	60
2	40
3	30
4	24
5	20
7	15
9	12
11	10

5. The following CALL will suspend any current image display:

CALL NOSHO

6. The following CALL will establish a dimensioned variable as a text buffer into which Hollerith or numeric variables may be output as characters of a text record:

CALL SETIO (Buff, n)

where: Buff = name of dimensioned variable into which characters are to be written.

n = the dimension size of Buff.

Once established, a record may be output to Buff under format control by an AFORT WRITE statement using unit 54. Once containing an output record, Buff may be used as an argument to the text generating image item LABEL (described below).

7. The following CALLS will allow a more efficient display of a sequence of two-dimension tables (i.e., X, Y coordinate pairs associated with a single Z-value). These tables can only hold constant, pre-computed coordinate lists.

CALL TABL (Pic, Buff, Loc)

or

CALL TABLF (Pic, Buff, Loc)

where: Pic = name of dimensioned variable in which the image item is to be added.

Buff = name of dimensioned variable containing the sequence of coordinate pairs.

Loc = Variable (error flag) which is set to negative if the item could not be entered because there were no more elements of Pic available , or if positive, a label for the next image item in Pic.

Note

TABLF is only used when all lines are short (less than 0.5 in. in length)

Associated with the above TABL and TABLF CALLS is a function ZSET, required to initialized the buffer into which 2-D lists may be generated and is used as follows:

Buff (1) = ZSET (z, name)

or

Buff (1) = ZSET (z)

where: Buff (1) = the first element of the array to be filled. Subsequent X, Y values start with the element Buff (2).

Name = when present, the array name of a possible subsequent 2-D list, to be displayed along with the list in Buff.

An X, Y coordinate pair is assigned to particular elements of an array by using the function TDFUN as follows:

Buff (n) = TDFUN (x, y, n1, n2)

where: x = the X coordinate value expression.

y = the Y coordinate value expression.

n1 = 0 if no line is to be drawn at this point, 1 if a line is to be drawn at this point.

n2 = 0 if this is not the last point in the list, 1 if this is the last point in the list.

DESCRIPTION OF IMAGE DEFINING ITEMS

The following items (and appropriate arguments) in IMCON and IMVAR statements will cause the described image definitions to be added to the specified buffer.

1. The item to reposition the beam is:

$$\text{MOVBM}\left(\left\{\begin{matrix}\text{v}\\ \text{PT1 (x, y, z)}\end{matrix}\right\}\right)$$

2. The item for drawing lines is:

$$\text{LINE}\left(\left\{\begin{matrix}\text{v1}\\ \text{PT1 (x1, y1, z1)}\end{matrix}\right\}\ ,\ \left\{\begin{matrix}\text{v2}\\ \text{PT2 (x2, y2, z2)}\end{matrix}\right\}\right)$$

The arguments are either names of dimensioned variables (v1 and v2), the elements of which contain, respectively, the X, Y, and Z coordinates of the end points, or are functions (PT1 or PT2) of the corresponding coordinates.

The line will be drawn from the first point to the second point leaving the beam positioned at the second point.

If the first point argument and the separating comma are omitted, the line will be drawn from the second point of the previous LINE item (or the current beam position if it has been changed).

Example:

```
CALL IMCON (PIC, LINE (V1, PT2 (-.5, .06, A*3.)), ER1)
```

where V1 is dimensioned and the arguments to PT2 lie within the allowable range of numbers.

3. The operation to generate image items which can be used to display character strings is:

LABEL (Buff, Mode)

The characters output into Buff by means of a WRITE statement will be displayed, according to the specified mode, in a plane parallel to the viewing screen. The LABEL will begin at the current beam position, which may be set by a MOVBM operation.

Mode = 1 for small horizontal line.
= 2 for medium horizontal line.
= 3 for large horizontal line.
= 4 for small column*
= 5 for medium column*
= 6 for large column. *

After the text has been displayed, the beam is returned to the start of the text. Thus. LABEL items do not affect the beam position.

4. The item used to specify scale, position, and any rotation of subsequent items is:

$$\text{PLACE}\left(\text{SHRINK (s)},\left\{\begin{array}{ll}\text{MOV} & \text{(dx, dy, dz)}\\ \text{XMOV} & \text{(dx)}\\ \text{YMOV} & \text{(dy)}\\ \text{ZMOV} & \text{(dz)}\end{array}\right\},\left\{\begin{array}{ll}\text{XTURN} & \text{(rx)}\\ \text{YTURN} & \text{(ry)}\\ \text{ZTURN} & \text{(rz)}\end{array}\right\}\right)$$

where s is the desired size reduction; dx, dy, and dz the amount of X, Y, and Z displacement respectively; and rx, ry, and rz are the amount of rotation about the X, Y, and Z axes respectively. The arguments to PLACE may be in order, or omitted. This imposed PLACEing will affect subsequent items until the next PLACE or "STop PLaCe" item (see 5).

5. The effect of the previous PLACE item which established the current scale, position, and rotation may be cancelled by an item, STPLC, entered as follows:

CALL STPLC (Pic, Loc)

Any positioning, rotation, or scaling which affected items prior to the last PLACE (being cancelled) will resume and affect items subsequent to the STPLC.

*Available only on system with LCG Character Generator Option.

The item is added to image Pic. Loc defines the location of the next item that can be added to Pic.

NOTE

This item, having no arguments, need not (and cannot) be entered via an IMCON or IMVAR call.

Thus, PLACE and STPLC items may nest segments of image descriptions. The outermost segment is "PLACEd" with origin at the center of the screen, X axis to the right, Y axis vertical, and Z axis coming out towards the viewer.

6. The item used to include to entire image described in another dimensioned array for processing under the currently established PLACEing is:

IMCAL (Name)

where Name is the name of the other dimensioned array containing image items.

When the sub-image selection is to be dynamically re-assigned during display, (an IMVAR was used to add it to image buffer), any variable may be used for Name.

Name may then be assigned during display by an ISVA or RSVA call.

7. The item to enable calling for the execution of another external subroutine while displaying an image is:

CALL (Name, Arg1, Arg2,...)

where Name is the name of the external subprogram to be called.

NOTE

Name must have been previously declared to be external by means of a specification statement of the following form: EXTERNAL Name

When the CALL is to be dynamically alterable during display (IMVAR), the arguments must be variables. The first argument, the subroutine to be called, must still be the name of an EXTERNAL subprogram.

8. The item to cause the processing of the current image to change to another location is:

GO (Loc)

where Loc is a variable referencing an item in the image. Loc is defined by the IMVAR or IMCON call preceding the item referenced.

9. Processing of the current image may be changed to another location under two tested conditions:

IFNEG (Var, loc)

IFZRO (Var, loc)

In the above, if the value of Var is negative or zero, respectively, processing is transferred to loc, a variable referencing an item in the image. Loc is defined by the IMVAR or IMCON call preceding the item referenced.

10. A further tested condition which may cause processing of the current image to be changed to another location is detection of operator light pen selection over any specified image portion, as follows:

IFPEN (Loc)

If the light pen was detected, processing of the current image will be changed to Loc, a variable referencing an item in the image. Loc is defined by the IMVAR or IMCON call preceding the item referenced.

Light pen detection is enabled by means of the following operation:

CALL PENON (Pic, Loc)

The item is added to image Pic. Loc defines the location of the next item that may be added to Pic.

NOTE

This item, having no arguments, need not (and cannot) be entered via an IMCON or IMVAR call.

Light pen detection is disenabled by means of the following call:

PENOF (Pic, Loc)

The operation to cause repeated processing of a sequence of items is:

LOP (n)

where: n = the number of times the sequence is to be repeated.

The item which specifies the point at which the preceding looping is to end is entered as follows:

CALL STPLP (Pic, Loc)

The operation to enable lines to be drawn in a dash mode is:

CALL DSH (Pic, Loc)

To discontinue the dash mode, the following call is used:

CALL NODSH (Pic, Loc)

The above three items are added to image Pic. Loc defines the next item to be added to Pic.

NOTE

The above three items, having no arguments, need not (and cannot) be entered via an IMCON or IMVAR call.

AFDSP VALUES AND NUMBER RANGES

An additional function is available and must be used to assign values to variables which are used as arguments to Items entered by IMVAR calls:

IMV (valid AFORT expression)

Example:

A = IMV (.3*B-C)

All other arguments may be specified by any valid floating or fixed point FORTRAN expression.

The range of allowable values is as follows:

fixed point: [-10,000, +10,000]

floating point: [-1, +1]

These correspond to [minimum, maximum] values of all arguments giving coordinates, scales, or angles. Angles correspond to $[-\pi, +\pi]$ radians.

IMAGE BUFFER SIZE REQUIREMENTS

The dimension size required by an image buffer may be determined by totaling the number of words per item or call, as follows:

Name of Call or Item	Number of Buffer Words Needed* IMCON Class	IMVAR Class	Call
RSET	-	-	2
TABL, TABLF	-	-	1
STPLC	-	-	1
PENON	-	-	1
PENOF	-	-	1
DSH	-	-	1
NODSH	-	-	1
LOP	1	1	-
STPLP	-	-	1
LINE			
PT1 & PT2, or V1 & V2	4	6	-
PT2 only, or V2 only	2	3	-
LABEL	(no. words in output buffer filled) +1	1	-
MOVBM	2	3	-
PLACE			
SHRNK	1	1	-
MOV	2	3	-
XMOV	1	1	-
YMOV	1	1	-
ZMOV	1	1	-
XTURN	1	1	-
YTURN	1	1	-
ZTURN	1	1	-
IMCAL	1	1	-
CALL	(no. args.)+1	(no. args.)+1	-
IFPEN	2	2	-
IFNEG	2	2	-
IFZRO	2	2	-
GO	1	1	-

*1 additional word is needed and used by AFDSP to specify the end of the image.

SUMMARY OF AFDSP FACILITIES

Calls

RSET (Name, size)	Initialize image buffer
IMCON (Name, item (args), loc/err)	Include unchangeable image item
IMVAR (Name, item (args), loc/err)	Include dynamic image item
SHOW (Name, rate)	Display image
NOSHO	Stop display
SETIO (Buff, n)	Initialize text buffer
TABL (Pic, Buff, Loc)	Include X-Y pair in table
TABLF (Pic, Buff, Loc)	
STPLC	End sub-space
PENON	Initialize pen detection
PENOF	Disenable pen detection
DSH	Initialize dash mode
NODSH	Discontinue dash mode
LOP (n)	Loop in image
STPLP	Loop end delimiter

Items

LINE (Point1, Point2)	Visible line item
LABEL (Buff, Mode)	Text item
MOVBM (Point 1)	Move beam position
PLACE (Size, location, rotation)	Start sub-space
IMCAL (Name)	Call sub-image
CALL (Subrtn, args,...)	Call sub-program
IFPEN (Loc)	Conditional image-jump on pen
IFNEG (Var, loc)	Conditional image-jump on negative value
IFZRO (Var, loc)	Conditional image-jump on magnitude
IFNEG (Var, loc)	Conditional image-jump on magnitude
GO (loc)	Unconditional image-jump

Functions

RMV (expr)	For dynamic value assignments
IMV (expr)	For dynamic value assignments
PT1 (x1, y1, z1)	For LINE and MOVBM arguments
PT2 (x1, y1, z1)	For LINE and MOVBM arguments
SHRNK (s)	For PLACE arguments
MOV (dx, dy, dz)	For PLACE arguments
XMOV (dx)	For PLACE arguments
YMOV (dy)	For PLACE arguments
ZMOV (dz)	For PLACE arguments
XTURN (rx)	For PLACE arguments
YTURN (ry)	For PLACE arguments
ZTURN (rz)	For PLACE arguments
ISVA (ARRAY)	Save address of array name
RSVA (ARRAY)	Save address of array name
ZSET (z, name)	Set Z value for table
TDFUN (x, y, n1, n2)	Enter coordinate pair table

Library Routines

FNSIO (sw, op, value/subroutine)	Read or set function sw or execute when set.
RVCD1 (dial, value)	Read a Variable Control Dial
RVCD6 (val1, val2, val3, val4, val5, val6)	Read all Variable Control Dials
RADTV (hor, ver, down, press)	Read Analog Data Tablet
PENHT (n, item, image)	Identify nth previous pen-hit

USE OF AFDSP ON 8K SYSTEMS

Version 1 of AFDSP is used on 16 or 32K AGT Systems. Version 2 is segmented and may be used on 8K AGT Systems. Version 2 consists of two parts, as follows:

Part 1 (Title "ADSP1") containing routines for building display buffers: All AFDSP routines except SHOW, NOSHO, ISVA, RSVA.

Part 2 (Title "ADSP2") containing routines to run displays: SHOW, NOSHO, IMV, RMV, ISVA, RSVA.

On 8K systems, a RECALL-OVRLY scheme may be used so that OBJPK, AFDSP, etc., and DSPLY are not residing in core at the same time, as follows:

```
      C     DRIVER PROGRAM
                SUBROUTINE MAIN
      C     DIMENSION DISPLAY BUFFERS
                DIMENSION BF1(n), BF2(n)
      C     LOAD AND EXECUTE BUFFER AND BUILD ROUTINE
      C     MAKE CONTAINS IMCON, IMVAR, ETC., CALLS
      C     BUT NOT SHOW CALLS
                RECALL-OVRLY: MAKE (BF1, BF2)
      C     WATCH HAS SHOW CALLS
      C     OVERLAY MAKE WITH SHOW, ETC. AND DSPLY
                RECALL-OVRLY: WATCH (BF1, BF2)
      C     DUMMY IF STATEMENT TO LOAD RUN-TIME ROUTINES
                IF (1) 2, 2, 1
2               CALL RTSR1 (args)
                CALL RTSR1 (args)
1               RETURN
                END
```

adage

AFORT

(AUGMENTED ASA BASIC FORTRAN)

Programmer's Reference Manual

Revision C

July 1969

TABLE OF CONTENTS

INTRODUCTION

AFORT is an extension of ASA Basic Fortran, and its source language conforms to the specifications proposed by the American Standards Association X3.4.3 subcommittee for Basic Fortran.*

The AFORT compiler is designed for use under the AMOS operating system. It accepts source input text of type ATEXT. The object programs are output in standard AMOS relocatable linkage format, compatible with those output by the ADEPT assembler.

AFORT includes the following extensions to ASA Basic Fortran:

1. Symbolic names specified in a GLOBAL statement may be referenced by programs written in ADEPT, or by expressions in AMOS Monitor control statements.

2. Symbolic names may be declared as function or subroutine names by an EXTERNAL statement.

3. Available core storage may be shared between disjoint program sets by a RECALL-OVRLY statement.

4. Selective symbolic tracing of program execution at the source language level is possible with TRACE statements.

5. In assignment statements, the first element of an array may be referenced by the array name alone.

* These specifications may be found in the Communications of the ACM, Volume 7/Number 10/October 1964.

HARDWARE REQUIREMENTS

The required hardware for the AFORT compiler is:

SCU-P1, ACC-P1, OPC-P1, DME-P8, P16, P32, and
MTP5/8 or MTP7;
DPR1-P2, P3, P4, and MTP5/8 or MTP7;
AGT/1Ø, M1Ø-P1, and (MTP5/8 or MTP7 or DMS2)
AGT/5Ø or AGT/3Ø, and (MTP5/8 or MTP7 or DMS2)

SOFTWARE REQUIREMENTS

AMRMX and PRIO (along with relocatable monitor symbol links, ARMSX) which correspond to the hardware configuration.

NOTE: Execution of compiled programs requires facilities implemented by the "Object Package" (OBJPK) and possibly subroutines from the library and Fortran library, FLIBR.

CHARACTER SET

An AFORT program may contain the following letters, digits and special characters:

LETTERS: A B C D E F G H I J K L M
N O P Q R S T U V W X Y Z

DIGITS: 0 1 2 3 4 5 6 7 8 9

ARITHMETIC OPERATORS:

+	Addition or positive value
-	Subtraction or negative value
*	Multiplication (** exponentiation)
/	Division

SPECIAL CHARACTERS:

	Blank or Space
=	Equals
(	Left parenthesis
)	Right parenthesis
,	Comma
.	Decimal point
:	Colon

TAB — The first "TAB" character in a line sets the column counter to Column 7. Subsequent "TAB" characters in the same line are ignored.

C/R — The Carriage Return character is used to terminate a line. A C/R during AFORT teletype I/O specifies an "End of Record."

↑ @ %] & $ # — permitted in Hollerith fields

LINE FORMAT

The columns in an AFORT line are used as follows:

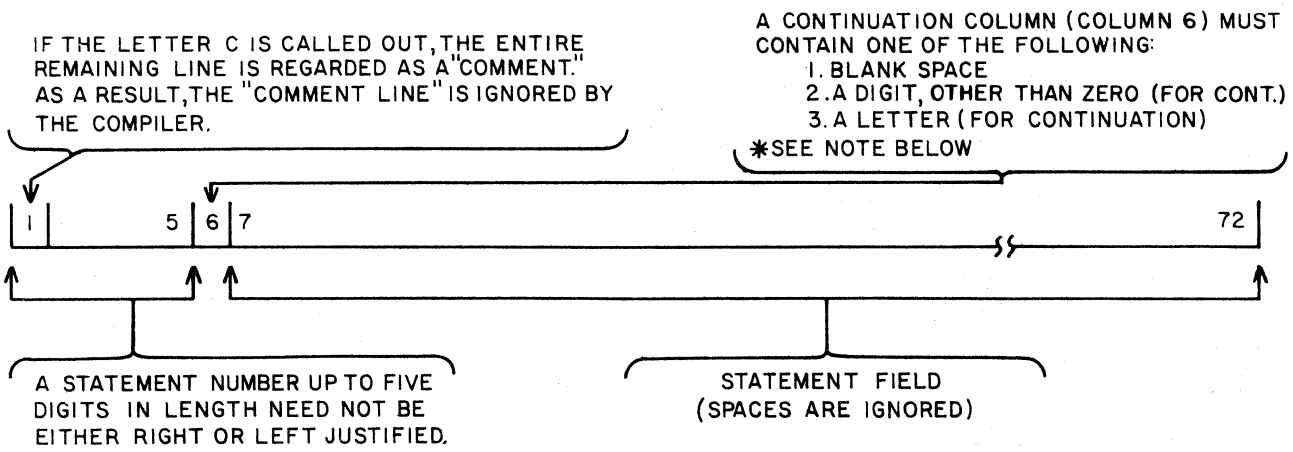


FIGURE I. SAMPLE AFORT LINE FORMAT

NOTE: An AFORT statement may contain up to 5 continuation lines, subject to the following restrictions:

1. DO statements must be written on only the first line.
2. The equals character (=) of a replacement statement must appear on the first line.
3. A statement label (if any) must appear on the first line.

STATEMENT FORMS

A. Rules

1. [X] means that "X" is optional.
2. $\{X\}_y$ means that "X" can occur one or more times, separated by "y" ("y" may be null).
3. $\{X\}_y^z$ means that "X" can occur one or more times up to a maximum of "z" times, separated by "y".

B. Definitions

1. ch is any character
2. n and format are statement-label numbers
3. octal is one to four octal digits
4. r is the number of times a format field specification is to be repeated
5. w is the field width (in characters) to which a format specification applies
6. d is the number of characters to be assumed after the decimal point
7. int is an integer constant or integer name
8. list element is either a variable or:

   ({list element}, [integer name = int, int[,int]])
9. spec is a formal specification and may be the following:

   a. rFw.d
   b. rEw.d
   c. rIw
   d. wH $\{ch\}^w$
   e. r (spec)
   f. wX
   g. /

10. sub is an integer constant used as subscript

11. unit is a logical I/O unit number:

    a. 1 through 4 for physical tape drives Ø through 3, respectively.
    b. 5 through 8 for the AMOS Format Library on tape drives Ø through 3 respectively.
    c. 21 through 24 for physical disk volumes last assigned (by a SDVOL call).
    d. 5Ø for the console teletype.
    e. 52 for the Card Reader
    f. 53 for the Printer
    g. 54 for the current ASCII buffer (set by SETIO call).
    h. 55 for the high speed paper tape.

C. AFORT Statements

DIMENSION {name(sub[, sub])},

Purpose: Name and declare size(s) of array(s).

Restrictions: Specification statements must appear in the order of DIMENSION, COMMON, EQUIVALENCE.
May only be preceded by a SUBROUTINE, FUNCTION, or other DIMENSION statement.
A name, if dimensioned here, must not have previously been dimensioned.

Example: DIMENSION A(5), I2(3, 6), CP(22)

COMMON {name},

Purpose: Specify that the variables and/or arrays listed are to be assigned to storage in the memory area called COMMON. Allow more than one program unit in an executable program to reference the same data directly.

Restrictions: Specification statements must appear in the order of DIMENSION, COMMON, EQUIVALENCE.
May only be preceded by a SUBROUTINE, FUNCTION, DIMENSION, or other COMMON statement.
The program unit with the largest common region must be loaded first.

Example: COMMON X, ANG, I2, IND

EQUIVALENCE {(name[(sub[, sub])], {name[(sub[, sub])]},) },

Purpose: Cause the same area of memory to be shared by two or more entities.

Restrictions: Specification statements must appear in the order of DIMENSION, COMMON, EQUIVALENCE.
May only be preceded by a SUBROUTINE, FUNCTION, COMMON, or prior EQUIVALENCE statement.
Effective lengthening of COMMON is permitted only if it increases COMMON in the same direction as additional COMMON elements would.
Two elements of the same array can not be equivalenced.

Example: EQUIVALENCE (X, A(2), Y), (B, I2)

EXTERNAL {name},

Purpose: Allow SUBROUTINE or FUNCTION names to be used as arguments to yet other subprogram calls.

Restrictions: Names must not be dimensioned. Names previously used must be used only in program definition.
Must follow any DIMENSION, COMMON, or EQUIVALENCE statements.

Example: EXTERNAL SAM, TIGER

GLOBAL ({name},)

Purpose: Allow symbolic names to be referenced by programs written in ADEPT or by expressions input with AMOS Monitor control statements.

Restrictions: Must follow any DIMENSION, COMMON, EQUIVALENCE, or EXTERNAL statements (or be embedded in the executable statements). GLOBAL variables may not be dummy arguments. Only one GLOBAL statement is permitted in a program or subprogram.

Example: GLOBAL (ALICE, PAT, SUE)

[n] variable = expression

Purpose: Replace the value of a variable with the results of the evaluation of an expression.

Restrictions: An "equals" sign must follow the first operand.

Example: A(JJ) = 7.*A(3)/2.

[n] GO TO n

Purpose: Transfer control to the statement labeled "n".

Example: GO TO 5

[n] GO TO ({n},), integer name

Purpose: Transfer control to the 1st, 2nd, ... statement with a statement number, "n", depending on whether "integer name" is 1, 2, ..., respectively.

Restrictions: The value of "integer name" should not exceed the number of statement numbers in the parenthesized list.

Example: GO TO (5, 22, 3Ø56, 31), ISAM

[n] IF (expression) n, n, n

Purpose: Transfer control to the statement numbered "n", depending on whether the value of "expression" is less than zero, equal to zero, or greater than zero (respectively, from left to right).

Restrictions: Three statement numbers, not necessarily different, must be given.

Example: IF (S -3/2*X) 1Ø, 22, 1Ø

[n] CALL name [({expression},)]

Purpose: Transfer control to a subprogram and present it with any parenthesized arguments.

Restrictions: CALLs are not recursive.

Example: CALL TES1 (A, B*C+6., I(2))

n CONTINUE

Purpose: Serve as a program unit reference point. May be last statement in a DO range when the loop would otherwise end with an IF or GO TO statement (which is illegal).

Example: 13 CONTINUE

[n] PAUSE octal

Purpose: Causes a temporary halt of program execution, which may be resumed (with the next statement) by depressing PULSE1, and also types "PAUSE octal."

Note: An external instruction on the console must be set to 24ØØØ77776 and be activated by PULSE1.

Example: PAUSE 24

[n] STOP octal

Purpose: Terminate program execution, type "STOP octal," transfer to the EXIT routine to type any object time error messages, and return control to AMRMX.

Example: 12 STOP 1375

[n] DO n integer name = int, int[, int]

Purpose: Repetitively execute the statements following the DO statement up to and including the statement labeled "n". "Integer name" is incremented (optionally by a value "int", greater than 1) from the value "int" (the first control parameter) until its value is greater than "int" (the second control parameter).

Restrictions: The terminal statement (labeled "n") must physically follow and be in the same program unit as the DO statement.
The terminal statement may not be a:

GO TO statement
IF statement
RETURN statement
STOP statement
PAUSE statement
DO statement

If the range of a DO loop contains another DO statement, the range of the contained DO loop must be a subset of the range of the containing DO loop. A GO TO or IF statement may not cause control to pass into the range of a DO loop from outside its range.
The control variable ("integer name") and/or control parameters ("int"'s) may not be redefined during execution of the range of the DO loop.
If a statement is the terminal statement of more than one DO loop, the label of that terminal statement may not be used in any GO TO or IF statement that occurs anywhere but in the range of the most deeply-contained DO with that terminal statement.

Example: DO 24 IP = 1, JP, 3

[n] READ (unit[,format])[{list element},]

Purpose: Input data from the next record on the external source "unit," according to the format specified by a statement labeled "format," and assign the values to the variable name(s) "list element(s)." (NOTE: The record size for formatted data is 12Ø characters. Unformatted records are 24 binary words in length.)

Restrictions: If only part of a record is input, the remainder is lost. Records are read sequentially until the list is exhausted and only enough values are read to fill the list.

Example: READ (5Ø, 2Ø6) A(2), B, I3, (II(N), N = 1, K, 3)

[n] WRITE (unit [, format]) [{list element },]

Purpose: Take values sequentially from the variable name(s) "list element(s)," convert these values according to the format specified by a statement labeled "format," and output records on the external device "unit." (NOTE: The record size for formatted data is 12Ø characters. Unformatted records are 24 binary words in length.)

Restrictions: Successive records are written only until the data is exhausted. If the data does not fill a record, the record is filled with blanks.

Example: WRITE (2, 28) A, II

format FORMAT ({spec})

Purpose: Describe the type of conversion and format of data to be used in the transmission of an input/output list.

Example: 5 FORMAT (/2X, 3F 1Ø.5, E15.8, /2HI = , I2)

[n] RECALL-OVRLY: name [({expression})]

Purpose: Permit a loaded program unit to allocate any remaining available storage to any of several disjoint programs. Functions or subroutines loaded in this manner are kept in memory until a subsequent overlay call from the same program unit.

Restrictions: The maximum depth of nested overlays allowed by AMRMX is 16.
When storage is released by a subsequent overlay call at the same level, all higher levels induced by the previous overlay call are also released.

Example: RECALL-OVRLY: TEST3 (A, B, C*D)

[n] REWIND unit

Purpose: Cause the object program to rewind tape "unit" to the BOT.

Restrictions: If "unit" is not a magnetic tape, no action is taken.

Example: REWIND 1

[n] BACKSPACE unit

Purpose: Cause the object program to backspace tape "unit" one record.

Restrictions: If "unit" is not a magnetic tape, no action is taken.

Example: 43 BACKSPACE 2

[n] ENDFILE unit

Purpose: Write a file mark on tape "unit," wherever the tape is positioned.

Restrictions: If "unit" is not a magnetic tape, no action is taken.

Example: ENDFILE Ø1

FUNCTION name ({name},)

Purpose: Define the name and dummy arguments of a FUNCTION subprogram.

Restrictions: The last statement executed in the FUNCTION subprogram must be a RETURN statement.
The symbolic name "name" must appear as an assigned variable in the subprogram, but must not appear in any nonexecutable statement in the subprogram.
A FUNCTION subprogram may not define or redefine any of its arguments nor any variable in COMMON.
The dummy arguments must correspond in type, number, and order with the actual arguments in the FUNCTION call.
If a dummy argument is an array name, the corresponding actual argument must be an array name.

Example: FUNCTION SALLY (A, B)

name ([{name},^{10}]) = expression

Purpose: Name and define a statement function.

Restrictions: May only be referenced within the program unit in which it is defined.
Arguments used in the references must agree in type, number and order with the corresponding dummy arguments.

Example: ANO (X, Y) = X**2. +5. *Y + 6

SUBROUTINE name [({ name },)]

Purpose: Define and name dummy arguments of a SUBROUTINE subprogram.

Restrictions: The symbolic name of the subroutine must not appear in any statement in the subprogram. The symbolic names of the dummy arguments may not appear in COMMON or EQUIVALENCE statements. The dummy arguments must correspond in type, number, and order with the actual arguments in the SUBROUTINE call. If a dummy argument is an array name, the corresponding actual argument must be an array name.

Example: SUBROUTINE AFL (PEL, MFL (11))

[n] TRACE name {, name}
[n] TRACE n
[n] TRACE n, name {, name}

Purpose: Trace (i.e., type values) of selected variables, or all variables and results of arithmetic expressions within a specified range, or both, during program execution. (NOTE: TRACE output may be terminated or restarted by depressing IC (Ø).)

Example: TRACE 51, A, B, C

[n] RETURN

Purpose: Return control back to the current calling program.

Restrictions: Each subprogram (Subroutine or Function) must have at least one RETURN statement (the last statement executed in the subprogram).

Example: 5 RETURN

END

Purpose: Indicate the end of a program unit (main program, function, or subroutine). (NOTE: A second END statement is required at the end of the last source program to be compiled.)

Example: END

COMPILATION INSTRUCTIONS

1. Load the AMOS Resident Monitor AMRMX (refer to AMRMX Operating Instructions for further details).

2. Type:

   START ("AFORT", sysunit)!

   to load the compiler, where "sysunit" is the tape drive or disk volume containing the relocatable program AFORT as well as the proper monitor links, ARMSX.

3. Type:

   SYST (output)!

   to specify that the output object files are to be appended to the library at the end of tape or disk volumes "output."

4. Mount a tape or disk with the AFORT source program texts in the "scratch pad" (File #1Ø) on tape unit Ø or in the currently assigned disk input volumes.

5. Type:

   FORTN!

NOTE: On 8K systems, AMRMX is overlayed by AFORT. The monitor may be reloaded after the compilation (control will be returned to the bootstrap loader by AFORT).

ALSO: If "BIG PAGES" are encountered during the compilation, AMRMX will be overlayed. The monitor may be reloaded after the compilation.

AFORT TELETYPE OUTPUT

During compilation, output in the following form is typed:

```
        name:

(any diagnostic messages--see next section)

PGRM SIZE nnnnn

TOTL SIZE nnnnn

CALLED SUBPROGRAMS

        subl
         .
         .
         .

name:  FILE NO. n

         .
         .
         .

END OF TEXTS
```

where:

name	=	subroutine or function name (="Title of text file" if not a subprogram)
PGRM SIZE	=	program length (no. words output)
TOTL SIZE	=	PGRM size + variable and temporary definitions
subl, etc.	=	programs called by "name" or names declared to be external in "name"
FILE NO.	=	relocatable file number output

NOTE: "name:" is typed at the point when AFORT is about to begin outputting from the first page of text. If errors were detected or the end of the program reached (e.g., a very short 1-page program) before this point, the corresponding messages may be typed before name.

FORMAT OF RELOCATABLE FILES

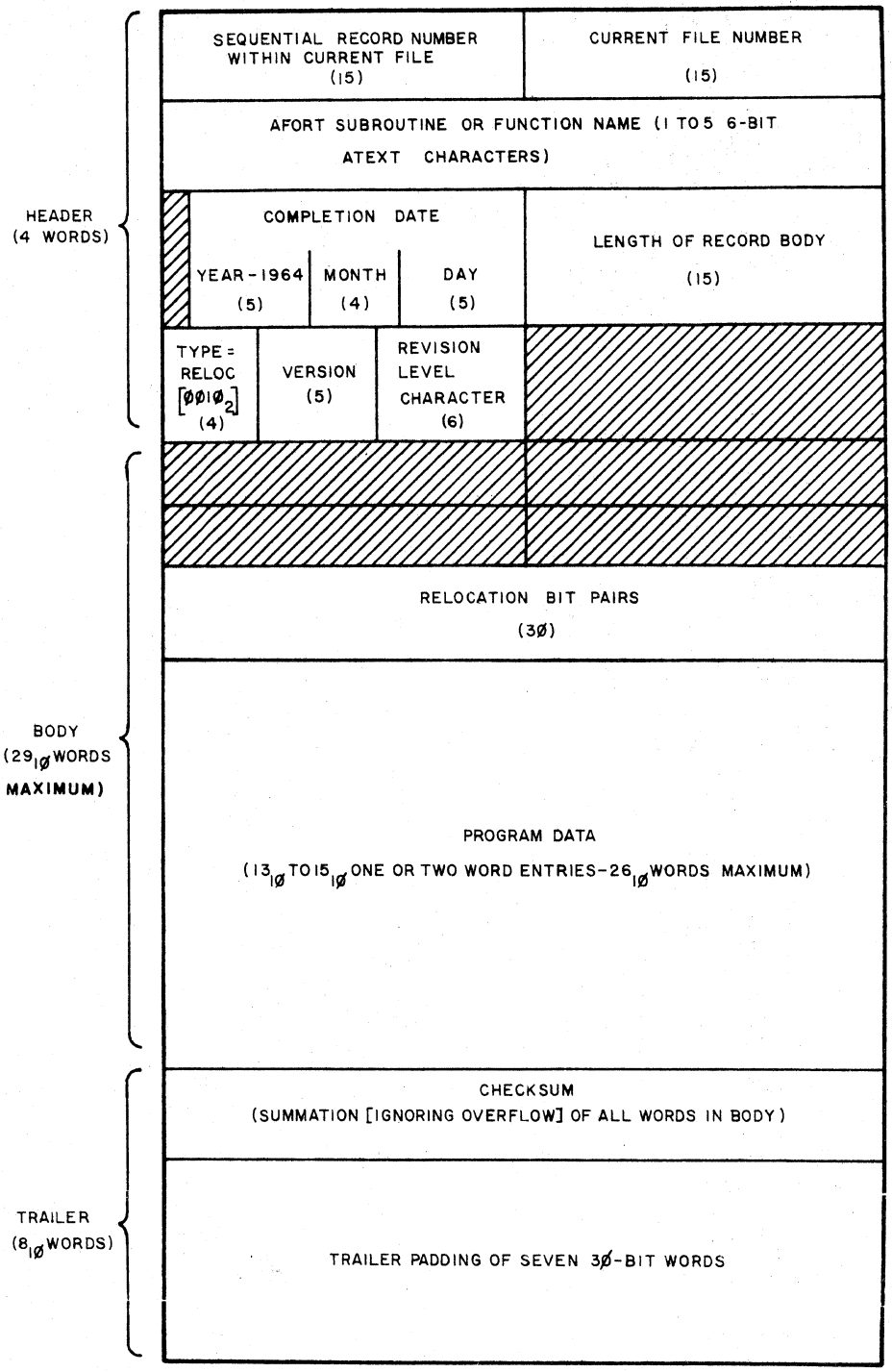


FIGURE II. FORMAT OF RELOCATABLE FILES

(See pages 17 and 18 for Relocation Bit Pairs and Special Sub-Codes.)

A. Relocation Bit Pairs

Relocation bit pairs refer to the next 13_{10} to 15_{10} data entries. This word must be rotated left one bit to get the code for the first entry in bits 28-29. Subsequent rotations of two bits left are needed for successive entry codes. The two bit codes describe the data entries as follows:

ØØ - Absolute loading of one word entry into next available memory location.

Ø1 - Load one word entry into next memory location and add the base address of the program (or subroutine) to bits 15-29.

1Ø - Load one word entry into next memory location and add the base address of common storage to bits 15-29.

11 - Look at first 6 bits of first entry word and perform the operation indicated (see Special Sub-codes). This entry may be one or two words long depending on Sub-code.

B. Special Sub-Codes

1. Two-word entries:

$Ø1_8$ - Subprogram Entry--name and address
WORD 1 - bits 15-29: Relocatable address
WORD 2 - bits Ø-29: Name (1 to 5 ATEXT characters)

$Ø2_8$ - Subroutine call name and chain address
WORD 1 - bits 15-29: Last address of subroutine call chain
WORD 2 - bits Ø-29: Name (1 to 5 ATEXT characters)

$Ø3_8$ - Common block request
WORD 1 - bits 15-29: Length of common block
WORD 2 - (Present but not used)

$Ø4_8$ - Chain continuity and address
WORD 1 - bits 15-29: Last address of word usage chain
WORD 2 - bits 15-29: Assigned address of word

2. One-word entries:

11_8 - Relative origin
WORD 1 - bits 15-29: New load address (relative to program origin)

12_8 - Global
WORD 1 - bits 15-29: Start address of global definition

13_8 - Absolute origin
WORD 1 - bits 15-29: New load address (absolute location)

60_8 - End of program
WORD 1 - bits 15-29: Length of program

(NOTE: The last record of a relocatable file is padded with End of Program entries. Only the first of these entries has the correct program length in its entry word.)

AFORT ERROR DIAGNOSTICS

Any errors encountered in a program during compilation are noted by diagnostic message output on the typewriter. The AFORT statement in which an error is detected is typed in the message.

Two kinds of diagnostic messages are typed by the Compiler--notification and termination. An error which causes a notification message does not affect the compiled program. An error which causes a termination message causes the object program output to be terminated (aborted after the previous file). The remainder of the program is then interrogated and any further diagnostic messages are given. The remainder of the program is not compiled into object output.

The diagnostic messages typed are of the following form:

en stn + n: stmt

where "en" is one of the error types shown in the following table, "stn" is the last statement label encountered prior to the error, "n" is the number of statements following the last statement label, and "stmt" is the portion of the statement in which the error was detected.

A. Notification Errors

ERROR 1	Improper statement construction
ERROR 2	Improper usage of variable
ERROR 4	Illegal statement for DO termination
ERROR 5	Improper statement label usage
ERROR 6	COMMON base less than Ø
ERROR 7	Illegal EQUIVALENCE grouping
ERROR 8	Reference to a non-executable statement
ERROR 9	No path to this statement
ERROR 1Ø	Doubly defined statement label
ERROR 11	Invalid FORMAT construction
ERROR 12	Spelling error
ERROR 13	FORMAT with no statement label
ERROR 14	Function not used as variable
ERROR 15	Real constant had to be truncated

B. Termination Errors

ERROR 1T	Improper statement construction
ERROR 2T	Improper usage of variable
ERROR 3T	Too many identifiers
ERROR 4T	Illegal statement
ERROR 5T	Improper use of variable name
ERROR 6T	Too many statement function parameters
ERROR 7T	Invalid mode
ERROR 8T	Constant too large
ERROR 9T	Improper DO nesting
ERROR 1ØT	Magnetic tape fail (parity error, off line state, etc.)

APPENDIX A

NUMBER LIMITATION

This appendix outlines the method by which numbers are treated. If the range limitations are exceeded at compile time, a terminating error will occur. An "INPUT ERROR" followed by a call to the "EXIT" routine will occur at object time if the oversize is detected during data input. If the oversize is the result of a computation at object time, the "OVERFLOW" flag will be set and a zero returned as the result of the computation. Unless tested at some other point in the object program, the "OVERFLOW" flag will be tested in the "EXIT" routine, with a typeout occurring if it is set. Numbers are input either at compile time or object time.

INTEGERS

Integers may be in the range $\emptyset \leq |\ I\ | \leq 536870911$ for both input or computation.

REAL NUMBERS

Real numbers must line in the range $2^{-217}(-1+2^{-21}) \leq R \leq 2^{127}(1-2^{-21})$ (approximately, $-\emptyset.588 \times 10^{-38} \leq R \leq \emptyset.588 \times 10^{+38}$). On input, numbers outside the range will produce errors, while only numbers greater than the range will produce computational errors. If the result of a computation is less than 2^{-127}, zero will be used as the result, with no error flag.

When input, any real number can be considered to be of the form I. E±n, where I is a decimal integer and n is the decimal exponent. If I is in the range 1ØØØØØØØØX $\leq$ I $\leq$ 536870911X (where X is a string of decimal digits), the X digits will be dropped and the exponent increased by the number of digits in X.

e.g., 1234567897234. E+2 will be treated as 123456789. E+6

If I is in the range 53687Ø91YX $\leq$ I $\leq$ 99999999X (where Y is a decimal digit greater than I), both Y and X digits will be:

e.g., 536870917. E+3 will be treated as 53687Ø91. E+4

After input (and truncation if necessary), the resulting decimal real number (I. E±n) is converted to a binary real number in the form e.f, where "e" ($1 \leq e \leq 377_8$ is the actual binary exponent plus $2ØØ_8$ so that both positive and negative exponents are represented by a positive e), and "f" (the binary mantissa) is a 21-bit number in the range $1/2 \leq f \leq 1-2^{-21}$. The conversion from I. E±n to e.f is done to 29 binary bits of significance and the result truncated to 21 binary bits.

The basic arithmetic operations on real numbers (+, -, *, /) are performed to 27 bits of significance, with a ONE being added to the 22nd bit prior to truncation to 21 bits.

ZERO

AFORT makes no distinction between +Ø and -Ø, treating both as an algebraic zero for computational and test purposes, with the result that any operation or routine may return either +Ø or -Ø for a correct Ø answer. In addition, for any real number, e.f, if e is Ø, f will also be Ø.

APPENDIX B

AFORT INTERNAL CHARACTER CODES

ATEXT internal code is converted to the following internal codes by--and for use by--the compiler. As AFORT outputs relocatable tape files, "names" are converted to standard AMOS code, and Hollerith strings from format statements remain unchanged.

CODE	TTY CHAR.	CODE	TTY CHAR.	CODE	TTY CHAR.	CODE	TTY CHAR.
ØØ	SPACE	2Ø	A	4Ø	W	6Ø	+
Ø1	Ø	21	B	41	X	61	-
Ø2	1	22	C	42	Y	62	*
Ø3	2	23	D	43	Z	63	/
Ø4	3	24	E	44	↑	64	.
Ø5	4	25	F	45	@	65	(
Ø6	5	26	G	46	%	66	)
Ø7	6	27	H	47	]	67	,
1Ø	7	3Ø	O	5Ø	I	7Ø	=
11	8	31	P	51	J	71	&
12	9	32	Q	52	K	72	:
13	...	33	R	53	L	73	...
14	...	34	S	54	M	74	$
15	...	35	T	55	N	75	#
16	...	36	U	56	...	76	...
17	...	37	V	57	...	77	C/R

NOTES

"..." indicates that this internal code is not used in the AFORT system.

A code may yield a character different from the one listed above if a device other than a TTY is used (e.g., a printer).

DESCRIPTION

AMLDX is an AMOS system program used to load absolute programs of type (BIN) from AMOS formatted magnetic tape or disk. AMLDX occupies core storage locations 0 through 117_8. AMLDX is intended to reside in memory during the execution of all other standard programs. AMLDX Version 5 may only be used to load the disk resident monitor, AMRMX Versions 13, 14, 15, or 16.

HARDWARE REQUIREMENTS

Version 1: MTP5, MTP8

Version 2: MTP7-DDC1

Version 5: DMS2-P1

AMLDX is short enough to be conveniently loaded from punched paper tape via the standard "BOOTSTRAPPING" procedure, see DPR2/SOI.

CALLING SEQUENCE AND USE

For operational procedure, consult AMLDX Software Operating Instructions, AMLDX/SOI.

INTRODUCTION

AMLPP is an AMOS system program which provides the programmer with open and closed subroutines for printing lines on the LPR1 Line Printer subsystem.

HARDWARE REQUIREMENTS

AMLPP assumes a DPR1 or DPR2 with LPR1-P1/P2/P3 subsystem.

SOFTWARE REQUIREMENTS

AMLPP is intended for use as an AMOS library program under the AMRMX AMOS Resident Monitor.

INITIALIZATION

The calling sequence:

```
JPSR    $RSPNT
```

resets all printer pivots, flags, and locks.

USE

The AMOS Line Printer Program, AMLPP, defines four entry points: PRINT, STPNT, REPNT, RSPNT. The calling sequence:

```
JPSR    $PRINT      · Complete print before return
        FMT         · Form control code
        FETCH       · Character-fetch instruction or Buffer address
        NUM         · Number of columns to print
        PI          · Error instruction
        OFLN        · Off-line instruction or idle code or wait code
                    · Return
```

initiates the closed (serially executed) call to the print program. The value FMT contains the skip control code in bits 24-29, see Table 1. The value, FETCH, is the execution instruction for getting characters if bits $\emptyset$-14 $\neq \emptyset$ or, it is a pointer to a packed character list if bits $\emptyset$-14 = $\emptyset$. The value, NUM, is the number of characters to be printed. The value, PI, is the instruction to be executed if there is a

parity error. The value, OFLN, is the instruction to be executed if the printer is off-line and OFLN $\neq \emptyset$ or $-\emptyset$. If OFLN = $\emptyset$ then the program idles, waiting for the on-line state. If OFLN = $-\emptyset$, an interrupt instruction is set up and control is returned to the calling program until the printer is returned to its on-line state.

The calling sequence:

```
JPSR    $STPNT      · Start printing in parallel with execution
        FMT         · Form control code
        FETCH       · Character-fetch instruction or Buffer address
        NUM         · Number of columns to print
        DI          · Printing completed instruction
        PI          · Error instruction
        OFLN        · Off-line instruction or idle code or wait code
                    · Return
```

initiates the open (executed in parallel) call to the print program. All argument values correspond to those in the call to PRINT, except for DI which is the instruction to be executed when printing is completed.

The calling sequence:

```
JPSR    $REPNT
```

re-calls the PRINT or STPNT subroutine with the same arguments as last used. The definition of this entry, REPNT, is provided for use as the error routine argument, PI, in the above calls and will cause continued attempts at correct printing of the line to be made.

TABLE 1. FORMAT CHARACTER CODES

Skip to:	24	25	26	27	28	29
VFU Channel 1 (TOF)	1	0	0	0	0	0
VFU Channel 2	1	0	0	0	0	1
VFU Channel 3	1	0	0	0	1	0
.			.			
.			.			
. (if used)			.	binary sequence		
.			.			
VFU Channel 8	1	0	0	1	1	1
No line feed	0	0	0	0	0	0
1 line space	0	0	0	0	0	1
2 line space	0	0	0	0	1	0
.			.			
.			.			
.			.	binary sequence		
.			.			
31 line space	0	1	1	1	1	1

adage

AMRMX

AMOS RESIDENT MONITOR

Programmer's Reference Manual

Revision B

June 1969

PREFACE

AMRMX (Magnetic Tape System) is the ADEPT format text which generates any of eight versions of the AMOS Resident Monitor. The six versions have the following hardware configurations:

1. SCU, OPC, ACC, MTP-5, DME (8, 16, or 32K)
2. SCU, OPC, ACC, DDC-1, MTP-7, DME (8, 16, or 32K)
3. DPR-P2, P3, or P4, MTP-5
4. DPR-P2, P3, or P4, DDC-1, MTP-7
5. SCU-TTY, ACC, MTP-5, DME (8, 16, or 32K)
6. SCU-TTY, ACC, DDC-1, MTP-7, DME (8, 16, or 32K)

11. AGT/10, M10-P1, PRI1-P2, MTP5-P1 or MTP8-P1 or AGT/30, PRI1-P2, MTP5-P1 or MTP8-P1 or AGT/50, PRI1-P2, MTP5-P1 or MTP8-P1
12. AGT/10, M10-P1, PRI1-P2, MTP7-P1, DDC1-P1 or AGT/30, PRI1-P2 MTP7-P1, DDC1-P1 or AGT/50, PRI1-P2, MTP7-P1, DDC1-P1

The AMOS Resident Monitor, the core resident section of the AMOS programming system, includes operator control statement handling routines, a relocatable, linkable loader for loading other portions of the AMOS programming system into memory when needed, various program debugging aids, magnetic tape subroutines, and routines to process other often-called system operations.

TABLE OF CONTENTS

INTRODUCTION

The AMOS Resident Monitor (AMRMX) is that portion of AMOS which accepts and interprets AMOS control statements, loads programs requested (if necessary), and executes all programs.

The Monitor is called "resident" because it is kept in core memory at all times during the use of AMOS, whereas all other programs are loaded into memory only as they are needed to execute "control statements".

The Monitor lives up to its name by continuously monitoring the operator's console, and responding to control statements entered therefrom by means of the keyboard or the punched tape reader (in versions 1 and 2). This monitoring function may be performed as a "background" task to another "foreground" program being executed, on a time-shared basis. In other words, while another program is being executed under control of AMOS, the Monitor will accept and execute control statements not in conflict with the real-time requirements of the running "foreground" program.

The function of the Monitor has been stated above -- it accepts, interprets and executes AMOS control statements furnished by the system operator. Each such control statement causes the execution of a previously-written program stored in memory, with the parameter values indicated by the control statement. The general form of an AMOS control statement is:

OPERATION (PARAMETER 1, PARAMETER 2,... PARAMETER N)!

where each parameter may be an octal/symbolic expression. Unneeded parameters may be omitted from a control statement, provided only that all of the parameter-separating commas preceding the last parameter used are given.

When such a control statement is input, the Monitor causes the program corresponding to the OPERATION to be executed, with the parameters indicated, after which control is returned to the Monitor. If the requested program is not currently in memory, the Monitor will first load this program from magnetic tape into core memory, together with the subroutines which it requires. Symbolic names of entry points and externally-referenced parameters are also added to the Monitor's symbol list during such loading, and an up-to-date memory map is available at all times.

The Monitor loader provides memory allocation control to permit dynamic overlaying of one or more nested levels of loaded programs. This permits any level of loaded programs to share the remaining available storage among any of several independent program sets. This permits running of program sets which would exceed core capacity if all necessary levels were requested as one memory load.

The Monitor also includes a processor interface package used for outputting mag tape files by AFORT and ADEPT.

The Monitor is loaded into memory from magnetic tapes by the AMLDR routine. AMRMX occupies octal locations 7ØØØØ thru 77777 (including all interrupt pivot locations).

CONTROL INPUTS

A. Control Statements

For every control input statement

TERM!

or

TERM (ARG1,... ARGN)!

the Monitor builds a standard calling sequence in available memory as follows:

Location	Contents	Explanation
LL	JPSR $TERM	Call to Subroutine
LL+1	ARG1	Argument-List Values
	:	(Optional)
LL+NN	ARGN	
LL+NN+1	Ø	End of Argument-List Flag
LL+NN+2	JUMP MON9	Return to Monitor

After the statement has been input, and the program with entry point TERM loaded (if necessary), the Monitor transfers control to this calling sequence.

An input or output of the character ? will re-start the control statement in which it occurs.

B. Arguments

For those programs requiring arguments, the Monitor will accept a list of inputs (enclosed in parentheses), and produce the corresponding entry in the generated calling sequence.

Each argument is one of the following:

15-bit Address
3Ø-bit Value
Null (omitted) Argument
Sub-Argument List
Undefined Symbol
Multi-Word String

Subroutines to be called on-line via Monitor control statements may be written to interpret the Monitor's encoding of the above options permitting the user full flexibility of assembly-like input arguments.

It is the responsibility of the user to ensure that the number and type of argument in each control statement correspond to the subroutine being called thereby. Within this constraint, one may type command statements with 3ø-bit expressions, quoted character strings, or parenthetically nested argument structures as arguments, or with omitted optional arguments. When arguments are omitted, all separating commas up to the last non-omitted argument must be given (final commas in a list may be omitted), since arguments are identified by their position in the generated calling sequence.

An input or output of * will re-start the current argument.

1. Expressions

Each 15-bit address or 3ø-bit value is entered as an expression. Expressions consist of terms separated by arithmetic operators, and are evaluated on a term-by-term basis, as they are input to the Monitor. The resulting 3ø-bit value is placed in the generated calling sequence.

a. Terms

A term is one of the following:

(1) Octal Number: From 1 to $1ø_{10}$ octal digits. Its value is the $3ø_{10}$-bit right-justified octal integer it represents.

(2) Decimal Number: From 1 to 9_{10} decimal digits followed by a period ".". Its value is the $3ø_{10}$-bit right-justified decimal integer it represents.

(3) Symbol: From 1 to $1ø_{10}$ alphanumeric characters, defined in the Monitor's symbol table. Its value is its $3ø_{10}$-bit definition in the symbol table.

(4) One-Word String: From 1 to 5 characters, enclosed in quotation marks ("). The value of the term is its 6-bit internal character code representation, left-justified and filled with nulls ($øø_8$), if less than 5 characters in length.

(5) Specific Values: During memory interrogation and change certain characters may be used in expressions as terms having specific values:

.	Has the value of the current OPEN location
¢ or]	Has the value of the contents of the current OPEN location

b. Logical Arithmetic Operators

The value of an expression is obtained by combining the values of its terms according to its logical and arithmetic operators. (The initial expression, or preceding expression for the first term, is assumed to be +Ø.) These operators are as follows:

Operator	Action
+, tab, space	Add the value of the following term to the value of the preceding expression.
-	Subtract the value of the following term from the value of the preceding expression.
± or ↑	Inclusive OR the value of the following term with the value of the preceding expression.
&	AND the value of the following term with the value of the preceding expression.
'	EXCLUSIVE OR the value of the following term with the value of the preceding expression.
!B	Bit (1-bit) left rotate the value of the preceding expression.
!K	Character (6-bit) left rotate the value of the preceding expression.
!H	Half-word (15-bit) left rotate the value of the preceding expression.

2. Null Arguments

A null argument is one that is omitted. It is represented in the generated calling sequence by the value -Ø.

3. Sub-Argument List

An argument-list nested in parentheses may be used as an argument. It is represented in the generated calling sequence by a 3Ø-bit word whose upper 15-bits contain the number of arguments in the sub-argument list; the lower 15-bits contain a pointer to the first sub-argument (i.e., its address, decremented by 1).

4. Undefined Symbol

An undefined symbol consists of a group of from 1 to $1\emptyset_{10}$ alphanumeric characters, not defined in the Monitor's current symbol table. It is represented in the generated calling sequence by a 3Ø-bit word whose upper 15 bits are -Ø; the lower 15 bits are a pointer to the 2-word symbol string (i.e., its starting address).

5. Multi-Word String

A multi-word string consists of 6 or more characters enclosed in quotation marks ("). It is represented in the generated calling sequence by a 3Ø-bit word whose upper 15 bits contain the negative of the number of words in the string; the lower 15 bits are a pointer to the symbol string (i.e., its starting address).

USE OF BUILT-IN MONITOR STATEMENTS

A. System Parameter Specification

1. Control Input Device (TYPEC, TAPEC)

Under versions AMRM1 and 2 only.

TYPEC!

Causes subsequent control statements to be accepted from the console typewriter until the next occurrence of the following statement.

TAPEC!

Causes subsequent control statements to be read from punched tape until the next occurrence of the preceding statement.

2. System Tape for Processor Output (SYST)

SYST (UNIT)!

Assigns tape on tape drive UNIT as the current system tape.

3. Register-List Control and Interrogation (PANEL)

PANEL ((Register-List))!

Replaces the saved register-list with new values, any of which may be omitted (if a new value is omitted, the previous value is not changed). The Register-List consists of the following:

Version	Register-List
1, 2	AR, BR, IC, TPT, OVF, RTI
3, 4, 5, 6, 11, 12	AR, BR, IC, TTYBT, TTYCH, OVF, RTI
where	
AR, BR, IC	are the saved contents of the AR, BR, IC registers
TPT	is the TPT interrupt pivot, 77774
OVF	is the AR overflow pivot, 77771
RTI	is the clock interrupt pivot, 77775
TTYBT	is the teletype bit pivot, 77750
TTYCH	is the teletype character pivot, 77751

PANEL!

Causes the current saved values of the Register-List to be typed out.

4. Memory I/O and Search Bounds (BOUND)

BOUND (FIRST, LAST)!

Sets FIRST and LAST addresses for processing by other control statements. If FIRST is not specified, it will be set, each time it is referenced, to the then current lower limit of available memory. If LAST is not specified, it will be set, each time it is referenced, to the then current upper limit of available memory. The FIRST and LAST addresses can also be set by the other bound-setting statements, which are LIST, SRCHW, SRCHA, and WRITE.

5. Output Format (MODE)

MODE ("TYPE")!

Sets the format(s) desired for subsequent typeouts of internal machine-word images. The available (built-in) formats and their type-codes are:

Type-Code	Format
O (Octal):	The octal number contained in the "upper half-word" Space The octal number contained in the "lower half-word"
S (Symbolic):	Symbolic name of this location (followed by colon), if any Tab A symbolic expression whose value is contained in the "upper half-word" Space A symbolic address expression whose value is contained in the "lower half-word"
A (Characters):	Five characters whose standard AMBILOG 200 internal bi-octal codes are in the word
F (Full):	All of the above

Any combination of O, S and A is also recognized. Whenever mode "S" is selected, locations are typed out one per line. Otherwise, the resulting type-out is given in several columns across the page.

6. Magnetic Tape Density Selection (DSET)

DSET (DSET) !

Causes the selection of magnetic tape density to be DENS (0 = 200BPI, 1 = 556BPI, 2 = 800BPI) for subsequent magnetic tape operations. (MTP7 systems only)

B. Memory Interrogation and Change

1. Memory Search (SRCHW, SRCHA)

SRCHW (WORD, MASK, FIRST, LAST, "MODE")!

Will list, in formats MODE, all memory words from location FIRST through location LAST whose contents masked by MASK equal the value of WORD. If no FIRST or LAST address is given that of the last bound-setting statement is taken. If no MODE is given that of the last mode-setting statement is taken. SRCHW is a bound-setting and a mode-setting statement. Note that if no MASK is given, it will be an "omitted argument" and represented by -Ø, so that the search will be for exact matching of the value of WORD.

SRCHA (ADDR, FIRST, LAST, "MODE")!

The behavior of SRCHA (A, F, L, M)! is identical to that of a control statement SRCHW (A, 77777, F, L, M)!.

2. Listing Memory Contents (LIST)

LIST ("MODE", FIRST, LAST)!

Will list all memory words from location FIRST through location LAST in formats MODE. If no FIRST or LAST address is given that of the last bound-setting statement is taken. If no MODE is given that of the last mode-setting statement is taken. LIST is a bound-setting and a mode-setting statement.

3. Changing Memory Contents (OPEN, TRAPS, FILL)

OPEN (ADDR, "MODE")!

Causes the val ue of expression ADDR to be typed in mode OCTAL and followed by a colon. Then the current contents of memory location ADDR are listed in the mode MODE. The memory cell at location ADDR is now "open". As long as a cell is left "open", further information concerning it may be requested by typing "@", "=" or "/" (see below), or information concerning it may be changed by typing:

NAMES VALUE CLOSE-CHARACTER

In the above, the inputs NAMES and/or VALUE may be omitted. If given, the input NAMES may be any number of undefined alphanumeric symbols, ten or fewer characters in length, each symbol followed by a colon (:). Each such symbol is thereby defined in the current Monitor symbol list as a name of the "open" location

ADDR. The expression VALUE, if given, is evaluated and its value replaces that contents of location ADDR when it is "closed". The CLOSE-CHARACTER, (Comma, Backspace or ←, Semicolon, or Carriage Return) enters the new value (if any) into the "open" cell (location ADDR) "closing" the cell, then does one of the following:

Close-Character	Action
comma	Opens the next cell in memory
backspace or ←	Opens the previous cell in memory
semicolon	Opens the cell addressed by the last open cell
carriage return	Does not open any further cells, returns to Monitor control statement input

While a cell is open, typing

@	lists the nearest symbolic expression defining the location of the open cell
=	lists the contents of the open cell in any modes other than MODE
/	lists the contents of the location addressed by last listed cell in the mode MODE
*	causes all input for expression VALUE to be dropped and ignored so that immediate closing would not alter the contents of the open cell
° or [	causes any comment (characters) up to the next ° or [to be ignored
#	closes the current open cell without altering its contents, and then opens the cell with address VALUE
$	stores VALUE in "open" cell

If no MODE is given, OPEN! uses that set by the last mode-setting statement.

TRAPS (FIRST, LAST)!

Causes memory locations FIRST through LAST to be filled with "trap" instructions. If no FIRST or LAST address is given, that of the last bound-setting statement is taken.

FILL (VALUE, FIRST, LAST)!

Causes the value of VALUE to be stored in memory locations FIRST through LAST. If no FIRST or LAST address is given, that of the last bound-setting statement is used.

C. Control of Memory Allocation (RESET, START, LOAD, MARK, RELEASE)

RESET!

Causes all loaded programs or program symbols, entries, external (global) parameters and memory allocations to be released. The state established by RESET is logically equivalent to that immediately after loading the Monitor into memory from magnetic tape.

START ("TITLE", TAPE)!

Causes a RESET operation and then loads the program named TITLE and all its required subprograms from tape unit TAPE. Loading defines all parameters and subprogram entries loaded, with their current memory locations. If no TAPE is given, the current "system tape" is used. If no TITLE is given, the first library program (type RELOC) on this tape is loaded.

LOAD ("NAME", TAPE)!

Causes any local program assembly symbols to be released, then loads the first subprogram with an entry NAME and all its required subprograms from tape unit TAPE and defines them. If no TAPE is given, the current "system tape" is used.

MARK!

Causes the Monitor to record the present limits of memory used for program storage and symbol storage, with a typeout of the current number of such "marks", which is also the identifying number of the one created by this operation.

RELEASE (NUMBER)!

Causes the Monitor to erase and make available for re-use all program and symbol storage loaded since the MARK operation identified by NUMBER. If no NUMBER is given, it is taken as the number of the most recently executed MARK operation not canceled by a previous RELEASE operation with a lower NUMBER. RELEASE operations may be performed automatically by operations which add programs to memory.

D. Symbol Table Control (DEFINE, DELETE)

DEFINE (NAME, VALUE,...)!

Enters, for each NAME-VALUE pair, the symbolic name NAME into the Monitor's current symbol table, assigning the value of expression VALUE as its

definition. If any NAME in a DEFINE statement is already defined in the Monitor's current symbol table, the Monitor will type out a message of "?".

DELETE ("NAME", "NAME",...)!

Removes each symbolic name NAME and its defining values from the Monitor's current symbol table.

E. Execution Control (DO, GOTO, SNAP, RETURN, STOP)

DO (INSTRUCTION, (REGISTER-LIST))!

Causes the values of the expressions in the REGISTER-LIST to be placed in the appropriate registers (as ordered in the PANEL statement) and then executes the octal value of the expression INSTRUCTION as a 30-bit machine-language instruction. Any registers for which expressions are omitted in the REGISTER-LIST will be restored to their saved values in the current PANEL. If execution does not transfer control to another program, the value left in the registers after execution are then saved, changing their previous saved-values in the PANEL. If execution causes a "skip" an extra carriage return will be typed out.

GOTO (ADDR, (REGISTER-LIST))!

If no ADDR is given, restores the registers from the current PANEL, as modified by (REGISTER-LIST), and then resumes execution of the last interrupted program at the point of interruption; otherwise the statement has the same behavior as DO (JUMP ADDR, (REGISTER-LIST))!

SNAP (FIRST, LAST, "MODE", (REGISTER-LIST))!

Restores registers and pivots to values saved in current PANEL, except where changed by new values in REGISTER-LIST, then starts execution at location FIRST. When execution reaches location LAST, or PULSE 1 (manual interrupt) switch is depressed, execution is suspended. The location at which execution was suspended is then typed in octal, followed by a listing of the resulting register values in formats MODE. The resulting register and pivot contents are then enstated as the current PANEL. If no FIRST or LAST address is given, that of the last bound-setting statement is taken. If no MODE is given, that of the last mode-setting statement is taken. SNAP is a bound-setting statement.

RETURN ((REGISTER-LIST))!

Restores registers and pivots to values saved in current PANEL, except where changed by new values in REGISTER-LIST, then resumes execution of last interrupted "foreground" program at point of interruption.

STOP!

Permanently suspends execution of any running "foreground" program.

F. Magnetic Tape I/O (DSET, READF, READN, WRITE, CHECKSUM, LOADA, DUMP, RESET)

In versions AMRM 2, 4 and 6 only, the control statement

DSET (DENSITY)!

Causes magnetic tape density to be set accordingly:

DENSITY	B.P.I.
Ø	2ØØ
1	556
2	8ØØ

READF (TAPE, FILE, FIRST RECORD, LAST RECORD)!

Causes the specified FILE, from FIRST RECORD through LAST RECORD, to be read from the specified TAPE into the memory area defined by the most recently executed bound-setting control statement. If no LAST RECORD is given, it is set equal to the last record in the specified logical file. If no FIRST RECORD is given, it is set equal to the first record in the specified logical file. If no TAPE is specified, it is set to the currently specified "system tape".

READN (TAPE, "TITLE", TYPE)!

Causes the logical file identified by TITLE and TYPE, to be read from the specified TAPE into the memory area defined by the most recently executed bound-setting control statement. If TYPE is not given, it is assumed to be BIN. If no TAPE is given, the current "system tape" unit will be selected. If no "TITLE" is given, the next file of the given "TYPE" on the specified unit will be read.

WRITE (FIRST, LAST, "TITLE", TAPE)!

Causes the contents of the memory defined by FIRST and LAST addresses to be written on the specified TAPE with the given TITLE, as a one-record file of type BIN. The TITLE and file number are typed out for the information of the operator. If no TAPE is given, the current "system tape" is selected. If no TITLE is given, the TITLE from the most recently executed DUMP, RESET, READ or WRITE operation will be used.

CHECKSUM!

Inhibits (or restores if already inhibited) the detection and notification of checksum errors during mag tape read operations, see page 19.

LOADA (FILE, TAPE)!

Loads type BIN records of file FILE from specified tape into memory at locations specified in header. WARNING: Will destroy programs, symbols or linkages currently allocated to that region of memory and not recover from doing so. If no TAPE is specified, the "system tape" is assumed.

DUMP ("TITLE", TAPE)!

Causes the memory contents exclusive of the Monitor itself to be written out as a file with the specified TITLE at the end of the specified TAPE, and the TITLE and file number to be typed out. If no TAPE is specified, the "system tape" is assumed. If no TITLE is given, one will be furnished by the Monitor.

RESET ("TITLE", TAPE)!

Causes the memory contents to be restored to the status existent just prior to execution of the preceding DUMP statement with same TITLE and TAPE parameters. If no TAPE number is given, the "system tape" is assumed. If no TITLE is given, the "system tape" is assumed. If no TITLE is given, the memory is restored to the state present immediately after loading the Monitor - i.e., all other programs and symbols are erased.

G. Version Setting and Interrogation (VERSION)

VERSION (VERS, "REV")!

Causes the setting of version to be written on subsequent magnetic tape headers to VERS (1 to 31_{10}, or Ø if omitted) and revision to REV (1 alphanumeric character). A typed "Ø" cannot be used as the version argument.

VERSION!

Causes the typing of the version, revision and date of the AMOS Monitor being used in the following format:

STANDARD AMRMn (REV. r, date)

where "n" is the version number, "r" is the revision level, and "date" is the assembly date.

USE OF MONITOR LIBRARY ROUTINES

Routines described in this section are released as separate programs and reside in the on-line library.

A. Internal Program Symbols Access

These statements permit a user to save the internal symbols defined during program assembly for subsequent retrieval and enstating in the Monitor's local symbol table to permit on-line debugging at the source language level.

1. Read Relocatable Symbols (READS)

READS ("TITLE", TAPE)!

Will cause the relocatable subroutine READS to be loaded (if not already in memory). Then the relocatable symbols (type RLSYM, output by DUMPS) for the program TITLE will be read in from the specified TAPE unit, adjusted and defined. If the file is not on the specified tape, or if the program "TITLE" has not been previously loaded, READS will type an appropriate message and return to the Monitor.

2. Dump Relocatable (ADEPT) Symbols

The DUMPS subroutine is called from the bootstrap loader immediately following the assembly of a program by ADEPT in the AMOS system. (See Section I of AMRMX Operating Instructions for the use of AMLDR.) It generates as output a file of type "RLSYM" (Relocatable Symbols) on the currently-assigned system tape, with the same TITLE as the latest ADEPT output. This file may be recognized and properly loaded and relocated by the READS subroutine. DUMPS overlays ADEPT on the 8K system (which has presumably been loaded at location 12Ø by a START ("ADEPT", TAPE)! statement).

B. Paper Tape I/O Control (READB, PUNCH, FEED, PTLST)

PTPIO is a relocatable library program for use in the AMOS operating system under control of the resident Monitor. Version 1 is used with AMRM1 and 2;

Version 2, with AMRM3, 4, and 5. It contains the routines **READB, PUNCH,** and FEED which provide facilities to read and punch bootstrap paper tape. An entry PTLST is also provided in Version 1 for listing a bootstrap tape.

READB ((FIRST, LAST))!

Causes the "bootstrap format" punched tape on the console punched tape reader to be read into (at most) the memory area specified by FIRST and LAST addresses, starting with the first non-null tape character encountered, and terminating when either the specified memory area has been filled, bit Ø of the IC register is set by the operator, or the checksum word (preceded by 2 bootstrap bits) has been read in. If no LAST is given, no operator-assigned upper memory limit is checked. If no arguments are given, the limits are set from the most recently executed bound-setting control statement. READB is a bound-setting statement. The actual limits of the memory loaded are typed out, together with the "check value" (if non-zero) obtained by combining all words read by the "checksum" operation. Under Version 2, a bell character is output at the beginning and end of the operation to allow the reader to be turned on/off.

PUNCH ((FIRST, LAST), CHECK)!

Causes the contents of the memory area defined by FIRST and LAST addresses to be output on "bootstrap format" punched tape, plus one extra bootstrap character, and one additional word to make the "checksum" of all output words equal to the CHECK value (octal). If no CHECK value is given, it is set to zero. Blank tape is punched before and after the output to provide a "leader" and "trailer". If FIRST or LAST is not given, it is set from the most recently executed bound-setting control statement. Under Version 2, a bell character is output at the beginning and end of the operation to allow the operator to turn the punch on/off.

FEED (LENGTH)!

Causes the punching of blank tape, where LENGTH specifies the number of characters. If IC[Ø] is pressed during the operation, FEED will return to the caller. If LENGTH -Ø or null, FEED will punch blank tape until IC[Ø] is depressed. Under Version 2, a bell character is output at the beginning and end of the operation to allow the operator to turn the punch on/off.

PTLST! (Version 1 only)

Causes the contents of a bootstrap paper tape to be interpreted and typed out in octal.

If the AMOS checksum correction value is encounterd, PTLST types out

. . . . .

VALUE

After the above, or when operation has been terminated by IC[Ø] PTLST types

SUM:SIGMA

where SIGMA is the arithmetic sum of all octal values read in.

MESSAGES TYPED BY THE MONITOR

A. While Typing Control Statement

* Improper argument in argument list, restart expression

? Improper statement formation, restart control statement

B. During Control Statement Execution

* Improper expression or close-character in OPEN subroutine, restart expression

? Wrong argument type in PANEL, SNAP, READN, DEFINE statements;
Symbol not found in DELETE;
Invalid BOUNDS; or
File not found in LOADA, RSMAG
Restart control statement in all cases

REQUIRED:	All of the following required programs were not contained on the specified unit. (Monitor then lists required subprograms.)
NAME (FIRST, LAST)	Subprogram NAME has been loaded into locations FIRST through LAST. Continues scanning if any more requests, or else returns to the Monitor.
NEW TAPE # =	Request for new tape unit number. The operator may type Ø, 1, 2, or 3 which is enstated as the new system tape number, and loading is resumed.
COMMON ERROR	The program requesting the largest amount of common storage was not encountered first during loading. Returns to the Monitor for input.
MEMORY TOO FULL	Not enough room in memory -- that is, the load limit and the symbol table have overlapped. The Monitor then types the name of the offending program, drops the current memory load, and returns to control statement input.
TOO MANY ENTRIES	The program being loaded has more entry points than can be accommodated in the loader's entry point buffer. Returns to control statement input.

TOO MANY OVERLAY SUBROUTINES IN MEMORY LOAD	Stack of entries to forget is too large. Drops the current memory load, and returns to the Monitor for control statement input.
LEVEL N	Nth overlay level in memory.
TOO MANY OVERLAY LEVELS	More than OLTL levels of overlay in memory. Drops one level and returns to the Monitor for control statement input.
N IS BIG PAGE	Page N of the text being compiled is too long to be contained in the page buffer, and has overflowed into the Monitor. Therefore, the Monitor must be reloaded before it is used again.
END OF TEXT	No more source text on scratch pad for processor (compiler, assembler). Correct termination after end of last program; unrecognizable end of program error in all other cases.
RECORD TOO LONG	Record which was specified to be read into memory BOUNDS was too long. Returns to control statement input.
WRITE LOCKOUT	Tape unit on which a record is to be written does not permit writing. Program waits for operator to enable write and depress IC bit Ø, and then tries to write again.
FILE/NO. N	File number of file just written.
CKS?	When reading record, indicates checksum error. May be avoided by typing CHECKSUM! (Note that this avoids the error output and not the error itself.)
STANDARD AMRMn (REV. r, date)	AMRM will identify itself after the operator types the control statement: VERSION!

AMOS CHARACTER CODES

A. Internal Characters

The Monitor implements single case AMOS internal characters:

Code	OPC Character	TTY Character	LPR Character
ØØ	° (null)	[	[
Ø1	%	%	%
Ø2	¢	]	]
Ø3	!	!	!
Ø4	&	&	&
Ø5	*	*	*
Ø6	:	:	:
Ø7	_	\	\
1Ø	+	+	+
11	tab	tab (3 spaces)	<
12	?	?	?
13	"	"	"
14	'	'	'
15	carriage return	return - L. F.	>
16	(	(	(
17	)	)	)
2Ø	Ø	Ø	Ø
21	1	1	1
22	2	2	2
23	3	3	3
24	4	4	4
25	5	5	5
26	6	6	6
27	7	7	7
3Ø	8	8	8
31	9	9	9
32	;	;	;
33	=	=	=

Code	OPC Character	TTY Character	LPR Character
34	,	,	,
35	-	-	-
36	.	.	.
37	/	/	/
4Ø	space	space	blank
41	A	A	A
42	B	B	B
43	C	C	C
44	D	D	D
45	E	E	E
46	F	F	F
47	G	G	G
5Ø	H	H	H
51	I	I	I
52	J	J	J
53	K	K	K
54	L	L	L
55	M	M	M
56	N	N	N
57	O	O	O
6Ø	P	P	P
61	Q	Q	Q
62	R	R	R
63	S	S	S
64	T	T	T
65	U	U	U
66	V	V	V
67	W	W	W
7Ø	X	X	X
71	Y	Y	Y
72	Z	Z	Z
73	$	$	$
74	#	#	#
75	@	@	@
76	±	↑	^
77	backspace	←	_

B. Input Characters

The following classes of input characters are recognized by the Monitor:

OCTAL DIGITS

consists of characters Ø through 7.

DECIMAL DIGITS

consists of characters Ø through 9.

ALPHANUMERIC CHARACTERS

consists of alphabetic characters A-Z, plus digits Ø through 9, plus period ".".

STRING CHARACTERS

consists of all AMOS internal characters except %, ? and ", which may be internally represented by the input strings %%, %? and %", respectively.

MONITOR CONTROL STATEMENT SUMMARY

OPERATION (ARGUMENT1,ARGUMENT2,...,ARGUMENT N)!

Control Parameter Specification

SYST (TAPE)!
MODE ("MODE")!
BOUND (FIRST, LAST)!
TAPEC! Versions 1 and 2 only
TYPEC! Versions 1 and 2 only
PANEL ((REGISTER-LIST)) !
VERSION (VERS, "REV") !
DSET (DENS) !

Memory Search, List and Change

SRCHW (WORD, MASK, FIRST, LAST, "MODE")!
LIST ("MODE", FIRST, LAST)!
SRCHA (ADDRESS, FIRST, LAST)!
FILL (VALUE, FIRST, LAST)!
TRAPS (FIRST, LAST)!
OPEN (ADDRESS, "MODE")!

Resulting typeout:	Input options:
LOCATION: CONTENTS	NAMES VALUE DCH CCH
where:	means:
CCH	enter any VALUE, close location, and:
,	open next
backspace or ←	open previous
;	open location addressed by contents
carriage return	return to Monitor
DCH	
@	list symbolic location
=	list contents in other MODES
/	list cell addressed by contents
*	ignore previous VALUE
° or [	ignore following comment
#	close location and open cell at VALUE
$	enter VALUE without closing

VALUE = An expression
NAMES = Undefined symbols followed by ":" to name "LOCATION"

Execution Control

DO (INSTRUCTION, (REGISTER-LIST))!
GOTO (ADDRESS, (REGISTER-LIST))!
SNAP (FIRST, LAST, "MODE", (REGISTER-LIST))!
RETURN (REGISTER-LIST)!
STOP!

Memory Allocation

START ("TITLE", TAPE)!
LOAD ("ENTRY", TAPE)!
LOADA (FILE, TAPE)!
DUMP ("TITLE", TAPE)!
RESET ("TITLE", TAPE)!
MARK!
RELEASE (N)!

Magnetic Tape I/O

READF (TAPE, FILE, FIRST RECORD, LAST RECORD)!
READN (TAPE, "TITLE", TYPE)!
WRITE (FIRST, LAST, "TITLE", TAPE)!
CHECKSUM!

Monitor Version Inquiry

VERSION!

Paper Tape I/O

READB ((FIRST, LAST))!
PUNCH ((FIRST, LAST), CHECK)!
FEED (LENGTH)!
PTLST!

Symbol Table Control

DEFINE (NAME, VALUE, NAME, VALUE,...)!
DELETE ("NAME", "NAME",...)!
DUMPS!
READS ("TITLE", TAPE)!

Arguments

TAPE = Specified Tape Unit Number Ø, 1, 2, or 3
FILE = Specified File Number
RECORD = Specified Record Number
"TITLE" = Title of a Specified File

"ENTRY" = Name of an Entry Point
NAME = A symbolic address enclosed in quotes, " ", in the DELETE operator and not enclosed in quotes in the DEFINE operator.
"MODE" = Typeout Mode: "O", "A", "S", "F",...
Register LIST = Values for AR, BR, IC, TPT, OVF, RTI in version 1, 2 or AR, BR, IC, TTYBT, TTYCH, OV, RTI in versions 3, 4, 5
FIRST, LAST = Lower and Upper Bounds in a Monitor Operation
N = A Level Number
CHECK = Value for checksum
LENGTH = Feed-hole count
VERS = Program version number (1 through 31_{10})
"REV" = A single program revision character (i. e., "A")
DENS = Density selection code (0 = 200BPI, 1 = 556BPI, 2 = 800BPI)

SYNTACTIC STRUCTURE OF MONITOR STATEMENTS

The following is an extended BNF description of the control statement syntax accepted by AMRMX.

< control statement >:: =< term > ! | < term >< argument list > !

< argument list > :: = (< argument > { , < argument > }$_0$)

< argument > :: =< expression > | | (< argument list >) | "{ < character > }$_6$ "

< expression > :: =< sign >< term > | < expression > {< rotation > }$_1$ |
< expression > { < op > }$_1$ < term >

< term > :: = { < octal digit > }$_1^{10}$ | { < decimal digit > }$_1^9$. | {< alphanumeric> }$_1^{10}$ |
. |] | "{ < character > }$_1^5$ "

< rotation > :: = ! B | ! K | ! H

< op > :: =< sign > | ↑ | ' | &

< sign > :: = + | - | space | tab

< octal > :: = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7

< decimal digit > :: =< octal digit > | 8 | 9

< alphanumeric > :: =< decimal digit > | A | B | C ... | X | Y | Z | .

< character > :: =< alphanumeric > | < op > | , | / | $ | # | @ | [| = | \ | (|) | ← | "|
carriage return |] | ! | : | ; | * | %% | %?

Explanation of notation

< x >	"a member of the syntactic class of x"
:: =	"is defined to be"
|	"or"
$\{x\}_m^n$	"from m to n" instances of x (if n is not specified, can be arbitrarily large)
...	"and so on"

NOTE: In non-TTY versions the characters: [ ] \ ↑ ← are replaced by the characters, ° ¢ _ ± backspace.

adage

AMRMX

AGT DISK MONITOR

Programmer's Reference Manual

Revision A

April 1969

TABLE OF CONTENTS

CONTENTS (Cont.)

CONTENTS (Cont.)

INTRODUCTION

The AMOS Resident Monitor (AMRMX) is that portion of AMOS which accepts and interprets AMOS control statements, loads programs requested (if necessary), and executes all programs.

The Monitor is called "resident" because it is kept in core memory at all times during the use of AMOS, whereas all other programs are loaded into memory only as they are needed to execute "control statements." Portions of the Monitor are called into memory from disk when needed and occupy space within the Resident section.

The Monitor continuously monitors the operator's console, and responds to control statements entered therefrom by means of the keyboard. This monitoring function may be performed as a "background" task to another "foreground" program being executed on a time-shared basis. In other words, while another program is being executed under control of AMOS, the Monitor will accept and execute control statements not in conflict with the real-time requirements of the running "foreground" program.

The function of the Monitor is to accept, interpret, and execute AMOS control statements furnished by the system operator. Each control statement causes the execution of a previously-written program stored in memory, with the parameter values indicated by the control statement. The general form of an AMOS control statement is:

OPERATION (PARAMETER 1, PARAMETER 2, ... PARAMETER N)!

where each parameter may be an octal/symbolic expression. Unneeded parameters may be omitted from a control statement, provided that all of the parameter-separating commas preceding the last parameter used are given.

This document gives operating information for the following versions of AMRMX, having the hardware configuration as specified:

Version 13- AGT/10, M10-P1, 2, or 3, DMS2-P1

Version 14- AGT/30, DMS2-P1

Version 15- AGT/50, DMS2-P1

Version 16- AGT/10, M10-P1, 2, or 3, DMS2-P1, LCG1-P1
or AGT/30, DMS2-P1, LCG1-P1
or AGT/50, DMS2-P1, LCG1-P1

CONTROL INPUTS

CONTROL STATEMENTS

For every control input statement

TERM!

or

TERM (ARG1, ... ARGN)!

the Monitor builds a standard calling sequence in available memory as follows:

Location	Contents	Explanation
LL	JPSR $TERM	Call to Subroutine
LL+1	ARG1	Argument-List Values
	:	(Optional)
LL+NN	ARGN	
LL+NN+1	Ø	End of Argument-List Flag
LL+NN+2	JUMP MON9	Return to Monitor

After the statement has been input and the program with entry point TERM loaded (if necessary), the Monitor transfers control to this calling sequence.

An input or output of the character ? will re-start the control statement in which it occurs.

ARGUMENTS

For those programs requiring arguments, the Monitor will accept a list of inputs (enclosed in parentheses) and produce the corresponding entry in the generated calling sequence.

Each argument is one of the following:

15-bit Address
3Ø-bit Value
Null (omitted) Argument
Sub-Argument List
Undefined Symbol
Multi-Word String

Subroutines to be called on-line via Monitor control statements may be written to interpret the Monitor's encoding of the above options, permitting the user full flexibility of assembly-like input arguments.

It is the responsibility of the user to ensure that the number and type of argument in each control statement corresponds to the subroutine being called. Within this constraint one may type command statements with 3Ø-bit expressions, quoted character strings, omitted optional arguments, or parenthetically nested argument structures as arguments. When arguments are omitted, all separating commas up to the last non-omitted argument must be given (final commas in a list may be omitted), since arguments are identified by their position in the generated calling sequence.

An input or output of \ (back slash) will re-start the current argument.

Expressions

Each 15-bit address or 3Ø-bit value is entered as an expression. Expressions consist of terms separated by arithmetic operators. They are evaluated on a term-by-term basis as they are input to the Monitor. The resulting 3Ø-bit value is placed in the generated calling sequence.

Terms -- A term is one of the following:

- Octal Number - From 1 to $1Ø_{10}$ octal digits. Its value is the $3Ø_{10}$-bit right-justified octal integer it represents.

- Decimal Number - From 1 to 9_{10} decimal digits followed by a period ".". Its value is the $3Ø_{10}$-bit right-justified decimal integer it represents.

- Defined Symbol - From 1 to $1Ø_{10}$ alphanumeric characters, defined in the Monitor's symbol table. Its value is its $3Ø_{10}$-bit definition in the symbol table.

- One-Word String - From 1 to 5 characters, enclosed in quotation marks ("). The value of the term is its 6-bit internal character code representation, left-justified and filled with nulls ($\emptyset\emptyset_8$), if less than 5 characters in length.

- Specific Values - During memory interrogation and change certain characters may be used in expressions as terms having specific values:

  . Has the value of the current OPEN location
  ] Has the value of the contents of the current OPEN location

Logical and Arithmetic Operators -- The value of an expression is obtained by combining the values of its terms according to its logical and arithmetic operators. (The initial expression or preceding expression for the first term is assumed to be $+\emptyset$.) These operators are as follows:

Operator	Action
+, tab, space	Add the value of the following term to the value of the preceding expression.
-	Subtract the value of the following term from the value of the preceding expression.
↑	Inclusive OR the value of the following term with the value of the preceding expression.
&	AND the value of the following term with the value of the preceding expression.
'	EXCLUSIVE OR the value of the following term with the value of the preceding expression.
!B	Bit (1-bit) left rotate the value of the preceding expression.
!K	Character (6-bit) left rotate the value of the preceding expression.
!H	Half-word (15-bit) left rotate the value of the preceding expression.
*	Multiply the preceding expression by the following term.

Operator	Action
/	Divide the preceding expression by the following term.

Null Arguments

A null argument is one that is omitted. It is represented in the generated calling sequence by the value -Ø.

Sub-Argument List

An argument-list nested in parentheses may be used as an argument. It is represented in the generated calling sequence by a 3Ø-bit word whose upper 15-bits contain the number of arguments in the sub-argument list. The lower 15-bits contain a pointer to the first sub-argument (i.e., its address, decremented by 1).

Undefined Symbol

An undefined symbol consists of a group of from 1 to $1Ø_{10}$ alphanumeric characters, not defined in the Monitor's current symbol table. It is represented in the generated calling sequence by a 3Ø-bit word whose upper 15 bits are -Ø. The lower 15 bits are a pointer to the 2-word symbol string (i.e., its starting address).

Multi-Word String

A multi-word string consists of 6 or more characters enclosed in quotation marks ("). It is represented in the generated calling sequence by a 3Ø-bit word whose upper 15 bits contain the negative of the number of words in the string. The lower 15 bits are a pointer to the symbol string (i.e., its starting address).

USE OF BUILT-IN MONITOR STATEMENTS

SYSTEM PARAMETER SPECIFICATION

Console Message Unit Assignment (TYPE, DISPLAY)

TYPE!

causes all normal character string output from Monitor statements LIST, OPEN, SRCHW, SRCHA, LISTA, LISTD, LISTV, PANEL, VERSION and MEMAP to be typed on the console TTY unit.

DISPLAY!

causes all normal character string output from Monitor statements LIST, OPEN, SRCHW, SRCHA, LISTA, LISTD, LISTV, PANEL, VERSION and MEMAP to be displayed on the CRT.

System Volume for AMRMX Disk I/O (SYST)

SYST (VOL) ! (where VOL is of the form pvv_8)

assigns volume vv (1 to 40_8) on disk pack p (Ø to 7) as the current system volume.

Input Volume and File for Processor Text Input (ASSIGNIN)

ASSIGNIN (FILE, VOL) !

assigns file FILE on volume vv of pack p as the current input text file for ADEPT or AFORT. If the volume is Ø, text will be read from the "Scratch Pad" area of pack p. When AMRMX is first loaded, assignment is made to the "Scratch Pad" area of Pack Ø.

Control Register-List Change and Interrogation (PANEL)

PANEL ((Register-List))!

replaces the saved register-list with new values, any of which may be omitted. (If a new value is omitted, the previous value is not changed.) The Register-List consists of the following:

AR, BR, IC, OVF

where AR, BR, IC are the saved contents of the AR, BR, IC registers
OVF is the AR overflow pivot, 77771

PANEL!

causes the current saved values of the Register-List, the saved AMRMX entry address LC, the current highest loaded location (LD), and the current lowest local symbol table bound (PTR), to be typed out (or displayed if in DISPLAY mode).

I/O and Search Memory Bounds (BOUND)

BOUND (FIRST, LAST)!

sets FIRST and LAST addresses for processing by other control statements. If FIRST is not specified, each time it is referenced it will be set to the current lower limit of available memory. If LAST is not specified, each time it is referenced it will be set to the current upper limit of available memory. The FIRST and LAST addresses can also be set by the other bound-setting statements which are LIST, SRCHW, SRCHA, WRITE, READB, and PUNCH.

Output Format (MODE)

MODE ("TYPE")!

sets the format(s) desired for subsequent typeouts of internal machine-word images. The available (built-in) formats and their type-codes are:

Type-Code	Format
O (Octal):	The 5-digit octal value contained in the "upper half word" followed by a space followed by the 5-digit value contained in the "lower half-word."
S(Symbolic):	Symbolic name of this location, followed by colon.
	A symbolic expression whose value is contained in the "upper half-word" followed by three spaces.
A (Characters):	Five characters whose standard AMOS internal bi-octal codes are in the word.
F (Full):	All of the above.

Any combinations of O, S, and A are also recognized. Whenever mode "S" is selected, locations are typed out one per line; otherwise, the resulting type-out is given in several columns across the page.

Disk Volume Directory Initialization (RELOAD, HOME)

RELOAD (PACK)!

causes the Monitor to reload the volume directory sector from disk pack PACK into memory. This command must be given after changing a disk pack so that the proper volume directory for the new pack will be used in any subsequent disk input/output. This command should not be used after changing pack Ø. The Monitor must be reloaded from a changed pack Ø by the system loader, AMLDX, Version 5.

The disk should be positioned at cylinder Ø (home cylinder) before stopping the disk drive. The following statement:

HOME (UNIT)!

will cause the selected unit to position at cylinder Ø.

Disk Program Initialization (CLEAR)

CLEAR!

causes all program status about disk position, open (selected) file names, and busy indications to be reset as the DSKIO section of the Monitor is used to read in the segments. CLEAR cannot be loaded if the DSKIO busy indicator is set. In order to clear this indicator, the operator should reset the BR register to zero and execute the instruction in location 77774_8.

MEMORY INTERROGATION AND CHANGE

Memory Search (SRCHW, SRCHA)

SRCHW (WORD, MASK, FIRST, LAST, "MODE")!

will list (in formats MODE) all memory words, from location FIRST through location LAST, whose contents masked by MASK equal the value of WORD. If no FIRST or LAST address is given, that of the last bound-setting statement is taken. If no MODE is given, that of the last mode-setting statement is taken. SRCHW is a bound-setting and a mode-setting statement. Note that if no MASK is given, it will be an "omitted argument" and represented by -Ø, so that the search will be for exact matching of the value of WORD. SRCHW may be prematurely terminated by depression of IC[0] or FNS1.

SRCHA (ADDR, FIRST, LAST, "MODE")!

The behavior of SRCHA (A, F, L, M)! is identical to that of a control statement SRCHW (A, 77777, F, L, M)!.

Listing Memory Contents (LIST)

LIST ("MODE", FIRST, LAST)!

will list all memory words from location FIRST through location LAST in formats MODE. If no FIRST or LAST address is given that of the last bound-setting statement is taken. If no MODE is given that of the last mode-setting statement is taken. LIST is bound-setting and mode-setting statement. LIST may be prematurely terminated by depression of IC[0] or FNS1.

Changing Memory Contents (OPEN, TRAPS, FILL)

OPEN (ADDR, "MODE")!

causes the value of expression ADDR to be typed in mode OCTAL and followed by a colon. Then the current contents of memory location ADDR are listed in the mode MODE. The memory cell at location ADDR is now "open." As long as a cell is left "open," further information concerning it may be requested by typing a "@", "=", or "/" (see below), or information concerning it may be changed by typing:

NAMES VALUE CLOSE-CHARACTER

In the above statement, the inputs NAMES and/or VALUE may be omitted. If given, the input NAMES may be any number of undefined alphanumeric symbols, ten or fewer characters in length, each symbol followed by a colon(:). Each symbol is defined in the current Monitor local symbol list as a name of the "open" location ADDR. The expression VALUE (if given) is evaluated, and its value replaces the contents of location ADDR when it is "closed." The CLOSE-CHARACTER (Comma, ← , Semicolon, or Carriage Return) enters the new value (if any) into the "open" cell (location ADDR) "closing" the cell, then does one of the following:

Close-Character	Action
comma	Opens the next cell in memory.
←	Opens the previous cell in memory.
semicolon	Opens the cell addressed by the last open cell.
carriage return	Does not open any further cells, returns to Monitor control statement input.

While a cell is open, typing

@	lists the nearest symbolic expression defining the location of the open cell
=	lists the contents of the open cell in any modes other than MODE
/	lists the contents of the location addressed by last listed cell in the mode MODE

\	causes all input for expression value to be dropped and ignored so that immediate closing would not alter the contents of the open cell
[	causes any comment (characters) up to the next [to be ignored
#	closes the current open cell without altering its contents, and then opens the cell with address VALUE
$	stores VALUE in "open" cell and leaves it still open

If no MODE is given, OPEN uses that set by the last mode-setting statement.

TRAPS (FIRST, LAST)!

causes memory locations FIRST through LAST to be filled with "trap" instructions. If no FIRST or LAST address is given, that of the last bound-setting statement is taken.

FILL (WORD, FIRST, LAST)!

causes the value of WORD to be stored in memory locations FIRST through LAST. If no FIRST or LAST address is given that of the last bound-setting statement is used.

CONTROL OF MEMORY ALLOCATION (RESET, START, LOAD, MARK, RELEASE)

RESET!

causes all loaded programs, local symbols, program entries, external references, and memory allocations to be released. The state established by RESET is logically equivalent to that immediately after loading the Monitor into memory from disk.

START ("TITLE," VOL)! [form 1]

START (FILE, VOL)! [form 2]

causes a RESET operation and then loads the program named TITLE (form 1) or having file number FILE (form 2) from disk volume VOL. Loading defines all subprogram entries loaded, with their current memory locations. A memory map file is created. If no VOL is given, the current "system volume" is used. If no TITLE is given (form 1), the first library program (type RELOC) on the volume is loaded.

```
LOAD ("NAME", VOL)!      [form 1]
LOAD (FILE, VOL)!        [form 2]
```

loads the first subprogram with an entry NAME (form 1) or with file number FILE (form 2) and all its required subprograms from the disk volume VOL. The current memory map is updated. If no VOL is given, the current "system volume" is used. LOAD is a "system volume" setting statement.

```
MARK!
```

causes the Monitor to record the present limits of memory and disk used for program and symbol storage with a typeout of the current number of such "marks." That number is also the identifying number of the one created by this operation.

```
RELEASE (NUMBER)!
```

causes the Monitor to erase and make available for re-use all programs and symbol storage loaded since the MARK operation identified by NUMBER. If no NUMBER is given, it is taken as the number of the most recently executed MARK operation not cancelled by a previous RELEASE operation with a lower NUMBER. RELEASE operations may be performed automatically by operations which add programs to memory via OVRLY calls.

SYMBOL TABLE CONTROL (DEFINE, DELETE, NTRY, DNTRY, READS, DUMPS)

```
DEFINE (NAME, VALUE, ...)!
```

enters the symbolic name NAME into the Monitor's current local symbol table for each NAME-VALUE Pair, assigning the value of expression VALUE as its definition. If any NAME in a DEFINE statement is already defined in the Monitor's current local symbol table, the Monitor will typeout a message of "?".

DELETE ("NAME", "NAME" ....)!

removes each symbolic name NAME and its defining value from the Monitor's current local symbol table.

NTRY (NAME, LOC, ...)!

enters the symbolic name NAME into the Monitor's current external symbol table for each NAME-LOC pair, assigning the address value of expression LOC as its definition. If any NAME in an NTRY statement is previously defined, the Monitor will typeout a message of "?".

DNTRY ("NAME", "NAME", ...)!

removes each symbolic NAME and its defining value from the current external symbol table.

DUMPS!

generates as output a file of type "RLSYM" (Relocatable Symbols) on the currently assigned "system volume," with the same TITLE as the last ADEPT output. This file may be recognized and properly loaded and relocated by the READS statement. DUMPS may be called only immediately following an ADEPT assembly.

READS ("TITLE", VOL)!

reads and defines in the Monitor's current local symbol table the file of the title TITLE and type "RLSYM" from the disk volume VOL. If no VOL is given, the current "system volume" will be used. If the file is not contained within the specified volume, or if the program "TITLE" has not been loaded, or if there is insufficient room in the local symbol table to make the symbol definitions, an appropriate error message will be typed and control will be returned.

EXECUTION CONTROL (DO, GOTO, SNAP, RETURN, STOP)

DO (INSTRUCTION, (REGISTER-LIST))!

causes the values of the expressions in the REGISTER-LIST to be placed in the appropriate registers (as ordered in the PANEL statement) and then executes the octal value of the expression INSTRUCTION as a 3Ø-bit machine-language in-

struction. Any registers for which expressions are omitted in the REGISTER-LIST will be restored to their saved values in the current PANEL. If execution does not transfer control to another program, the values left in the registers after execution are then saved, changing their previous saved-values in the PANEL. If execution causes a "skip," an extra carriage return will be typed out.

GOTO (ADDR, (REGISTER-LIST))!

restores the registers from the current PANEL as modified by (REGISTER-LIST) if no ADDR is given, and then resumes execution of the last interrupted program at the point of interruption; otherwise the statement has the same behavior as DO (JUMP ADDR, (REGISTER-LIST))!

SNAP (FIRST, LAST, "MODE", (REGISTER-LIST))!

restores registers and pivots to values saved in current PANEL except where changed by new values in REGISTER-LIST, then starts execution at location FIRST. When execution reaches location LAST, or the PULSE 1 (manual interrupt) switch is depressed, execution is suspended. The location at which execution was suspended is then typed in octal, followed by a listing of the resulting register values in formats MODE. The resulting register and pivot contents are then enstated as the current PANEL. If no FIRST or LAST address is given, that of the last bound-setting statement is taken. If no MODE is given, that of the last mode-setting statement is taken. SNAP is a bound-setting statement.

RETURN ((REGISTER-LIST))!

restores registers and pivots to values saved in current PANEL except where changed by new values in REGISTER-LIST, then resumes execution of last interrupted "foreground" program at point of interruption.

STOP!

permanently suspends execution of any running "foreground" program. It also stops the Monitor's console message character string display routine.

VERSION SETTING AND INTERROGATION (VERSION)

VERSION (VERS, "REV")!

causes the setting of version to be written on subsequent magnetic tape headers to VERS (1 to 31_{10} or Ø if omitted) and revision to REV (1 alphanumeric character). A typed "Ø" cannot be used as the version argument.

VERSION!

causes the typing of the version, revision, and date of the AMOS Monitor being used in the following format:

STANDARD AMRMX (VERS. n, REV. r, date)

where "n" is the version number, "r" is the revision level, and "date" is the source text date.

PAPER TAPE I/O CONTROL (READB, PUNCH, FEED)

READB ((FIRST, LAST))!

causes the "bootstrap format" punched tape on the console punched tape reader to be read into (at most) the memory area specified by FIRST and LAST addresses, starting with the first non-null tape character encountered, and terminating when either the specified memory area has been filled, bit Ø of the IC register or FNS1 is set by the operator, or the checksum word (preceded by 2 bootstrap bits) has been read in. If no LAST is given, no operator-assigned upper memory limit is checked. If no arguments are given, the limits are set from the most recently executed bound-setting control statement. The actual limits of the memory loaded are typed out together with the "check value" (if not-zero) obtained, by combining all words read by the "checksum" operation. A bell character is output at the beginning and end of the operation to allow the reader to be turned on/off. READB is a bound-setting statement.

PUNCH ((FIRST, LAST), CHECK)!

causes the contents of the memory area defined by FIRST and LAST addresses to be output on "bootstrap format" punched tape, one extra bootstrap character, and one additional word to make the "checksum" of all output words equal to the CHECK value (octal). If no CHECK value is given, it is set to zero. Blank tape is punched before and after the output to provide a "leader" and "trailer." If FIRST or LAST is not given, it is set from the most recently executed bound-setting control statement. A bell character is output at the beginning and end of

the operation to allow the operator to turn the punch on/off. PUNCH is a bound-setting statement.

FEED (LENGTH)!

causes the punching of blank tape where LENGTH specifies the number of characters. If IC[Ø] or FNS1 is pressed during the operation, FEED will return to the caller. If LENGTH -Ø or null, FEED will punch blank tape until IC[Ø] or FNS1 is depressed. A bell character is output at the beginning and end of the operation to allow the operator to turn the punch on/off.

DISK STORAGE ALLOCATION (INITIALIZE, CREATE, CHANGE, PURGE, REMOVE)

All the monitor statements in the following section access on-line disk storage for the creation and manipulation of user and system files. For a detailed description of the standard formats and conventions, refer to the FILE I/O PRM document.

INITIALIZE (PACK, "IDENTIFICATION STRING")!

causes the initialization of disk pack PACK. All files stored on the designated pack are deleted, new address information is written on all sectors, and empty volume directory and file directories are created. The IDENTIFICATION STRING, a character string of 19 or fewer characters in length, is written on the pack to identify it. This string will be output when listing an index of the pack in the Monitor statements LISTA and LISTD. The current date is also written on the disk pack for identification purposes. This operation is illegal on disk pack Ø.

CREATE (PACK, VOLUME, #CYLS)!

causes volume VOLUME on disk pack PACK to be created. The number of disk cylinders to be allocated to the volume is specified by #CYLS. If the volume is already defined or if there is insufficient free disk storage for the requested number of cylinders, an appropriate error message will be typed and control returned without completing the operation.

CHANGE (PACK, VOLUME, ± #CYLS)!

causes the number of cylinders assigned to volume VOLUME on disk pack PACK to be increased by #CYLS, or decreased if #CYLS negative. This operation may cause moving of cylinder data on the disk. If the volume is not defined, if there is insufficient free disk storage for the request, or if there is not enough unused disk storage within the volume to reduce the number of cylinders as requested, an error message will be typed and control returned without completing the operation.

PURGE (PACK, VOLUME)!

causes the entire contents of volume VOLUME on disk pack PACK to be deleted. The volume directory for the specified volume is reset, and a CREATE operation must be performed to use this volume again. This operation may cause moving of cylinder data on the disk. If the volume is not defined, an error message will be typed and control returned.

REMOVE (PACK, VOLUME, FILE)!

or

REMOVE (PACK, VOLUME, (FILE1, FILE2 ....))!

deletes the file or files specified from volume VOLUME on disk pack PACK. If a file specified to be deleted is not found in the selected volume, or if the file list is not in order, an error message is typed and the operation is terminated with the previously designated files deleted. If the volume is not defined, an error message is typed, and control is returned. Re-ordering of sector data within the volume may be done by this operation.

DISK INPUT/OUTPUT (DUMP, RESET, LOADA, WRITE, COPY, COPYAMRMX)

DUMP ("TITLE", VOL)"

causes the writing of all currently used program storage areas of core memory, the current local and external symbol tables, the overlay table, and selected AMRMX storage allocation parameters onto disk volume vv of pack p as a new file titled TITLE of type "DUMP." If VOL is omitted, output will occur on the currently selected "system volume." This file may be read from disk by a subsequent RESET operation with the effect of restoring all programs, symbol definitions, and memory allocations as were in effect when the DUMP operation was executed.

RESET ("TITLE", VOL)!

causes the reading of the file titled TITLE of type "DUMP" from volume vv on disk pack p and restores all programs, symbol definitions, and memory allocation in effect at the time of the DUMP operation which created the file. If VOL is omitted, the current "system volume" is used.

LOADA (FILE, VOL)!

causes the reading of the file with file number FILE and type "BIN" from volume vv on disk pack p into the locations specified in the record control words in the file.

CAUTION

This operation may overwrite programs, symbols, etc., if they lie in areas changed by the specified absolute (BIN) file.

WRITE (FIRST, LAST, "TITLE", VOL)!

creates a file on volume vv of disk pack p of type "BIN" and titled TITLE containing the contents of core memory from FIRST through LAST. Record control words are inserted into the file so that the file created may be read back into the same area of core memory by a LOADA statement. If FIRST and LAST are omitted, the area to be written is specified by the last bound-setting statement. If VOL is omitted, the current "system volume" is used. WRITE is a bound-setting statement.

COPY (FIRST, LAST, IVOL, OVOL)!

or

COPY ((F1, L1, I1, OVOL), (F2, L2, I2), (F3, L3, I3) ....)!

causes the copying of files FIRST through LAST from volume vv of pack p (IVOL) to volume vv of pack p (OVOL). If LAST is omitted, it is set equal to FIRST. In the multiple argument group form, the first argument group must specify the four arguments as in the non-multiple argument form. In subsequent argument groups, OVOL (if present) must be equal to the initial OVOL. If IVOL is omitted on any subsequent argument group, the input volume and pack is specified by the last given IVOL argument. If LAST is omitted in any argument group, it is set equal to FIRST.

COPYAMRMX (PACK)!

causes the copying of the binary monitor, cylinders 2 through 14_8, from pack Ø to the specified output pack. This operation may only be performed on an initialized disk pack which contains no "CREATE"d volume assignments.

DISK VOLUME/FILE AND MEMORY MAP LISTING (LISTD, LISTV, LISTA, MEMAP)

LISTD (PACK)!

causes the typing (displaying if in DISPLAY mode) of the volume directory of disk pack PACK. Included in the output is the pack's IDENTIFICATION STRING, creation date, number of free (unassigned) cylinders, and number of cylinders used in the "scratch pad" area. For each defined volume, the number of files used, number of cylinders assigned, the number of free (unused) sectors, and the disk storage area assignments of the volume are output.

LISTV (PACK, VOLUME, VOLUME, VOLUME ....)!

causes the typing (displaying if in DISPLAY mode) of the volume information as specified in the LISTD statement and a file index for each file in the volume for each volume specified in the statement. The file index gives the file number, title, type, version, revision, date, length, cylinder/track/sector origin of the file (relative to the first cylinder in the volume) for each file and whether or not the file is in "record" format.

LISTA (PACK)!

causes the typing (displaying if in DISPLAY mode) of the volume directory of disk pack PACK along with a file index for each file in each defined volume.

MEMAP!

causes the typing (displaying if in DISPLAY mode) of the current memory map. The output consists of each loaded program file title along with the limits of core memory assigned to that program.

TEXT DISPLAY OUTPUT FORMAT

When the Monitor is displaying the output from the statements LIST, SRCHW, SRCHA, LISTV, LISTD, and LISTA on the CRT, the entire output may not fit on the screen at one time. In this case, the Monitor will display the current "screen - load" ("page") until the operator depresses any FNS button except FNS1, at which time the next "page" is generated on the CRT. This operation continues until the last "page" is generated at which time control is returned to accept subsequent Monitor statements while the last "page" is being displayed. The depression of FNS 1 will cause the operation to be terminated and control returned to monitor statement input. As the text output from the statements LISTV, LISTD, and LISTA may be buffered in a core area of the Monitor used by the statement input routine, the typing of the next statement to the Monitor may cause the latter portion of the text being displayed to be blanked out at the time of the first symbol search operation performed by the statement input routine.

AMOS CHARACTER CODES

INTERNAL CHARACTERS

The Monitor implements single case AMOS internal characters:

$Code_8$	USASCII $Equivalent_8$	TTY Character	LPR/LCG Character
ØØ	133	[	[
Ø1	045	%	%
Ø2	135	]	]
Ø3	041	!	!
Ø4	046	&	&
Ø5	052	*	*
Ø6	072	:	:
Ø7	134	\	\
1Ø	053	+	+
11	074	tab (3 spaces)	<
12	077	?	?
13	042	"	"
14	047	'	'
15	076	return - L.F.	>
16	050	(	(
17	051	)	)
2Ø	060	Ø	Ø
21	061	1	1
22	062	2	2
23	063	3	3
24	064	4	4
25	065	5	5
26	066	6	6
27	067	7	7
3Ø	070	8	8
31	071	9	9
32	073	;	;
33	075	=	=

Code$_8$	USASCII Equivalent$_8$	TTY Character	LPR/LCG Character
34	054	,	,
35	055	-	-
36	056	.	.
37	057	/	/
4Ø	040	space	blank
41	101	A	A
42	102	B	B
43	103	C	C
44	104	D	D
45	105	E	E
46	106	F	F
47	107	G	G
5Ø	110	H	H
51	111	I	I
52	112	J	J
53	113	K	K
54	114	L	L
55	115	M	M
56	116	N	N
57	117	O	O
6Ø	120	P	P
61	121	Q	Q
62	122	R	R
63	123	S	S
64	124	T	T
65	125	U	U
66	126	V	V
67	127	W	W
7Ø	130	X	X
71	131	Y	Y
72	132	Z	Z
73	044	$	$
74	043	#	#
75	100	@	@
76	136	↑	^
77	137	←	_(underline)

INPUT CHARACTERS

The following classes of input characters are recognized by the Monitor:

OCTAL DIGITS

consists of characters Ø through 7.

DECIMAL DIGITS

consists of characters Ø through 9.

ALPHANUMERIC CHARACTERS

consists of alphabetic characters A - Z, plus digits Ø through 9, plus period ".".

STRING CHARACTERS

consists of all AMOS internal characters except %, ?, and ", which may be internally represented by the input strings %%, %?, and %", respectively.

MONITOR CONTROL STATEMENT SUMMARY

OPERATION (ARGUMENT1, ARGUMENT2, ..., ARGUMENT N)!

Control Parameter Specification

TYPE!

DISPLAY!

SYST (VOL) !

ASSIGNIN (FILE, VOL) !

PANEL ((REGISTER-LIST))!

BOUND (FIRST, LAST)!

MODE ("TYPE")!

RELOAD(PACK)!

HOME(UNIT)!

CLEAR!

VERSION (VERS. "REV")!

Memory Search, List, and Change

SRCHW (WORD, MASK, FIRST, LAST, "MODE")!

SRCHA (ADDR, FIRST, LAST, "MODE")!

LIST ("MODE", FIRST, LAST)!

TRAPS (FIRST, LAST)!

FILL (VALUE, FIRST, LAST)!

OPEN (ADDR, "MODE")!

Resulting typeout		Input options
LOCATION: CONTENTS		NAMES VALUE DCH CCH
where:		means:
CCH		enter any VALUE, close location, and:
	,	open next
	←	open previous
	;	open location addressed by contents
C/R		return to Monitor
DCH		
	@	list symbolic location
	=	list contents in other MODES
	/	list cell addressed by contents
	\	ignore previous VALUE
	[	ignore following comment
	#	close location and open cell at VALUE
	$	enter VALUE without closing

VALUE = An expression

NAMES = Undefined symbols followed by ":" to name "LOCATION"

Memory Allocation

RESET!

START ("TITLE", VOL)!

START (FILE, VOL)!

LOAD ("ENTRY", VOL)!

LOAD (FILE, VOL)!

MARK!

RELEASE (N)!

Symbol Table Control

DEFINE (NAME, VALUE, NAME, VALUE, ...)!

DELETE ("NAME", "NAME", ...)!

NTRY (NAME, LOC, NAME, LOC,....)!

DNTRY ("NAME", "NAME", ...)!

DUMPS!

READS ("TITLE", VOL)!

Execution Control

DO (INSTRUCTION, (REGISTER-LIST))!

GOTO (ADDR, (REGISTER-LIST))!

SNAP (FIRST, LAST, "MODE", (REGISTER-LIST))!

RETURN ((REGISTER-LIST))!

STOP!

Version Setting and Interrogation

VERSION (VERS, "REV")!

VERSION!

Paper Tape I/O

READB ((FIRST, LAST))!

PUNCH ((FIRST, LAST)!, CHECK)!

FEED (LENGTH)!

Disk Storage Allocation

INITIALIZE (PACK, "IDENTIFICATION STRING")!

CREATE (PACK, VOLUME, #CYLS)!

CHANGE (PACK, VOLUME, ±#CYLS)!

PURGE (PACK, VOLUME)!

REMOVE (PACK, VOLUME, FILE)!

REMOVE (PACK, VOLUME (FILE1, FILE2, ...))!

Disk Input/Output

DUMP ("TITLE", VOL)!

RESET ("TITLE", VOL)!

LOADA (FILE, VOL)!

WRITE (FIRST, LAST, "TITLE", VOL)!

COPY (FIRST, LAST, IVOL, OVOL)!

COPY ((F1, L1, I1, OVOL), (F2, L2, I2), (F3, L3, I3),...)!

COPYAMRMX (PACK)!

Disk Volume/File and Memory Map Listing

LISTD (PACK)!

LISTV (PACK, VOLUME, VOLUME, VOLUME, ...)!

LISTA (PACK)!

MEMAP!

Arguments

VOL = Specified Disk Volume (pvv_8 = pack p, volume vv)

FILE = Specified File Number

"TITLE" = Title of a Specified File

"ENTRY" = Name of an Entry Point

NAME = A symbolic name enclosed in quotes, " ", in the DELETE and DNTRY operators and not enclosed in quotes in the DEFINE and NTRY operators.

"MODE" = Typeout Mode: "O", "A", "S", "F", ...

REGISTER LIST = Values for AR, BR, IC, OVF

FIRST, LAST = Lower and Upper Bounds in a Monitor Operation; or first and last files in a COPY operation.

N = A Level Number

CHECK = Value for checksum

LENGTH = Feed-hole count

VERS = Program version number (1 through 31_{10})

"REV" = A single program revision character (i.e., "A")

VOLUME = Volume number (1 through $4\emptyset_8$)

PACK = Disk pack ($\emptyset$ through 7)

ADDR = Expression yielding 15-bit address value

WORD = Expression yielding 30-bit value

MASK = 30-bit expression value used as search mask

INSTRUCTION = Expression yielding 30-bit machine instruction

"IDENTIFICATION STRING" = 19_{10} character or less string used for disk pack initialization

#CYLS = Number of cylinders for volume creation or ± number of cylinders for volume allocation change

IVOL = Input VOL for disk file copy

OVOL = Output VOL for disk file copy

SYNTACTIC STRUCTURE OF MONITOR STATEMENTS

The following is an extended BNF description of the control statement syntax accepted by AMRMX.

< control statement >:: =< term > ! | < term >< argument list > !

< argument list > :: = (< argument > { , < argument > }$_{0}$)

< argument > :: =< expression > | | (< argument list >) | "{ < character > }$_{6}$ "

< expression > :: =< sign >< term > | < expression > {< rotation > }$_{1}$ |
< expression > { < op > }$_{1}$ < term >

< term > :: = { < octal digit > }$_{1}^{10}$ | { < decimal digit > }$_{1}^{9}$. | {< alphanumeric> }$_{1}^{10}$ |
. |] | "{ < character > }$_{1}^{5}$ "

< rotation > :: = ! B | ! K | ! H

< op > :: =< sign > | ↑ | ' | &

< sign > :: = + | - | space | tab

< octal > :: = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7

< decimal digit > :: =< octal digit > | 8 | 9

< alphanumeric > :: =< decimal digit > | A | B | C ... | X | Y | Z | .

< character > :: =< alphanumeric > | < op > | , | / | $ | # | @ | [| = | \ | (|) | ← | "|
carriage return |] | ! | : | ; | * | %% | %?

Explanation of notation

< x >	"a member of the syntactic class of x"
:: =	"is defined to be"
|	"or"
{ x }$_{m}^{n}$	"from m to n" instances of x (if n is not specified, can be arbitrarily large)
...	"and so on"

GENERAL

ARITH is a relocatable AMOS library program that performs fixed and floating point arithmetic operation on the AGT (Adage Graphics Terminal). It is available in two versions:

Version 1 - DPR1 or OPC

Version 2 - DPR2 or AGT (with EAU).

ARITHMETIC ROUTINES

ARITH contains the following routines:

A. Integer Arithmetic Routines

Two integer arithmetic routines are available in the ARITH package: integer multiply and integer divide. Integer computations are performed to 29 bits of precision by the routines. The result is in (AR) when control is returned to the user program. The magnitude of a number is in bit positions 1 - 29, and the sign of the number is in bit position 0. Negative numbers are represented as the ONEs complement of the corresponding positive number. The result is accurate to 29 bits.

1. Integer Multiply

Name:	98T	
Purpose:	To divide I by J	
Calling Sequence:	JPSR	98T
	0	J

Operation: The dividend, I, is loaded in the AR register prior to executing the call to the 98T routine. The routine develops the quotient I/(J) in AR, and returns control to the calling program at the instruction immediately following the divisor address, J. The remainder, if any, is disregarded by the routine. If the divisor is zero, (AR) is set to zero and the divide check error bit is set.

B. Floating Point Arithmetic Routines

The ARITH Package includes four routines to perform floating point arithmetic.

- Floating Point Add 9CQ
- Floating Point Subtract 9CR
- Floating Point Multiply 9CS
- Floating Point Divide 9CT

Floating point computations are performed to 28 bits of accuracy. The result is rounded to a 21-bit mantissa with an 8-bit exponent. The result will be in the AR register when control is returned to the user program.

The form of a real number is as follows:

- Bit position 0 - Sign
- Bit position 1-8 - Exponent (with a bias of $+200_8$)
- Bit position 9-29 - Mantissa (binary fraction)

A negative number is represented as the 30-bit ONES complement of the corresponding positive number.

1. Floating Point Add

Name:	9CQ	
Purpose:	To add A+(B)	
Calling Sequence:	JPSR	9CQ
	0	B

Operation: The augend, A, is loaded in the AR register prior to executing the call to the 9CQ routine. The routine computes the sum A+(B) in AR, and returns control to the program at the instruction immediately following the addend address, B. If overflow occurs, the (AR) is set to zero, and the overflow error bit is set.

2. Floating Point Subtract

Name:	9CR	
Purpose:	To subtract A-(B)	
Calling Sequence:	JPSR	9CR
	0	B

Operation: The minuend, A, is loaded in the AR register prior to executing the call to the 9CR routine. The routine computes the difference A-(B) in AR, and returns control to the main program at the instruction immediately following the subtrahend address, B. If overflow occurs, the (AR) is set to zero and the overflow error bit is set.

3. Floating Point Multiply

Name:	9CS	
Purpose:	To multiply A*(B)	
Calling Sequence:	JPSR	9CS
	0	B

Operation: The multiplicand, A, is loaded into the AR register prior to executing the 9CS routine. The routine computes the product A*(B) in AR, and returns control to the main program at the instruction immediately following the multiplier address, B. If overflow occurs, the (AR) is set to zero and the overflow error bit is set.

4. Floating Point Divide

Name:	9CT	
Purpose:	To divide A/(B)	
Calling Sequence:	JPSR	9CT
	0	B

Operation: The dividend, A, is loaded into the AR register prior to executing the 9CT routine. The routine computes the quotient A/B in AR, and returns control to the main program at the instruction immediately following the divisor parameter, B. If the divisor is zero, or if overflow occurs, the (AR) is set to zero and the divide check error bit is set.

C. I to J Power

This routine raises the integer number in (AR) to the power specified by the integer number included in the parameter statement following the call to the 98E routine. The result is computed to 29 bits of accuracy, with the sign of the result in bit position 0, and the magnitude of the number in bit positions 1-29.

The result is in the AR register when control is returned to the user program.

Name:	98E	
Purpose:	To raise I to (J) power	
Calling Sequence:	JPSR	98E
	0	J

Operation: The integer I is loaded in the AR register prior to executing the call to the 98E routine. The routine evaluates $I^{(J)}$ in AR by multiplying I by itself (J)-1 times, and returns control to the program at the instruction immediately following the address, J. If the exponent is zero, the result is equal to 1. Any exponent less than zero causes a zero result. If overflow occurs, (AR) is set to zero and the overflow error bit is set. If I is zero and (J) is less than one, (AR) is set to zero and the indeterminate error bit is set.

Subroutine Called: 98S

Accuracy: 29 bits

D. A to I Power

The routine raises the floating point number in AR to the power specified by the integer exponent following the 9BE call. Computations are performed to 28 bits of accuracy. The result is rounded to the standard floating point format of a 21-bit mantissa and 8-bit exponent. The result is in the AR register when control is returned to the user program.

Name:	9BE	
Purpose:	To raise A to the power of (I)	
Calling Sequence:	JPSR	9BE
	0	I

Operation: The floating point number, A, is loaded in the AR register before executing the call to the 9BE routine.

- To raise A to a positive power, the result is calculated by multiplying A by itself (I)-1 times.
- To raise A to a negative power, the result is determined by the first calculating 1/A, and multiplying it by itself (I)-1 times.

If the exponent is zero, the result is equal to 1. Control is returned to the main program at the instruction immediately following the power address, I. If overflow occurs, (AR) is set to zero and the overflow error bit is set. If A is zero and (I) is less than one, (AR) is set to zero and the indeterminate error bit is set.

Subroutines called: 9CS

Accuracy: Computation is performed with the floating point arithmetic routines giving a relative accuracy of 21 bits.

E. Real to Integer

Converts the real value F in AR to an integer value by truncation. Places this value in AR and the specified location.

```
Calling Sequence:   (AR) is R
                    JPSR    98Y
                            IF
                                [Address of result (integer)
```

F. Integer to Real

Converts the integer value I in AR to a real value and stores it in the specified location.

```
Calling Sequence:   JPSR
                            9CY
                            RI
                                [Address of result (real)
```

G. Utility Routines

POWRF - Used by the External Functions to obtain the binary exponent of a real number.

SICOF - Reduces the range of a SIN or COS argument to the range $|A| \leq \pi/4$.

BASIC INTRINSIC FUNCTIONS

A. General

AGT Basic FORTRAN includes the following Intrinsic Functions:

- ABSOLUTE VALUE (ABS) (IABS)
- FLOAT (FLOAT)
- FIX (IFIX)
- TRANSFER OF SIGN (SIGN) (ISIGN)

The result of a call to an Intrinsic Function is in the AR register when control is returned to the user program.

- If the Function type is real, the result is rounded to a 21-bit mantissa and an 8-bit exponent.
- If the Function type is integer, the result is accurate to 29 bits, with the magnitude of the number in bit positions 1^{-29}, and the sign in bit position 0.

B. Absolute Value

This routine converts a real or integer number to its absolute value. If the argument is of type real, the function ABS, is of type real; if the argument is of type integer, the function, IABS, is of type integer. The argument is specified by the address (possibly indirect) immediately following the ABS or IABS Call. The result is in the AR register when control is returned to the user program at the instruction immediately following the argument address.

Name:	ABS	
Purpose:	Convert the real argument (A) to absolute value	
Calling Sequence:	JPSR	ABS
	0	A

Name:	IABS	
Purpose:	Convert the integer argument (I) to absolute value	
Calling Sequence	JPSR	IABS
	0	I

C. Float

This routine converts a type integer value to type real. The argument is of type integer and the function is of type real. The argument is of type integer and the function is of type real. The argument is specified in the parameter statement immediately following the FLOAT call. The computed real number is in the AR register, in standard floating point format of a 21-bit mantissa and 8-bit exponent, when control is returned to the user program at the instruction immediately following the address, I.

Name:	FLOAT	
Purpose:	Convert the integer argument (I) to a real number	
Calling Sequence:	JPSR	FLOAT
	0	I

D. Fix

This routine converts a type real value to type integer. The argument is of type real, and the function is of type integer. The argument is specified in the parameter statement immediately following the IFIX call. The computed integer number will be in the AR register, to 29 bits of accuracy when control is returned to the user program at the instruction immediately following the address, A.

Name:	IFIX	
Purpose:	Convert the real argument (A) to an integer number	
Calling Sequence:	JPSR	IFIX
	0	A

E. Transfer of Sign

This routine transfers the sign of a real or integer number, by multiplying the sign of the second argument by the absolute value of the first argument. If the arguments are of type real, the function, SIGN, is of type real; if the arguments are of type integer, the function, ISIGN, is of type integer. The arguments are specified by the address (possibly indirect) immediately following the SIGN or ISIGN Call.

The result is in the AR register when control is returned to the user program at the instruction immediately following the second parameter address.

Name:	SIGN	
Purpose:	Transfer the sign of (B) to (A)	
Calling Sequence:	JPSR	SIGN
	0	A
	0	B

Name:	ISIGN	
Purpose:	Transfer the sign of (J) to (I)	
Calling Sequence:	JPSR	ISIGN
	0	I
	0	J

GENERAL

The relocatable AMOS routine ARMW may be loaded (e.g., START ("ARMW")!) and used to convert the relocatable AMOS Monitor, AMRMX (as output by the ADEPT assembler) to type "BIN" (Binary) for loading via bootstrap loaders.

VERSIONS

ARMW, Version 1, is used on MTP-5 systems.

ARMW, Version 2, is used on MTP-7 systems.

CALLING SEQUENCE AND USE

The following calling sequence:

ARMW (FILE, CODE, INPUT TAPE, OUTPUT TAPE)!

where:

FILE	=	File No. of AMRMX, type "RELOC"
CODE	=	0 - Jump to new monitor after conversion
	=	1 - Return to bootstrap loader, AMLDX
INPUT TAPE	=	Tape unit containing the relocatable AMRMX
OUTPUT TAPE	=	Tape unit on which the binary AMRMX is to be written

will accomplish the conversion of AMRMX from relocatable to binary.

GENERAL

This library raises the floating point number in (AR) to the power specified by the floating point number included in the parameter statement following the call to 9CE.

Name:	9CE
Purpose:	To raise A to the power B
Calling Sequence:	JPSR 9CE
	Ø B

OPERATION

The floating point number A is loaded in the AR register before executing the call to the 9CE routine. The routine calculates A^B using the formula:

$$A^B = e^{B\log A}$$

Subroutines Called: 9CE, EXP, ALOG, and arithmetic subroutines in the ARITH package.

Accuracy: Computation is performed with the floating point arithmetic routine, giving a relative accuracy of 21 bits.

SOFTWARE REQUIREMENTS

9CE, EXP, ALOG, arithmetic subroutines in ARITH.

HARDWARE REQUIREMENTS

The hardware configuration is the same as that required for the above software items.

CORE REQUIREMENTS

20_8 words

EXECUTION TIME

300 μs

adage

AGT BUILD OPERATOR, BUILD

Programmer's Reference Manual

Revision A

April 1969

ADAGE, INC.
1079 Commonwealth Avenue
Boston, Massachusetts 02215

CONTENTS

I. INTRODUCTION

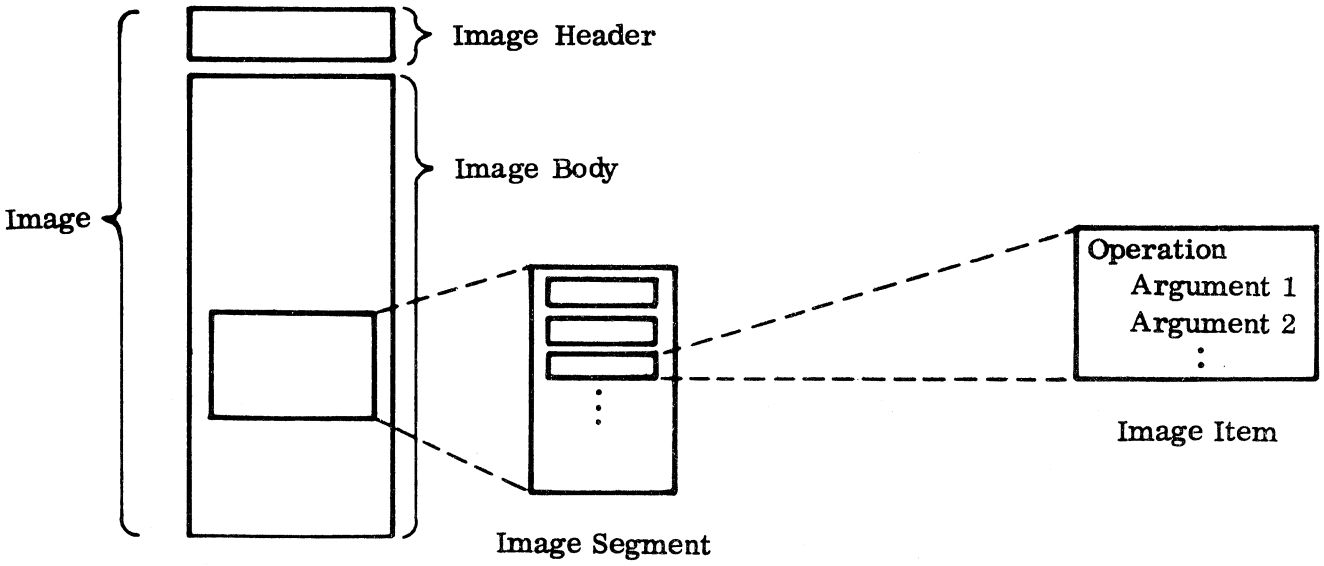
A. BUILD OPERATOR

The Build Operator is a program which permits an online user of an Adage Graphics Terminal to create, assemble, and adjust visual representations of program variables, models, or structures. The two or three dimensional visual representation is called an Image. The data structure in memory which defines an Image will also be referred to as the Image Description or simply the "Image." When an Image Description is processed for viewing, the resulting two dimensional projection displayed on the CRT will be referred to as its current "Picture."

B. DSPLY OPERATOR

The DSPLY Operator is a program which processes selected Image Descriptions and generates their resulting Image for display at a chosen frame-rate.

The Image Descriptions processed by DSPLY consist of an Image Header and an Image Body. The Image Header only contains a character string with the Image's name; the remainder of the Image Description is the Image Body.



An Image Segment is a portion of an Image Body in a contiguous area of memory. An Image Body is not necessarily kept as a contiguous table in memory. Segments of an Image Body may be in separate linked tables or scattered throughout a programmatic model or its data structure.

1. Image Items

Each Segment of an Image Body consists of a set of contiguous Image Items. Image Items are the basic elements of an Image Description. An Image Item consists of an item code and a set of argument references.

There are four classes of Image Items:

a. Element Generating

These Items specify the visual elements of an Image Description such as lines, points, and characters.

b. Transform Specifying

These Items apply affine transforms such as scalings, rotations, and translations to bracketted Image Segments.

c. Control

These Items effect control over image scanning, and execution of any chosen programs.

d. View Defining

These Items specify effects on the projected Picture. These effects include cut-offs, viewing windows, and intensity modulated depth cueing.

2. Item Arguments

Many of the available Image Items require one or more arguments to define their effect. Examples of several Items and the arguments they use are given in the following table:

Type of Items	Sample Items	Arguments
Visual elements	Lines	Coordinates of end-points
	Text	List of characters
	Table	Addr of packed vector values
Transforms	Scale	Scale-factor
	Rotation	Angular rotation
Control	Subroutine Call	Subroutine entry point
	Subimage Call	Address of image
	Transform delimiting bracket	None

For a complete specification of Image-Description data structures, formats, and their processing, the reader should refer to the "DSPLY Programmer's Reference Manual." The mnemonics for Image Items and their arguments as given in the DSPLY/PRM are used by the BUILD Operator.

C. BUILD SYSTEM ENVIRONMENT

The BUILD Operator provides an online user with facilities for creating and linking image segments and binding them to loaded Programs or Images. All control of the BUILD Operator is effected via interactive use of online inputs such as the data tablet and foot pedals. The results of all BUILD actions are accessible for viewing via the display screen.

1. Resident Programs

During operation of the BUILD Operator, the online user will have loaded programs of the following four types:

a. The BUILD and the DSPLY Operators which present the results of BUILD usage to the user.

b. I/O drivers for the online devices by which the BUILD user manipulates BUILD Image Segments.

c. Other resident or selected standard system programs for storage and peripheral communication, program loading or dumping, control statement processing, etc. ...

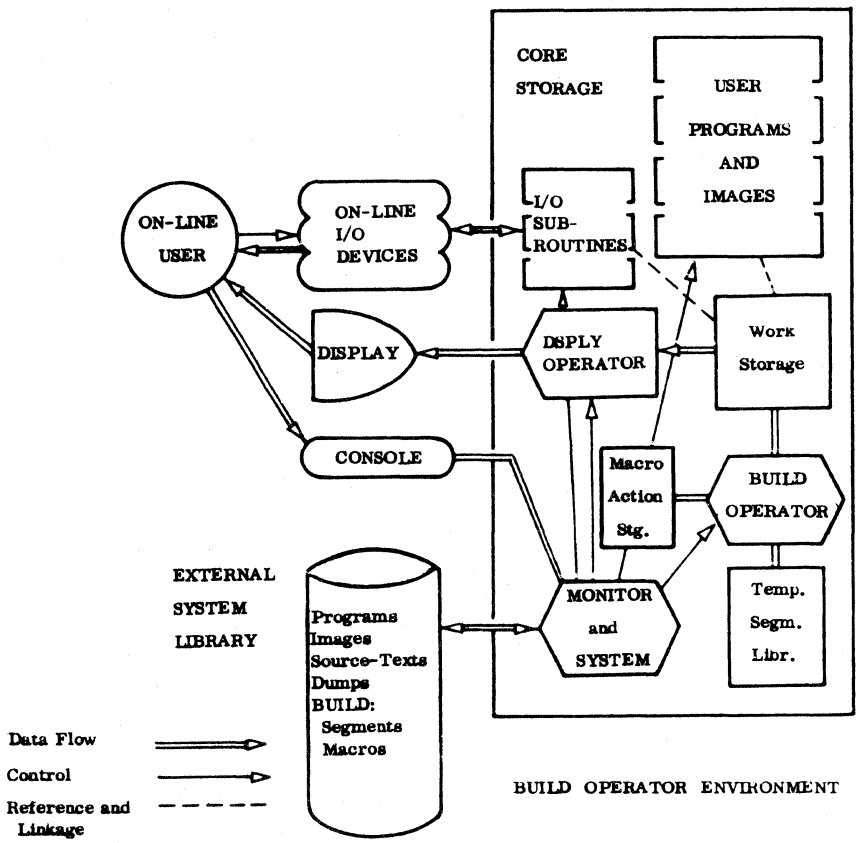
d. The user's problem, or its model, coded as programs or Images to which the BUILD user is to bind his created or assembled visual imagery.

2. External System Library

In addition to the Programs and Images loaded in core, the user has access to all files resident in the External System Library on online mass storage (tape/disk/other).

The files in the External System Library may be compiled or assembled object programs or images, or the source texts which generated them. Also binary absolute dumps of core areas and previously link-loaded symbolic core-loads may be generated or re-instated from the Library via System functions.

Of interest to a BUILD user are files of saved Macro-Action definitions for extending the set of BUILD operations and files of Image Segments for RETRVing into a local Temporary Library of available definitions. These can then be used for local Image BUILDing.



BUILD OPERATOR ENVIRONMENT

D. BUILD OPERATOR COMPONENTS

During operation, the BUILD Operator maintains four classes of information:

1. Work Storage

This is a table which contains the Image Segment currently being defined via user interaction. The contents of the Work Storage are continually being processed by the DSPLY Operator and presented to the user.

2. Temporary Library

This is a file of saved Image Segments. Segments created in the Work Storage may be saved in the Temporary Library at any time. Segments saved in the Temporary Library are used as building pieces to create new segments in the Work Storage. The Temporary Library initially contains one-item segments for the items implemented by the DSPLY Operator.

Selections from the Temporary Library may be gathered and filed in the External System Library by the SAVE Operator. Sets of saved Image Segments may be fetched from the External System Library and instated in the current Temporary Library via the RETRV Operator.

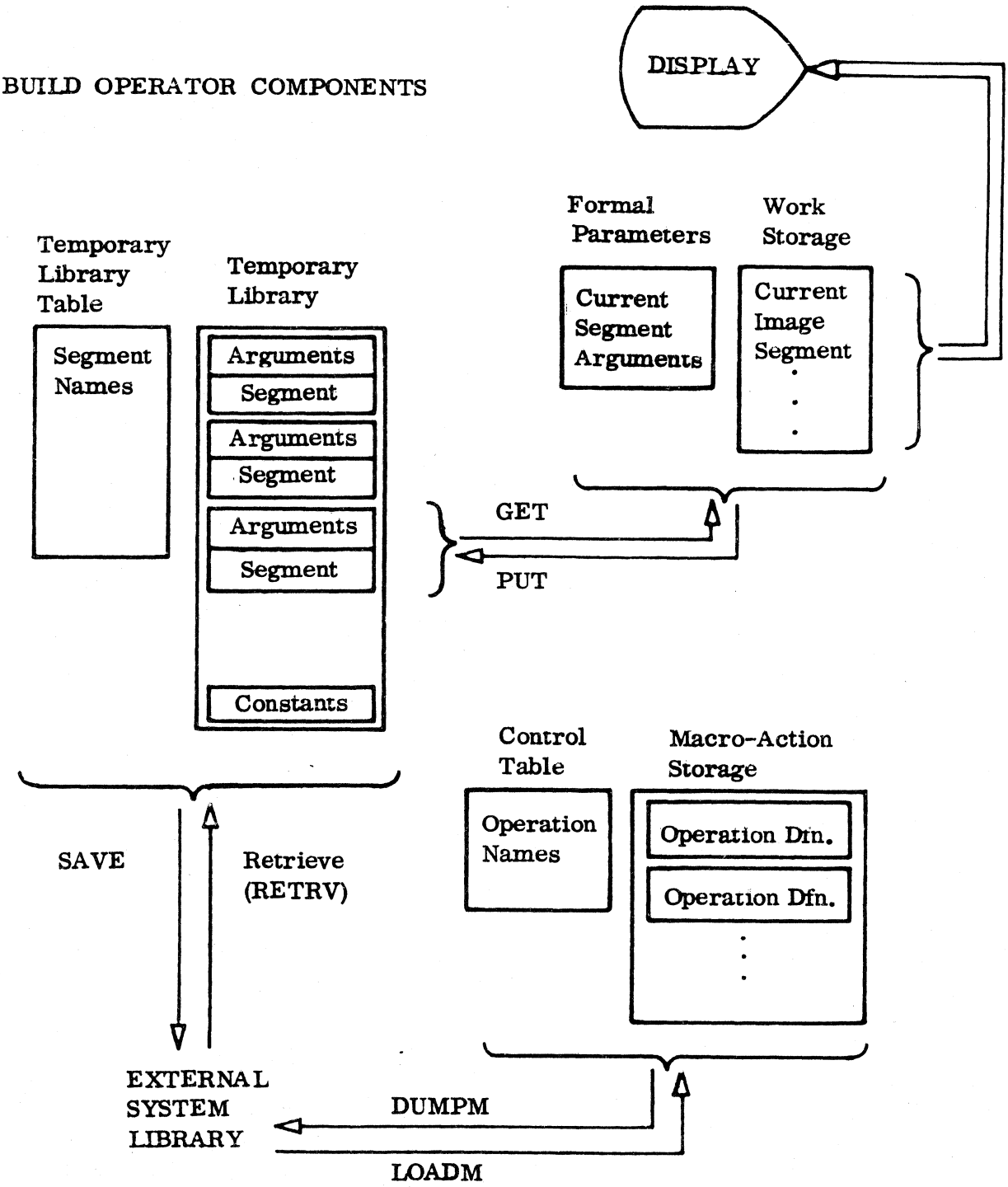
3. Macro-Action Storage

All operations performed by the BUILD Operator are encoded as strings of text characters in a machine independent form. These strings specify the sequence of actions, either internal transformation or user communication, required to implement any BUILD function. Via the BUILD Macro Facilities, a user may either create anew or condense a sequence of existing actions into a single new action to be added to the existing BUILD repertoire. A set of user defined action sequences may be selected and filed into the external System Library. Sets of filed BUILD action-sequences may be fetched from the library and instated; thus, facilities implemented under BUILD may be selectively tailored for different applications.

4. Tables and Directories

During the use of the BUILD Operator, there are many occasions where entities are created, objects are referenced, or selection of alternatives must

BUILD OPERATOR COMPONENTS



be made. For this purpose, Names are used to label created entities, to refer to defined objects, and to list available options for input of user decisions.

The BUILD Operator maintains all defined Symbols (Names) and their assigned definitions in tables and directories. These are all stored in a threaded list storage area, which may be extended by request of the online user.

The BUILD Operator initially has six tables defined. The user may extend, replace or create new tables during operation. The six initially defined tables are heavily used for communication with the user to control BUILD Operation. They can be used to present the user with alternate courses of action for his selection.

5. Initial Tables

The six initially provided tables are:

a. Control Table

This is a directory of the Macro-Action Storage contents. All BUILD Operations available to the user may be selected for execution from this table.

b. Temporary Library Table

This is a directory of the contents of the Temporary Library of Image Segments. All currently defined Image Segments available for use in building the Image Segment in the Work Storage can be selected from the Temporary Library Table.

c. Table of Tables

This is a directory of all currently defined Tables available to the BUILD user for use or modification.

d. External Symbol Table

This is a table through which a BUILD user may access values, images, or subroutine entries loaded in core. These are used as arguments to Image Items or to Segments in the Work Space. The contents of this table are created during loading or other system operations which create External Symbol definitions.

e. Work Storage Table

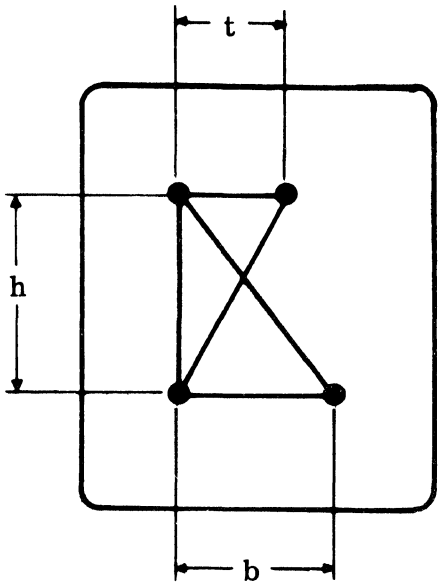
The Work Storage Table is a directory of all Image Segments in the current Work Storage.

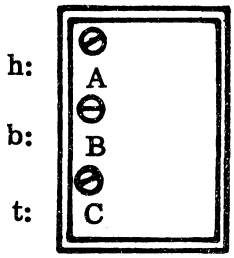
As each new Segment is selected from the Temporary Library and added to the Work Storage, its name is added to the Work Storage Table.

f. Formal Parameter Table

This table contains Formal Parameters. Each Formal Parameter consists of an Argument for the current Work Storage Segment and its current value assignment. An Argument for an Image Segment is a Name and the set of Item arguments in the Segment, jointly taking the Name's current value assignment. An assignment is a Name of a value and a reference to the value. The Formal Parameter Table will be represented as a set of Arg-Value name pairs. The set of all first names for all pairs forms a directory of arguments for the current image being built in the Work Space. As values are selected and assigned to these arguments, the second name is added. This is the name of the value assigned.

When the Work Space Image Segment is finished and saved in the Temporary Library, both the segment in the Work Space and the contents of the Formal Parameter Table are filed in the Temporary Library as the definition of the Segment being saved.





Work Storage

w1:	MOVE 1,1,0
w3:	DRAW (f2),1,0
w6:	DRAW 1,(f1),0
w9:	DRAW 1,1,0
w12:	DRAW (f3),(f1),0
w15:	DRAW 1,(f1),0

Current Image Segment

Formal Parameters

F.P. Table

f1(w7,w13,w10):	HIGHT:	DIAL A
f2(w3):	BASE:	DIAL B
f3(w12):	TOP:	DIAL C

Current Segment Arguments

Current Value Assignments

Example of Formal Parameters

Schematic of a bracket with variable dimensions under user control.

II. ENVIRONMENT REQUIREMENTS

For its operation, the BUILD Program requires the presence of AMRMX for monitor statements; it requires the DSPLY Operator to process the Image Descriptions for CRT display and it requires user I/O or cursor communication via such device sampling routines as RADT, RVCD, or RJSB. The BUILD Operator maintains a large ($10,000_8$ locations) buffer-internal to the BUILD program to store the Image Segment description currently appearing on the CRT (Work Storage), and to save copies of Image Description and various constants and strings (Temporary Library). An additional buffer ($1,000_8$) internal to BUILD is used to store the tables created by the user in the operation of BUILD. Additional core storage may be requested by BUILD should more be required. In addition to the libraries resident in core, external libraries may be used to store Image Segment Descriptions and BUILD Operation statement definitions.

III. IMAGE UNDER BUILD

A. WORKING STORAGE (W.S.)

The Working Storage is that part of the in-core library that contains the image currently being displayed on the CRT, and currently being acted upon by the various operations of the BUILD Operator. The BUILD Operator allows the contents of the Working Storage to be modified (i.e., an image may be sketched in, altered, or assembled by the user while he is viewing it).

B. DYNAMIC VARIABLES

Each Item of an Image Description may require one or more arguments to specify its effect (i.e., a DRAW item requires 3: Z, X, Y coordinate values of vector end-point while a SAVT item requires none). The addressing modes implemented by the DSPLY Operator permit these arguments to be given as immediate values or as addresses which refer to the value. An image can be defined with some (or all) of its arguments referring to changing variables in running programs or to online inputs. Thus, an image being worked on via BUILD may be dynamically changing to reflect a programmed model or respond to online control.

C. FORMAL PARAMETERS

A Formal Parameter is defined as a data block that relates arguments of items in an Image Segment description to values elsewhere in memory. A Formal Parameter is presented as a pair of names: its own and that of the value addressed by those Image Segment arguments it represents. It also maintains the address of the value referenced by the Image Arguments and a list of all its occurrences in the Image Segment. This facilitates re-assignment of formal parameters.

D. FORMAL PARAMETER TABLE

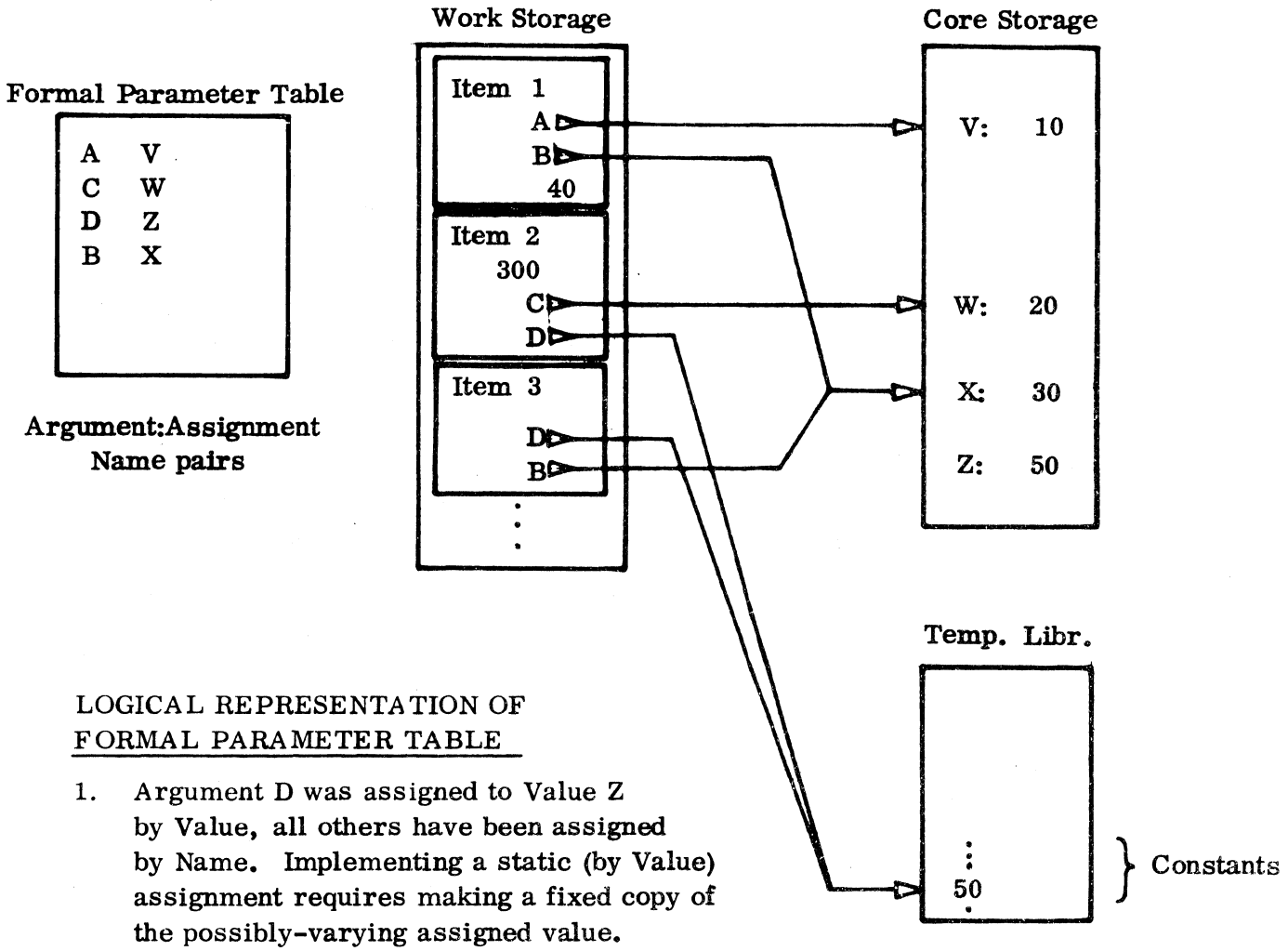
The Formal Parameter Table is a list of all Formal Parameters associated with the image currently stored in the Working Storage. Each new item or set of

items called into the working storage for display and adjustment has names of all its formal arguments added to the Formal Parameter Table.

1. Use of Formal Parameters

The usefulness of Formal Parameters is that they permit user-defined, dynamic variation of the arguments of an Image Description, while that image resides in the Working Storage. A user may assign an Item Argument, via its entry in the Formal Parameter Table to an online input (dial, tablet, joystick, etc.) for the purpose of adjusting the argument to a satisfactory value by sight. Then the user may "HOLD" the argument so that it no longer tracks the input dynamically. When adding a non-dynamic item to an Image Description, the user may define the arguments to be precise values of defined constants. A Formal Parameter may at any time be modified so that its references relate to different values in core (i.e., change the arguments of a ROTZ item, the angle of a Z-rotation, AZ, from addressing dial input VCDA to addressing dial VCDF).

Thus, via the Formal Parameter Table, Image Item Arguments may be assigned static, (by Value), or dynamic (by Name) whether the value of the variable being assigned is dynamically changing or not. Formal Parameters may be permanently merged together so that all argument references to one Formal Parameter are changed to reference the other. The name of a Formal Parameter may be changed so that the user may construct an image whose parameter names are clear and distinct. When the user is satisfied with an argument's assignment (whether static or dynamic) he may permanently "close-off" its Formal Parameter, deleting the entry from the Formal Parameter Table and in effect reducing by one the number of Formal Parameters associated with the image in the Working Storage.



LOGICAL REPRESENTATION OF
FORMAL PARAMETER TABLE

1. Argument D was assigned to Value Z by Value, all others have been assigned by Name. Implementing a static (by Value) assignment requires making a fixed copy of the possibly-varying assigned value.
2. Arguments #2 of Item 1 and #1 of Item 2 are not (now) Formal Parameters.
3. Arguments #2 of Item 1 and #2 of Item 3 (also 3rd of Item 2 and 1st of Item 3) have been merged.
4. All the references shown are "assignments" actually contained in the Formal Parameter area and the Items using them do so by addressing indirectly through pointers maintained for each Formal Parameter.

E. TEMPORARY LIBRARY

A portion of the BUILD internal buffer is designated the Temporary Image Segment Library (as opposed to a permanent library on tape or disc). This temporary library contains all user defined strings and constants fetched throughout the operation of BUILD. It also contains copies of Image Segments that at one time were resident in the Work Storage and were "PUT" away. After a BUILD user has assembled an image portion in the Working Storage, chosen any desired assignment for arguments and adjusted any fixed values, the entire contents of the Working Storage may be given a name and copied along with the contents of the Formal Parameter Table into the in-core BUILD Temporary Library. The name given to the Image Description will then be added to the Temporary Library Table, which is a directory of all Image Segments residing in the Temporary Library.

The Temporary Library (and its Table) initially contain all DSPLY Image Items as defined in the DSPLY/PRM with their arguments all Formal (i.e., with each argument there is associated one Formal Parameter with the name given to the argument in the DSPLY/PRM). Thus, the predefined MOVE Image Item has three Formal Parameters, FZ, FX, and FY, while the predefined ROTZ Image Item has one Formal Parameter Name AZ. The predefined mnemonics used by BUILD correspond to the item mnemonics and argument mnemonics used in the DSPLY/PRM.

During the use of BUILD, all named Image Segments (including all assemblages the user has previously created, named, and put into the Temporary Library) are available to be added to the Working Storage Image Segment. When an image portion is added to the Work Storage all Formal Parameters for the image portion are added to the Formal Parameter Table. The user may also have generated fixed planar Segments using one of the available sketching modes.

IV. CONTROL OF BUILD

A. BUILD OPERATIONS

The BUILD Program can perform various operations on the Image Segment residing in the Working Storage. An operation may actually consist of a series of actions and is called a BUILD Action Sequence. The online user selects and executes the desired BUILD Action Sequence. Action sequences may have arguments which must be specified by the user when requested.

B. MENUS

Selections of action sequences to be executed, or arguments of action sequences, are made by the user from a displayed menu or table. The BUILD Operator maintains many distinct menus for display. The selection process is central to the operation of BUILD.

1. Control Table Use

When no action sequence is currently in progress, a table (menu) of available BUILD Action Sequences is displayed. This list of available BUILD Operations will be called the Control Table. When a selection is made, the action sequence is executed up to the point where an additional argument must be selected by the user. At that point, the BUILD Operator automatically presents for display the proper menu for this selection (such as Temporary Library, the Formal Parameter Table, the External Symbol Table or some other user-defined table). When a selection is finally made, the action sequence steps to seek a further argument, or to perform the operation. When the action sequence is finished, the Control Table is again displayed so that another action sequence may be initiated.

2. Format and Use of the Menus

Some menu will appear on the CRT during most of the time BUILD is in operation. Any menu will appear on the left quarter of the screen and will not interfere with the Working Storage Image, which normally appears on the CRT. All menus presented to the user on the CRT are in a standard format which per-

mit scanning and selection by means of the same user actions. Only one table is displayed at any one time, no more than sixteen entries are displayed at one time, and each entry consists of from one to ten characters.

Associated with each menu may be a message, which will appear in italics at the top of the screen. Each menu also has a table name, which appears above the menu list.

The user uses the data tablet stylus to position a blinking double asterisk to the left of the desired menu entry. When the blinker is positioned next to the selection desired, the user expresses his choice by depressing the pen stylus. The action-sequence then steps either to display another table for argument selection, or to complete the operation.

In addition to the information described above, which changes from menu to menu, there is displayed in large print in the lower left corner of each menu a set of unchanging menu selection aids. These items each consist of one character (see layout which follows) which perform the following actions when selected.

M: Selection of this character will display more entries (up to 16) from the current menu.

L: Display less entries.

U: Roll the menu up half a frame, if possible (needed to scan tables with more than 16 entries).

D: Roll the menu down half a frame, if possible.

A: ABORT the current action sequence, and present the Control Table to allow operator to select a new action sequence.

T: Allow the operator to choose a different menu from which to make the currently requested selection. The table of Tables is presented for choice of alternate menu, then it is presented for selection of the current requested input.

H, Q: These menu aids are displayed only in Macro definition mode. See Section VI-D, Macro, of this document for details.

The menu aids may be selected by positioning the tablet stylus until a double blinking underbar is situated under the desired menu aid. Depressing the pen stylus will perform the selected aid function.


MESSAGE FOR THIS USER SELECTION
NAME OF TABLE
SELECTION 1
SELECTION 2
SELECTION 3
SELECTION 4
A U M H
T D L Q

C. CONTROL TABLE: OPERATIONS

All the facilities initially built into the BUILD Operator are listed in a Control Table. It may be extended via the MACRO facilities covered in Section VI-D. The Control Table is presented whenever a new action sequence may be initiated. By selecting the name of an action sequence from the displayed control table, the user activates the sequence.

The non MACRO-extended BUILD facilities fall into the following four general categories:

1. Work Storage Management

This includes commands for emptying, saving, or adding to the image portion being worked on in the Working Storage.

2. Argument Processing

This includes commands for creating, merging, renaming or deleting Formal Parameters and for setting or altering their assigned values or changing assignments between static and dynamic.

3. I/O Communication

These commands create menu tables for user input selection, display them, process the resulting selections. They also include the ability to input strings of characters from the keyboard.

4. MACRO Facilities

These commands permit the user to compose new action sequences from existing ones. Thus a useful sequence may be carried out by the user as the "definition stage" of a macro input. The resulting sequence is saved and assigned a name which is entered in the Control Table. Later when the execution of any such predefined action sequence is desired, it can be selected from the Control Table as if it had been "built-in" and the entire set of its constituent action sequences performed as one user step. Such selection of a macro name to activate its definition will be termed "invoking" of the MACRO.

V. EXTERNAL BUILD ENVIRONMENT

A. EXTERNAL SYMBOLS

Many Image Items take as arguments numerical values representing angles, coordinates, or counts, but some Items make references to character strings, executable subroutines, other images or image portions. In order to permit the static or dynamic assignment of Formal Parameter values to range easily over all of these value classes, the assignment commands permit value selection over all defined external symbols. These include all loaded entry-point names as maintained in the External Symbol Table. It is recommended that when preparing application packages those desirable external symbol choices be gathered into new Tables for presentation to the user.

B. EXTERNAL ROUTINES AND IMAGES

A user-created BUILD Image portion can be easily linked to new values or coding entered from the on-line monitor. Also pre-assembled routines or images can be loaded from the External System Library and referenced by new BUILD-created image segments.

C. EXTERNAL SYSTEM LIBRARY IMAGE SEGMENTS

The SAVE and RETRV Operators are available as standard system programs to gather selected Image Segments filed in BUILD's in-core Temporary Library and save them as a set, with any Formal Parameter variables, on the External System Library. These can later be selected for retrieval and reinstatement into some later core resident BUILD item library for use in subsequent BUILDing.

D. EXTERNAL CONTROL-SEQUENCE-DEFINITION MACROS

A BUILD user may select a set of MACRO defined action-sequences to be dumped as a set into the permanent External System Library. Previously dumped sets of control definitions may be loaded and the constituent commands added to the Control Table. Thus, the universe of possible useful commands may be selectively shuffled to provide more optimum command sets for differerent applications.

VI. INITIAL BUILD OPERATIONS AND THEIR USE

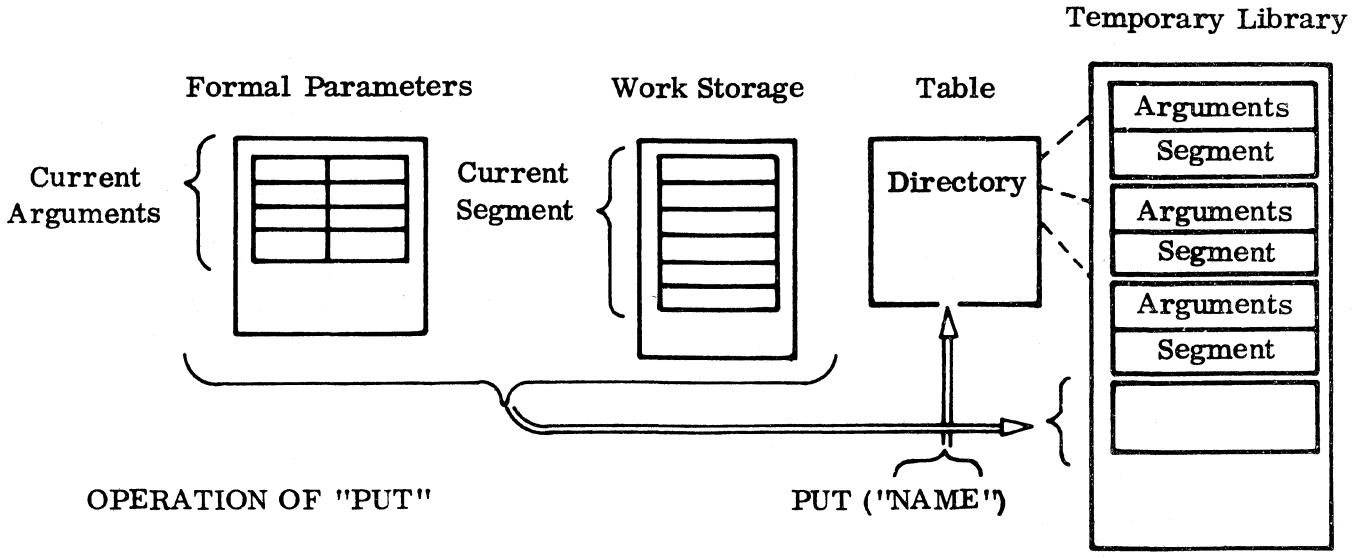
A. WORK STORAGE CONTROL OPERATIONS

1. CLEAR

Executing this operation causes the Working Storage and the Formal Parameter Tables to be emptied. Any created image portion being viewed will disappear. It will no longer be available for BUILD use.

2. PUT

This operation causes a copy of the Image Segment being displayed (the Working Storage) to be saved with all its current static or dynamic argument assignments (Formal Parameters) into the core-resident BUILD Temporary Library. The names, identity, and current assignments of all Formal Parameters are noted and saved with the image portion. The keyboard is activated and the user is requested to type in a name under which the image portion is to be filed. The name is added to the directory for the Temporary Library.



3. GET

This operation presents the user with the Temporary Library Table (directory) for selection. Once the user selects the name of an entry from the Temporary Library, the Image Segment filed under that name is appended to current Working Storage contents and therefore appears in the display under the current transformation. The Formal Parameter definitions of the filed Image Segment are also retrieved and appended to those currently in the Formal Parameter Table.

4. SKETCH

This operation displays a tracking cross on the current X-Y plane. The tracking cross can be moved over the X-Y plane by the ADT1 Tablet stylus. When the stylus is depressed, a blanked (MOVE) vector item to the current cross position is added to the current Image Segment in the Work Storage. Any further motion of the stylus (while depressed) will add line segments to the current Image Segment. Releasing stylus pressure will again track in preparation to enter a MOVE item.

5. SKBAND

This operation permits sketching of straight lines. A menu is presented with a D or an M entry. Selecting D will remove the D and present the M. Selecting M will remove the M and present the D.

While the D is displayed, a line will be drawn from the last point of the current Image Segment to the current tracking cross position. Depressing the stylus (anywhere but on the D) will stop moving the endpoint of the line, enter the line into the Work Storage as part of the current Segment, and start a new line from there to wherever the tracking cross is subsequently moved.

While the M is displayed, no line will be displayed to the tracking cross position; depressing the stylus will enter a MOVE (blanked vector Item) to the cross position into the current Image Segment.

6. SKETCHING - GENERAL

The current Image Segment is maintained when the SKTCH mode is entered; all items "sketched" in will be appended to that image.

RETURNING - Depressing an FNS switch will return to the normal mode where tables are displayed.

ERASING - Depressing an FNS switch will erase the last item constructed in the current SKETCH call. If the operator attempts to erase above the beginning of the current call to the SKTCH routine, then the program will return as in RETURNING. Also, the tracking cross position is moved to the new last beam position after each erase call.

The entire two-dimensional construction built while in Sketch operations, including the tracking cross, is displayed under the current transformation of the array. Thus, if a rotation is currently enstated, the sketched image and the tracking cross will be drawn under that rotation, and the user will see on the screen a two-dimensional projection of the cross and the image he is sketching.

B. ARGUMENT ASSIGNMENT AND FORMAL PARAMETER CONTROL

1. SETVALUE - Set by Value

Sets a Formal Parameter to the current value of any variable (e.g., DIALA). After the SETVALUE operation is selected, the Formal Parameter Table is displayed until a selection is made from it.

After a selection is made from the Formal Parameter Table, the External Symbol Table, which contains all names loaded as entry points will be displayed. Then the parameter selected will be set to the value of the variable selected.

2. SETNEGVAL - Set to Negative Value

Sets a Formal Parameter to the negative of the current value of a variable. It is exactly like SETVALUE, except that the value retrieved is negated. This operation is useful in the construction of symmetric images. To sequence SETNEGVAL, first select SETNEGVAL from the Control Table, select the Formal Parameter Table, and finally select the parameter variable from the Symbol Table.

3. SETBNAME - Set by Name

This operation sets a formal parameter to "track" a variable (e.g., for a SCALE factor to track dial A). Also SETBNAME should be used to assign the arguments of such DSPLY items as JSR, IMG, LABL, JMP, CJMP, WJMP

and all Table operations. This includes all Image Items whose arguments deliver values which are Addresses of core locations. These are all described in the DSPLY/PRM with argument mnemonics beginning with a letter "L".

To sequence through the SETBNAME operation, select SETBNAME from the Control Table, then select from the Formal Parameter Table the name of the Image Argument which will track the selected variable name, then select from the displayed External Symbol Table the name of the variable to be tracked. The operation will then complete itself.

The two names picked are put into a name pair that will appear whenever the Formal Parameter Table is again displayed. This name pair will be broken if the Formal Parameter is later reassigned.

Thus, for the example above, the displayed Formal Parameter name will be changed from SCALE to "SCALE VCDA" and any changes in the position of Variable Control Dial A will change the scale of subsequent affected Image Items.

4. HOLD

Holds the value of a parameter at its current value, making the assignment static. This is useful in stopping a Formal Parameter from tracking a variable. The operation is equivalent to doing a SETVALUE of a variable to a Formal Parameter. With HOLD, however, it is necessary only to select operation HOLD and then the Formal Parameter to be held.

If the argument of the Formal Parameter had last been set by a SETN command, i.e., was dynamic, then the external name appended to the Formal Parameter name will be deleted. Thus, a HOLD of "SCALE VCDA" will change the Formal Parameter Table entry displayed to "SCALE."

5. MERGE - Merge Formal Parameter References

Causes all references to the first Formal Parameter to use the second Formal Parameter. The first Formal Parameter is dropped from the Formal Parameter Table. An example of the use of this operation would be to GET (MOVE) and GET (DRAW), then MERGE both Y parameters. The line drawn would then always be horizontal because the Y values would be "merged" and would from that point on, always have the same values.

The sequence for this operation is to select the MERGE control operation, then a formal parameter. Select a second Formal Parameter and the first will be merged onto it. The first parameter selected will be erased from the Formal Parameter Table.

6. RENAME - Rename a Formal Parameter

Select operation RENAME, then the Formal Parameter to be renamed, then input a name (up to 10 characters & terminated by a "control C") on the console input unit. The name of the selected Formal Parameter will be changed to the one just typed.

7. CLOSE - Eliminate a Formal Parameter

CLOSE removes a selected Formal Parameter from the Formal Parameter Table, and sets all references to the value of the parameter or to the name of the parameter, depending on whether a SETVALUE or a SETBNAME was last applied (i.e., a rotation angle, AX, can be permanently left at its current value, or can permanently reference a dial, for example). To sequence this operation, select CLOSE from the Control Table, then select the Formal Parameter to be CLOSE'd, and the operation will take place.

C. I/O AND COMMUNICATION CONTROL OPERATIONS

1. INPSTRING - Input String

Select INPSTRING from the Control Table, then select from the Formal Parameter Table the variable which is to address the input character string, then type in the string terminated by a typed control-C. The string will be stored into constant storage in LCG or DSPLY ASCII format. This is useful for inputting LABL arguments.

2. DELETE - Delete from a Table

Select DELETE from the Control Table, then select from the list of Tables the name of the table from which an item is to be deleted. Finally, select the item from the chosen table. The item selected will be deleted from the table. If the list of Tables is the one selected above, then a selection of an item from it will not only delete that table name from the list of Tables, but will also delete the entire table itself.

3. MOVTBLE - Move Table Entry

Select MOVTBLE from the Control Table, then select from the list of Tables the two tables to be involved in the moving operation. Select from the first table involved all items to be copied onto the second table. As each selection is made, the table entry is copied from the one table to the other.

4. ADDENTRY - Add an Entry to a Specified Table

This command enables the operator to add new entries to any table. The operator will be shown a list of all tables. He then selects the table to which he wishes to append the new entry, and then types in the name he wishes to give the entry. If the operator should select the "table of tables" as the table to which the appendment is to be made, then he has in fact created an entirely new table, to which other table entries can be moved. For example, the user can create a table of all the control dial variables, giving it the name "DIAL TBL" and then selecting a MVTE operation to move VCDA through VCDF into "DIAL TBL."

D. MACRO FACILITIES

1. MACRO - Start a Macro Definition

Selection of MACRO from the Control Table will put BUILD into the macro mode. All selections made by the user will be recorded until ENDM is selected to end the macro definition mode. When MACRO is selected, the TTY is enabled for the user to typein the name of the macro sequence to be constructed. This macro will be a recording of the BUILD action sequences selected following MACRO.

While in the MACRO definition mode, the operator, as before, will be making selections from tables in order to step BUILD action sequences. If the operator makes a selection from some menu, then that selection is recorded, such that when the macro is later invoked as an operation, the same selection will be automatically accepted with no user choice or intervention. However, if in the Macro definition, the operator wishes the choice he makes now not be automatic, but to be only a dummy, later to be replaced by a selection made and accepted at Macro invoke time, he may so specify by positioning the blinker and depressing the pen stylus over one of two control areas (H, Q) in the menu display. (These two special control areas are displayed only at Macro definition time. The remaining control areas are always displayed when a menu selection

is to be made). Selecting the Q(Quit) menu-control tells the Macro processor not to record the menu selection made now, so that when this Macro is later invoked the user will again have to choose selection from this menu. Selecting the "H" menu-control will do everything the first control will do, but in addition will ask for a comment to be typed in. This comment is recorded. When the macro is later invoked, the comment input during Macro definition time will be displayed in italics at the top of the screen. Thus, the operator defining new Macro is able to give instructions to the operator who will be using the macros.

During Macro definition, if the operator decides to select an item from a different menu than that originally presented, he may do so as described in section IV. The Macro processor will record the new table chosen for display and selection, and when the macro is later invoked only the menu finally chosen for display will be displayed.

This feature has many obvious uses. For example, a GET command will ordinarily display a menu of all display segments in the Temporary Library and wait for a selection to be made. If the person defining a macro, instead wants the macro user to GET only items moved into some "PARTS" list, then when he selects MACRO, followed by operation GET, he can select the menu display control T to pick another table, choose the "PARTS" table, then select the special Macro menu display control H to input a message. Typing in "GET ITEM FROM PARTS LIST," will be recorded. When this macro is later invoked, at this point, the "PARTS" menu will be displayed for selection, and the above message string will be displayed in Italics.

2. ENDM - End Macro Definition

Selection of ENDM from the Control Table will halt the macro definition and enter the macro name in the Control Table. The action string is moved into the temporary library, and this new control may now be selected like any other predefined BUILD operation.

3. DUMPM - Dump Macros

This operation copies MACRO strings (BUILD operation definitions) from the Macro-Action Library to the External System Library. The user is asked to input a name for this collection of action strings. He is then shown the menu of all current action strings (the Control Table) so that he may select the desired operations to be DUMPed. Each user selection is recorded; when the user signifies (via pedal or FNS) that the list is complete, the entire collection of operations is dumped into the External System Library under the file title input by the user.

4. LOADM - Load Macros

This operation loads into the Macro-Action Storage a collection of Macro definitions (BUILD operations) which was previously dumped into the External System Library via a DUMPM operation. The user must input the name of the collection to be retrieved. Then the Macro strings are loaded into the Macro-action storage, and the Macro names are appended to the list of available BUILD operations (Control Table).

E. OTHER CONTROL OPERATIONS

1. HIDE - Suppress Display

This operation permits the user to suppress the normal display of either the menu, or the current Work Storage Image Segment, or both. When the menu display is suppressed, a small U is displayed in the lower left-hand corner of the screen; selecting it will re-instate display of menus.

After selecting HIDE from the Control Table, selecting from anywhere in the center of the screen will suppress the normal Work Storage display, and selecting the menu Table Name will suppress any further display of the Control Table.

2. UNHIDE - Stop Display Suppression

This operation is used to re-activate normal display of the current Image Segment (Work Storage) after a HIDE operation.

APPENDIX A

Predefined Image-Items (Initial Contents of Temporary Library)

Item Name	Names of Arguments	Item Function
VISUAL ELEMENT GENERATION:		
MOVE	FZ, FX, FY	Blanked vector to end-pt. coor-ds.
DRAW	FZ, FX, FY	Displayed vector to end-pt. coor-ds.
LABL	LTEXT	String of text characters
2DTBL	LTABLE	Table of packed 2D vectors
CONTROL EFFECTING:		
IMG	LIMG	Sub-image call
JSR	LSUBR	Sub-program call
NUL	LARG	Null (for passing arguments to S.R.'s)
RET		Sub-image return exit (restores transf. & count)
SAVT		Open-brackett for transform & loop range
REST		Close-brackett for range of transform & loop
LOOP	ICNT	Repeat start
ENDL		Repeat end (cannot be nested w/o SAUT-REST)
JMP	L	Jump in image
CJMP'PEN	L	Conditional jump in image (on Pen)
CJMP'F1	L	Conditional jump in image (on full-wd)
CJMP'F2	L	Conditional jump in image (on rt-half-wd)
LDSN1	LVAL	Load signs to flags
LDLS1	LVAL	Load "=0" test to flags
LDMB	B	Load all mode bits
ORMB	B	Set selected mode bits
ANDMB	BMSK	Reset un-selected mode bits

TRANSFORM SPECIFYING:

SCL	FSCAL	Scale down from current size
ROTX	AX	Rotate about local X-axis by AX
ROTY	AY	Rotate about local Y-axis by AY
ROTZ	AZ	Rotate about local Z-axis by AZ
RXYZ	AZ, AX, AY	Rotate about local X, Y, AX, then about resulting Y by AY, then about final Z axis by AZ.
DX	FX	Displace along local X from current position
DY	FY	Displace along local Y from current position
DZ	FZ	Displace along local Z from current position
DV	FZ, FX, FY	Displace following image from current position along local X, Y, Z axii by (FX FY FZ).
LDSCL	FSCAL	Scale subsequent image seg. by FSCAL
LDRX	AX	Instate rotation by AX about CRT-X axis
LDRY	AY	Instate rotation by AY about CRT-Y axis
LDRZ	AZ	Instate rotation by AZ about CRT-Z axis
LDRV	AZ, AX, AY	Instate successive X, Y, Z rotations from CRT frame.
LDX	FDX	Reset local X displacement to FDX from CRT center
LDY	FDY	Displace FDY from CRT center along local Y axis
LDZ	FDZ	Displace FDZ from CRT center along local Z axis
LDV	FDZ, FDX, FDY	Displace from CRT center by $\overline{FD}$ along local axis.
LDI		Instate CRT reference frame.

APPENDIX B

Pre-defined Operations

Operations	Additional User Inputs Required Select from Tables: - TTY Input
CLEAR	-----
GET	Temporary Library Table
SKETCH	-----
SKBAND	-----
PUT	Input a Name
SETBNAME	External Symbol Table Formal Parameter Table
SETVALUE	External Symbol Table Formal Parameter Table
SETNEGVAL	External Symbol Table Formal Parameter Table
HOLD	Formal Parameter Table
MERGE	Formal Parameter Table Formal Parameter Table
RENAME	Formal Parameter Table Input a Name
CLOSE	Formal Parameter Table
INPSTRING	Formal Parameter Table Input a String on TTY
DELETE	Table of Tables Entry from Selected Table
MOVETE	Table of Tables Table of Tables Any entry(s) from first table selected

APPENDIX B (Continued)

Operations	Additional User Inputs Required Select from Tables: - TTY Input
ADDENTRY	Table of Tables Input a Name
MACRO	Input a Name
ENDM	-----
DUMPM	Entry from Control Table Input a Name
LOADM	Input a Name
HIDE	-----

ABSTRACT

CDRDR is a set of relocatable routines in the AMOS library which are used to read punched cards on the CDR-P1. It contains the following subroutine entry points:

RDCB	Read Binary Card(s)
RDCH	Read Hollerith Card(s)

CDRDR contains no external references, and so is independent of all other programs.

REFERENCE DATA

CDRDR is a set of relocatable routines in the AMOS library which are used to read either binary or Hollerith cards. It does not depend on any external references. CDRDR contains the following entry points:

A. RDCB-Read Binary Card(s)

This subroutine reads as many cards on the CDR-P1 as is necessary to fill the TABLE of specified LENGTH. (One card will fill a table of 40_8 words). Each column on the cards is interpreted as two 6-bit binary characters.

Calling Sequence:

```
JPSR    RDCB
        TABLE   °Address of lst word of table
        LENGTH  °Length of table
        ERROR   °Error instruction
        DONE    °Instruction to be executed when done
        .       °Foreground program
```

B. RDCH-Read Hollerith Card(s)

This subroutine reads as many cards on the CDR-P1 as is necessary to fill the TABLE of specified LENGTH with characters packed 5 per word. (One card will fill 20_8 words. Each column on the cards is interpreted as one character.) Before the characters are stored, they are converted from the Burroughs 6-bit code to standard AMOS 6-bit internal code. If more than one card is read in, a carriage return character is packed into the table to indicate an end of card.

Calling Sequence:

```
JPSR    RDCH
        TABLE    °Address of lst word in table
        LENGTH   °Length of table
        ERROR    °Error instruction
        DONE     °Done instruction
        .        °Foreground program
```

If the reader is in the "not ready" state as indicated by S4[7], the "start pivot" (STPVT) is set to return to a location in the program, and the foreground program is executed, in anticipation of the operator's turning on the reader. CDRDR may not be re-entered from RDCH or RDCB until STPVT and BUSY have been cleared.

In case of an error, the card reader is turned off and an ERROR instruction is executed.

RDCB and RDCH are "open" subroutines, that is, a foreground program may be in operation while the cards are being read, which may be interrupted by the card reader to process characters and end-card pivots. The contents of all registers will be restored when CDRDR finishes, and control returned to the foreground program.

While the card reader is being used, location BUSY is negative, indicating that it should not be called again until it has finished. If it is called again before it has finished, RDCB will wait until BUSY has been cleared by the character processor before proceeding to its next task. In this case, the foreground program being executed is the waiting for location BUSY to be cleared.

When CDRDR has finished filling its TABLE, the card reader is turned off and the instruction DONE is executed.

C. CON-Convert from Burroughs to AMOS 6-bit Code

This subroutine converts the Burroughs 6-bit character in BR[24-29] to the corresponding AMOS 6-bit character and leaves it in BR[24-29].

Calling Sequence:
°BR[24-29]=Burroughs Character

```
JPSR    CON
        .        °Returns to next location
                 °BR[24-29] = AMOS character
```

D. OFF-Turn Off Card Reader

This subroutine is called to turn off the card reader. It sets the character, end-card and error-end-card pivots to point to a dummy subroutine, clears the BUSY flag, and turns off IC control bits for the card reader IC[6-7].

Calling Sequence:

```
JPSR    OFF
        °Returns to next location
```

GENERAL

The AMOS routine COPVR may be used to copy specified files on an input tape to be appended to an output tape.

OPERATION

The statement input to the AMOS Monitor includes (optional) the means to specify that an attempt to verify the copied files against the input files be made. The COPVR calling sequence is as follows:

COPVR (FIRST, LAST, INTAPE, OUTTAPE, MODE, DENS1, DENS2)!

where:

FIRST	= First input file no.
LAST	= Last input file no.
INTAPE	= Input tape unit no.
OUTTAPE	= Output tape unit no.
MODE	= Ø copy only, no verify
	= 1 copy and verify, with minimal typeout (i.e., "verify" or "no verify")
	= 2 copy and verify, with typeout (i.e., "verify" or "record, file, no. words, no. words failing to verify, contents of first input word failing, contents of first output word failing")
DENS1	= Input tape density (= Ø, 1, or 2)
DENS2	= Output tape density (= Ø, 1, or 2)

The verify subroutine VRFY in COPVR may be called separately under AMOS to attempt to verify a specified number of files from specified first files on specified input and output units. The VRFY calling sequence is as follows:

VRFY (FI, FO, IN, OUT, MOD, NOF, DI, DO)!

where:

FI = First input file no.
FO = First output file no.
IN = Input tape unit no.
OUT = Output tape unit no.
NOF = No. files to be verified
MODE = 1 output corresponds to that in copy call
2
DI = Input tape density (= Ø, 1, or 2)
DO = Output tape density (= Ø, 1, or 2)

HARDWARE REQUIREMENTS

1. Two tape drives compatible with following required software items.

2. Enough available core (after program loading) to hold twice the length of the longest record to be checked.

SOFTWARE REQUIREMENTS

AMRM, MTAC and RDHDR versions appropriate for the hardware configuration used.

ABSTRACT

CRDTT is a utility routine in the AMOS library which is used to read in a batch of text files from cards on the CDR-P1, and output them as successive ATEXT files on the specified mag tape unit. It links with CDRDR and mag tape routines within the AMOS Monitor. It is independent of system version.

REFERENCE DATA

This subroutine is called internally by the calling sequence:

```
JPSR  CRDTT
      UNIT        °Unit No. =0,1,2, or 3
      .           °Next instruction
```

It reads Hollerith cards on the CRD-P1 and outputs them at the end of the specified tape (unit) as a series of files which may subsequently be edited by the AMOS text editor or translated by ASMT, ADEPT, or FORTRAN.

The Deck Format of the cards input must be as follows:

```
Column:  1 2          6-10      16-20
         * *          TITLA     MODE
         Page 1
         *
         Page 2
         *
         .
         .
         .
         *
         Page N
         * *          TITLB     MODE
         Page 1
         *
         .
         .
         .
         *
```

```
Page M
**          TITLC     MODE
.
.
.
*           TITLZ     MODE
.
.
.
* *
* *
```

where:

TITLA,... TITLZ are the titles to be given to the output files for each program.

MODE indicates in which mode the characters are to be packed.

If MODE=FORTN, no tabs will be inserted for spaces. This is necessary for the proper compilation of AFORT programs.

Otherwise tab characters will be inserted to replace strings of blanks ending at the specified tab stops, which are set at 10_{10}, 22_{10}, and 34_{10}. Trailing spaces will be deleted.

NOTE

The tab stops and the maximum number of input columns may be changed. See the maintenance manual.

CRDTT will write the current page and start a new one if either the buffer length is exceeded or the maximum number of lines is exceeded.

NOTE

This parameter may be changed; it is specified by entry point LINES.

As CRDTT completes each separate file it will type TITLE: FILE NO. N. where N is the file number of the output ATEXT file.

If an illegal character is encountered on a card, a ? character will replace it in the text.

When CRDTT has completed processing the deck, it will write a terminating file mark on the output tape, and rewind the tape.

CRDTT must link to CDRDR, the AMOS library card-reader routines, as well as to AMRMS.

GENERAL

DBUG is a relocatable AMOS library file which contains programs that may be used for debugging purposes. They are loaded when requested by a statement input to AMRMX. If symbolic typeout MODE is requested, file DPS will also be loaded from the library.

MONITOR STATEMENTS

OPEN (ADDR, "MODE")!

OPEN causes the value of expression ADDR to be typed in mode OCTAL and followed by a colon. Then, the current contents of memory location ADDR are listed in the mode MODE. The memory cell at location ADDR is now "open." As long as a cell is left "open," further information concerning it may be requested by typing "@", "=" or "/" (see below), or information concerning it may be changed by typing:

NAMES VALUE CLOSE-CHARACTER

In the above, the inputs NAMES and/or VALUE may be omitted. If given, the input NAMES may be any number of undefined alphanumeric symbols, ten or fewer characters in length, with each symbol followed by a colon (:). Each such symbol is thereby defined in the current Monitor symbol list as a name of the "open" location ADDR. The expression VALUE, if given, is evaluated and its value replaces the contents of location ADDR when it is "closed." The CLOSE-CHARACTER, (comma, backspace or ←, semicolon, or carriage return) enters the new value (if any) into the "open" cell (location ADDR) "closing" the cell, then does one of the following:

Close-Character	Action
Comma	Opens the next cell in memory.
Backspace or ←	Opens the previous cell in memory.
Semicolon	Opens the cell addressed by the last open cell.
Carriage Return	Does not open any further cells, returns to Monitor control statement input.

While a cell is open, typing

@	Lists the nearest symbolic expression defining the location of the open cell.
=	Lists the contents of the open cell in any modes other than MODE.
/	Lists the contents of the location addressed by last listed cell in the mode MODE.
*	Causes all input for expression VALUE to be dropped and ignored so that immediate closing would not alter the contents of the open cell.
° or [	Causes any comment (characters) up to the next ° or [to be ignored.
#	Closes the current open cell without altering its contents, and then opens the cell with address VALUE.
$	Stores VALUE in "open" cell.

If no MODE is given, OPEN! uses that set by the last mode-setting statement.

SRCHW (WORD, MASK, FIRST, LAST, "MODE")!

SRCHW will list, in formats MODE, all memory words from location FIRST through location LAST whose contents masked by MASK equal the value of WORD. If no FIRST or LAST address is given that of the last bound-setting statement is taken. If no MODE is given that of the last mode-setting statement is taken. SRCHW is a bound-setting and a mode-setting statement. Note that if no MASK is given, it will be an "omitted argument" and represented by -0, so that the search will be for exact matching of the value of WORD.

SRCHA (ADDR, FIRST, LAST, "MODE")!

SRCHA -- the behavior of SRCHA (A, F, L, M)! is identical to that of a control statement SRCHW (A, 77777, F, L, M)!.

TRAPS (FIRST, LAST)!

TRAPS causes memory locations FIRST through LAST to be filled with "trap" instructions. If no FIRST or LAST address is given, that of the last bound-setting statement is taken.

SNAP (FIRST, LAST, "MODE," (REGISTER-LIST))!

SNAP restores registers and pivots to values saved in current PANEL, except where changed by new values in REGISTER-LIST, then starts execution at location FIRST. When execution reaches location LAST, or PULSE 1 (manual interrupt) switch is depressed, execution is suspended. The location at which execution was suspended is then typed in octal, followed by a listing of the resulting register values in formats MODE. The resulting register and pivot contents are then enstated as the current PANEL. If no FIRST or LAST address is given, that of the last bound-setting statement is taken. If no MODE is given, that of the last mode-setting statement is taken. SNAP is a bound-setting statement.

INTRODUCTION

This manual is intended as a reference manual for DISP, the AMOS EDIT Display Package. Certain subroutines contained in DISP can be used by other user programs; however, it should be noted that many DISP routines contain external references into EDIT and FONT, the EDIT character coordinate lists, and that these programs (or user furnished equivalents) would be loaded into core if DISP routines were called elsewhere.

For further information, the user is advised to consult the following documents:

1. DISP Software Maintenance Manual
2. DISP Program Listing
3. DISP Software Operating Instructions
4. FONT Programmer's Reference Manual

CONFIGURATION REQUIRED

The DISP source program is written with conditional assembly statements to cause the assembly for any of the following configurations:

Version 1:	OSD-1 with DAC/CMP-2 pair loaded from destination $D14_8$.
Version 2:	OSD-1 with DAC/CMP-2 pair loaded from destination D7.
Version 3:	OSD-2, CTO with DAC/ACE pairs in destinations $D10_8$, $D11_8$, and $D12_8$ and with CTO distributor in destination D6.
Version 4:	AGT/10
Version 5:	AGT/30

DISPLAY ROUTINES

A. DCHAR - Display Character Subroutine

DCHAR is called to display a character in a given position on the oscilloscope screen. The calling program must first set up the character position in location $DCH05 with X/2 in the upper half and Y/2 in the lower half and must load the

ambilogical configuration register, D5, with the appropriate control word (4ØØØ1! H ØØØØ1 for versions 1 and 2, and 44Ø6Ø! H 1ØØØØ for version 3). Control is returned to the calling program (see calling sequence below) after the character has been displayed in versions 1 and 2 and after the vector generator has been started in versions 3, 4, and 5. The calling program has the responsibility of turning off the ambilogical control register (versions 1, 2, and 3 only).

Calling Sequence:

DCHØ5:	Character coordinates (X/2!H Y/2)
AR:	6-bit AMOS character code (bits 24-29)
L:	JPSR $DCHAR
L+1:	Returns here if normal character
L+2:	Returns here if backspace (77_8)
L+3:	Returns here if C/R (15_8)
L+4:	Returns here if tab (11_8)
L+5:	Returns here if SPACE ($4Ø_8$)

B. DTBL - Display Special Character Subroutine

DTBL is called to display a special character not in the standard AMOS character set (normally EDIT calls DTBL to display the characters "down arrow", "right arrow", and "page mark"). On entry, a fetch instruction addressing the desired coordinate list for the special character is in the AR register. The FONT Maintenance Manual should be consulted for the fetch instruction and coordinate list structures. As in DCHAR, it is the responsibility of the calling program to place the character coordinates in DCHØ5 and to turn on and off the ambilogical control register (versions 1, 2, and 3 only).

Calling Sequence:

AR:	Special fetch instruction
L:	JPSR $DTBL
L+1:	Returns here

C. LNDIS - Display Line Subroutine

LNDIS is called to display a text line on the display scope. The arguments to LNDIS include the X and Y origin coordinates for the beginning of the line and the address - 1 of the packed text line. The characters are packed five per word and terminated by a carriage return character (15_8) or BACKSPACE (77_8). LNDIS handles all necessary loading of the ambilogical control destination (D5).

Calling Sequence:

AR:	Origin of line coordinates (X/2!H Y/2)	
L:	JPSR	$LNDIS
L+1:	Address - 1 of first word in line	
L+2:	Returns here after C/R (15_8)	
L+3:	Returns here after BACKSPACE (77_8)	

D. BMSET - Set Oscilloscope Beam Subroutine

BMSET is called to load the ambilogical control register and move the scope beam to a given position for versions 1 and 2, and to load the ambilogical control register for version 3. Also, the first time the version 3 BMSET is entered, the Coordinate Transform Operator array is loaded with appropriate coefficients and the end-of-list interrupt pivot is initialized. In versions 4 and 5, the array is loaded and the vector generator enabled.

Calling Sequence:

AR:	Beam coordinate (X/2!H Y/2)	(versions 1 and 2 only)
L:	JPSR	$BMSET
L+1:	Returns here	

DISPLAY PARAMETER SETTING

A. SCALE - Set Parameters Subroutine

SCALE is the subroutine entered to set the desired character coordinate scaling, character and line spacing, tab settings, and the standard number of lines displayed before and after the current line in EDIT. This subroutine is normally called from the AMOS Monitor by an AMOS control statement (see below). Any of the functions performed by SCALE may be omitted by making the appropriate argument null (-Ø). In addition, certain tab increments can be left undisturbed by a zero argument.

Calling Sequence:

L:	JPSR	$SCALE
L+1:	Address - 1 of a list of five tab spacing values	
L+2:	No. lines displayed before and after the EDIT "current line"	
L+3:	Character spacing !H line spacing	
L+4:	Character scale	
L+5:	Ø	
L+6:	Returns here	

AMOS Monitor Statement:

SCALE (T1, T2, T3, T4, T5), LINES, XINC!H YINC, SCALE)!

B. TBSET - Compute Tab Coordinates Subroutine

TBSET is called by SCALE to compute the coordinate values of the tab settings according to the current tab count values and the character spacing value. The tab coordinates computed are used in LNDIS for setting the X coordinate when a tab character is scanned in the text line.

Calling Sequence:

```
L:      JPSR            $TBSET
L+1:    Returns here
```

C. DCH2Ø - Scale Characters Subroutine

DCH2Ø is called by SCALE to scale the character coordinate lists in FONT according to the given character scale size (versions 1, 2, and 3 only).

Calling Sequence:

```
AR:     Scaling factor (X*4!H Y*4)
L:      JPSR            $DCH2Ø
L+1:    Returns here
```

D. DCH27 - Digital Multiply Subroutine

DCH27 is called by TBSET and DCH2Ø to provide digital multiplication. The result is returned in the AR register shifted two bits right (i.e., RESULT - A * B / 4).

Calling Sequence:

```
AR:     Multiplicand
BR:     Multiplier
L:      JPSR            $DCH27
L+1:    Returns here with:
AR:     Product/4
```

GENERAL

DPS is an AMOS relocatable file which contains the routines (formerly included in AMRMX) for disassembly of machine words for symbolic typeouts, and symbolic typeout of addresses. The entry points provided are DPS and DPAD.

USE

When MODE "S" is required by AMRMX, it will load DPS and then call it to make symbolic typeouts.

INTRODUCTION

The DSKIO is a package of subroutines that drive the DMS2 Disk Memory Subsystem. Included in this package are subroutines and arguments for reading and writing data, formating tracks, track seeking, and requesting status information and current head position. Two versions are provided for single or multiple disk unit systems.

Automatic cylinder (track) seeking and stepping between successive cylinders are provided in the I/O operations initialized by the DSKIO calls.

DISK DESCRIPTION

DSKIO, Version 2, is capable of driving the four-disk configuration which is the maximum disk system. Each unit consists of two removable disk packs; each pack containing two recording surfaces which have 203_{10} concentric tracks. The term "cylinder" is used to describe the two recording surfaces of a track. There are eight 104_{10} word sector blocks on each track (16_{10} per cylinder), therefore, disk storage capacity is as follows:

Words/cylinder = $1,664_{10}$

Words/pack = $337,792_{10}$

Words/unit = $675,584_{10}$

Words/four units = $2,702,336_{10}$

Version 1 of DSKIO drives a single unit system. The core requirements of the DSKIO versions are as follows:

Version 1 - 1020_8 locations

Version 2 - 1300_8 locations

HARDWARE CONFIGURATION

DSKIO requires the following hardware configuration:

1. Disk Memory Subsystem, DMS2-(P1 for Version 1, P2 for Version 2).
2. Extended Arithmetic Unit, EAU1.
3. Priority Interrupt Instruction, "JPSR'I 77743." This instruction is executed by a programmatic interrupt of a lower priority than DMS2 when called by special instruction C27.

For timing information on DMS2 operations, refer to the DMS2 Subsystem Programming Instruction Manual.

DSKIO SUBROUTINE CALLS

A. Read and Write Subroutines

READ

The read subroutine reads data from disk into core memory. A read operation starts at the beginning of a specified sector and continues until the desired number of words have been transferred. The disk heads will be positioned at the desired cylinder before the read operation is started. Automatic stepping through sequential cylinders will be done by this routine if necessary.

Calling sequence:

```
JPSR    $8RD      Call
        BUSY      Returns here if unit busy
        DISK      Unit (27-28), Pack (29)
        SECT      Cylinder (18-25), Track (26),
                    Sector (27-29)
        #WRDS     Number words to be transferred
        ADDR      First core address for data
        ERR       Instruction executed if error
        DONE      Instruction executed when completed
          .       Returns here after operation started
```

WRITE

This call writes data from core memory onto disk. A write operation starts at the beginning of a specified sector and continues until the desired number of words have been transferred. Zero words are written to pad out the last sector if the output data will not completely fill it. The disk heads will be positioned to the desired cylinder before the write operation is started. Automatic stepping through sequential cylinders will be done by this routine if required. Automatic read checking after writing may be specified by having the flag $8RCF set to -0 before initiating the write call. In this mode, DSKIO will read the data written on the disk after the write operation and check for any errors. If the automatic read check after write mode is not desired, the flag $8RCF should be set to +0 before the call.

Calling sequence:

```
JPSR    $8WR     Call
        BUSY     Returns here if unit busy
        DISK     Unit (27-28), Pack (29)
        SECT     Cylinder (18-25), Track (26),
                     Sector (27-29)
        #WRDS    Number words to be transferred
        ADDR     First core address for transfer
        ERR      Instruction executed if error
        DONE     Instruction executed when completed
          .      Returns here after operation started
```

READ CHECK

This subroutine reads data from the disk and checks for any data error. No data is read into core memory by this operation. The disk heads will be positioned at the desired cylinder before the check operation is started. Automatic stepping through sequential cylinders will be done by this routine if necessary.

Calling sequence:

```
JPSR    $8RC     Call
        BUSY     Return here if unit busy
        DISK     Unit (27-28), Pack (29)
```

SECT	Cylinder (18-25), Track (26), Sector (27-29)
#WRDS	No. words to be checked
0	Address argument (ignored)
ERR	Instruction executed if error
DONE	Instruction executed when completed
.	Returns here when operation started

B. Write Format Subroutine

WRITE FORMAT

The write format subroutine is called to initialize a selected cylinder. This operation writes the sector addresses and clears the sector data areas. This operation is required before writing data on any new disk packs. The disk heads will be positioned to the desired cylinder before the write format operation is started.

Calling sequence:

JPSR	$8WF	Call
	BUSY	Returns here if unit busy
	DISK	Unit (27-28), Pack (29)
	CYL	Cylinder (18-25)
	ERR	Instruction executed if error
	DONE	Instruction executed when completed
	.	Returns here when operation started

C. Track Seeking Subroutines

SEEK

The seek subroutine positions the disk heads on a selected unit to a particular cylinder. Although the read and write commands will position to the specified cylinder prior to performing the operation, the seek subroutine is provided for positioning the heads before requesting input/output to allow instruction execution overlap.

Calling sequence:

JPSR	$8SK	Call
	BUSY	Returns here if unit busy
	UNIT	Unit (27-28)
	CYL	Cylinder (18-25)
	ERR	Instruction executed if error
	DONE	Instruction executed when completed
	·	Returns here when operation started

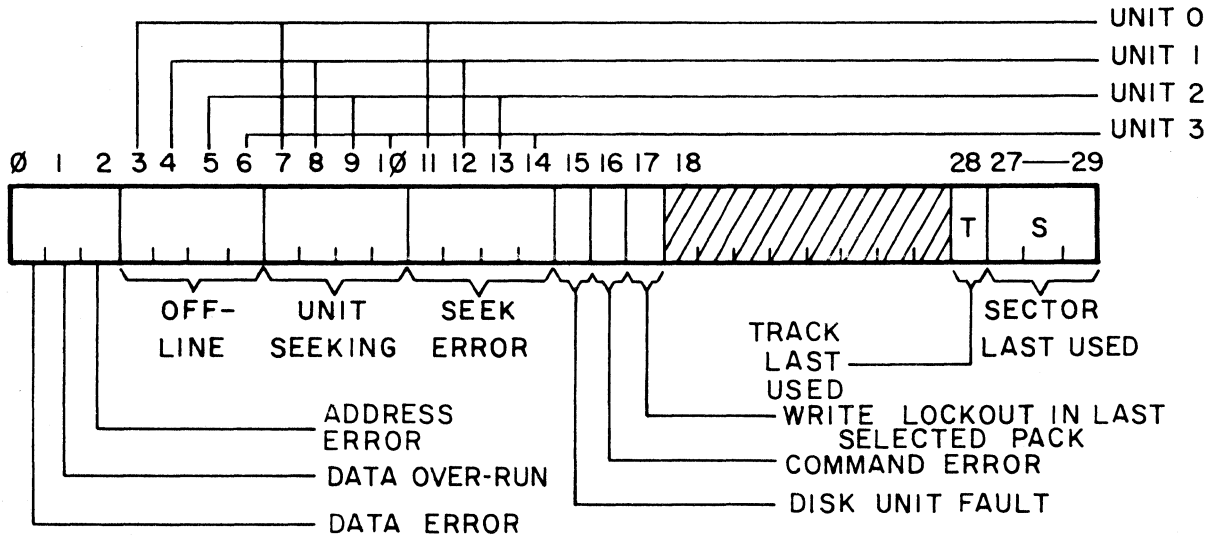
D. Status and Positions Subroutine

STATUS

The status subroutine may be called to read the current DMS2 status word. This routine will wait until the DMS2 is not busy, then read the status word and store it into the calling program. The subroutine may not be called on a priority level equal to or higher than the DMS2. The "DISK" argument (see below) is needed for proper pack selection for write lockout and unit fault indications. The figure below illustrates the standard status word format for the DMS2.

Calling sequence:

JPSR	$8ST	Call
	DISK	Unit (27-28), Pack (29)
	ADDR	Address where status word is to be stored
	·	Returns here



POSITION

The position subroutine is called to request the current cylinder position of a disk unit. The current cylinder position will be returned or an indication will be made that the position of the unit is undefined (DSKIO has not used that unit since being loaded).

Calling sequence:

```
JPSR    $8PO    Call
        BUSY    Returns here if unit busy
        UNIT    Unit (27-28)
        ADDR    Addr. where position is to be stored
                  (will store -0 if position undefined)
          .     Returns here
```

RESET STATUS

The reset status subroutine is called to reset software flags and status words about the disk units. Each active unit is designated non-busy and the current position is set undefined. No hardware disk operation is initiated by this subroutine. The effective operation caused by this subroutine is to reset the information kept by DSKIO about the units. This subroutine should be called in the event any disk operations were interrupted by system power down, etc.

Calling sequence:

```
JPSR    $8RS    Call
          .     Returns here
```

ARGUMENTS FOR SUBROUTINE CALLS

BUSY

Control is returned to this point in the calling sequence if conditions are such that the requested operation cannot be started on the selected unit. Control will be returned on the same priority level as that in effect at the time of the subroutine call. If the calling program desires to wait until the unit is free, the instruction located at BUSY might be "JUMP .-1."

DISK

This argument gives the selected disk pack with the disk unit number in bits 27-28 and the pack number in bit 29.

SECT

This argument gives the selected sector with the cylinder number ($0-203_{10}$) in bits 18-25, the track number (0 or 1) in bit 26, and the sector number (0-7) in bits 27-29.

#WRDS

This argument gives the number of words to be transferred in a read or write operation.

ADDR

This argument gives the first core address used in a read or write operation, or storage address for status or position information.

CYL

This argument gives the selected cylinder number ($0-203_{10}$) in bits 18-25.

UNIT

This argument gives the selected unit number (0-3) in bits 27-28.

DONE

This instruction is executed when the specified disk operation has been completed without error on the programmatic interrupt priority level. It must be a single instruction which does not modify the Location Counter (LC), or may be a subroutine jump to a user subroutine which returns to the following location. The content of the AR is set to zero before the execution of the DONE instruction.

ERR

This instruction is executed on the programmatic interrupt priority level if an error condition is found in trying to perform the specified disk operation.

It must be a single instruction which does not modify the Location Counter (LC), or may be a subroutine jump to a user subroutine which returns to the following location. The AR is loaded with the error code prior to executing the ERR instruction. The error code bits are as follows:

AR(0) = 0
AR(1) = 1 } Available for setting sign flags
AR(2) = Unit off-line
AR(3) = Read redundancy error after 3 re-tries
AR(4) = Sector address error
AR(5) = Cylinder address error
AR(6) = Write lockout
AR(7) = Hardware malfunction error. This error should not occur in normal operation. Bits (27-29) of AR give the particular error condition as follows:

$AR(27-29)_8$

1 Seek error condition when executing home track seek operation.

2 Command error when normal error interrupt occurs. This condition could occur if the disk unit power is dropped while an operation is in progress or if the write protect switch were turned on during a write or write format operation.

3 Error terminate interrupt occurred and no error bits on in the status word.

4 Count error (i.e., no words = 0).

5 "Fault" condition is disk unit.

SUMMARY OF DSKIO CALLS

A. Read and Write Subroutines

READ	$8RD (BUSY, DISK, SECT, #WRDS, ADDR, ERR, DONE)
WRITE	$8WR (BUSY, DISK, SECT, #WRDS, ADDR, ERR, DONE)
READ CHECK	$8RC (BUSY, DISK, SECT, #WRDS, 0, ERR, DONE)

B. Write Format Subroutine

WRITE FORMAT	$8WF (BUSY, DISK, CYL, ERR, DONE)

C. Track Seeking Subroutine

SEEK	$8SK (BUSY, UNIT, CYL, ERR, DONE)

D. Status and Position Subroutines

STATUS	$8ST(ADDR)
POSITION	$8PO(BUSY, UNIT, ADDR)
RESET	$8RS

GENERAL

DSKL is an ADEPT language program used to create a binary monitor on disk from the ADEPT relocatable output of AMRMX, Version 7. Versions of DSKL are provided to load the relocatable file from disk or magnetic tape.

HARDWARE REQUIREMENTS

Version 1 - AGT/10, M10-P1 or AGT/30 or AGT/50 and MTP5 or 8-P1 or MTP7-P1, DDC1-P1 and DMS2-P1

Version 2 - AGT/10, M10-P1 or AGT/30 or AGT/50 and DMS2-P1

SOFTWARE REQUIREMENTS

Version 1 - DSKIO - Disk I/O Routines
LIBIO - Magnetic Tape I/O Routines

Version 2 - DSKIO - Disk I/O Routines

MEMORY REQUIREMENTS

Version 1 - 2000_8 words.

Version 2 - 2400_8 words.

OPERATION

The following control statement input to the Resident Monitor causes the loading of AMRMX onto disk:

DSKL (FILE, TAPE/VOL, PACK, "IDENTIFICATION", FLAG)!

where:

FILE is the file number of AMRMX, type RELOC.

TAPE/VOL is the tape number [Version 1] or disk input volume/pack of the form pvv_8 where p is the pack (0 through 7) and vv is the volume (1 through 40_8) [Version 2].

PACK is the output pack number (cannot be the same as the pack in input VOL [Version 2]).

"IDENTIFICATION" is the 19-character or less pack identification string to be written on the output pack's directory.

FLAG is the termination flag with:

FLAG = 0 - enter Monitor just written at end of operation (only valid if PACK = 0).

FLAG ≠ 0 - return control at end to BOOTSTRAP loader, AMLDX.

INTRODUCTION

DSKPY is an ADEPT language program for copying standard AMOS files from disk to magnetic tape and from tape to disk. DSKPY occupies approximately 1000_8 locations of core memory.

SOFTWARE REQUIREMENTS

DSKPY requires the following software items:

1. AMRMX (Version 7) - disk system monitor
2. LIBIO (Version 1 or 2) - magnetic tape I/O routines.

DSKPY also requires the hardware configuration specified for the above software items.

OPERATION

The following call, input via AMRMX statement or equivalent machine language calling sequence, causes the copying of files from tape to disk.

CPYTD (FIRST, LAST, OVOL, DENS)!

where:

FIRST is the first file to be copied
LAST is the last file to be copied
TAPE is the input tape number
OVOL is the output disk volume in the form pvv_8 (p = pack, 0 through 7; vv is the volume, 1 through 40_8)
DENS is the input tape density (0 = 200 BPI, 1 = 556 BPI, 2 = 800 BPI). This argument is ignored on MTP5/8 systems.

Multiple argument sets may be used for copying in the following form:

CPYTD ((F1, L1, T1, O1, D1), (F2, L2, T2), ...etc.)!)

The following call causes the copying of files from disc to tape:

CPYDT (FIRST, LAST, IVOL, TAPE, DENS)!

where:

FIRST is the first file to be copied
LAST is the last file to be copied
IVOL is the input disk volume in the form pvv_8 (p = pack, 0 through 7; vv is the volume, 1 through 40_8.

TAPE is the output tape number
DENS is the output tape density (0 = 200 BPI, 1 = 556 BPI, 2 = 800 BPI)
This argument is ignored on MTP5 and MTP8.

Multiple argument sets may be used for copying in the following form:

CPYDT ((F1, L1, I1, T1, D1), (F2, L2, I2), ...etc.)!)

adage

DSPLY

AGT DISPLAY OPERATOR

Programmer's Reference Manual

Revision F

July 1969

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont.)

INTRODUCTION

THE DSPLY PROGRAM

The AGT Display Operator, DSPLY, is a relocatable machine-language program which serves as the principal user and systems software interface to the graphics processing subsystems of the AGT/10, AGT/30, and AGT/50 graphics terminals. DSPLY, in conjunction with the Resident Monitor, processes graphically-oriented data structures or three-dimensional "images" in memory, and interprets these data structures when necessary to produce the appropriate two-dimensional picture on the CRT.

Once DSPLY has been started it operates as a background program interrupting the foregound program to process particular "image items" or parts of the data structure. Input to DSPLY may be modified as a result of operator interaction with the image or processing of the image input by another program or subroutine. DSPLY refreshes the picture on the CRT at a constant frame rate, and provides for the execution of error routines in the event of a timing error or an error in the data structure.

VERSIONS AND HARDWARE REQUIREMENTS

The DSPLY program exists in several versions, accommodating different hardware configurations of the AGT, as follows:

Version 1 - AGT/10
Version 2 - AGT/30
Version 3 - AGT/50
Version 4 - AGT/10 and LCG1
Version 5 - AGT/30 and LCG1
Version 6 - AGT/50 and LCG1
Version 7 - AGT/30 and HWD1
Version 8 - AGT/50 and HWD1
Version 9 - AGT/30 and LCG1 and HWD1
Version 10 - AGT/50 and LCG1 and HWD1

DESCRIBING DSPLY IMAGES

The unit data structure interpreted by the DSPLY Operator program is a graphical machine-language entity called an Image Item. The set of Image Items interpreted by DSPLY includes the following types:

Element-Generating Image Items draw lines on the CRT, position the CRT beam, and draw character strings.

Transform-Specifying Image Items cause the elements generated by the image description to be changed quantitatively, i.e., to be moved about, scaled, and rotated in three dimensions before they are projected to the two-dimensional face of the CRT.

Viewing Image Items control the appearance of the projected two-dimensional picture to the viewer.

Control Image Items process sub-images conditionally or unconditionally, perform conditional and unconditional branching, and perform program loops.

DSPLY provides several methods of accessing its data. These are specified by the Data Type and Address Mode fields of the data structure.

GRAPHICAL CONCEPTS

A. Spaces and Reference Frames

Images displayed on an AGT are contained in the VIEWING SPACE of the CRT, an imaginary 3-dimensional space defined with reference to the CRT. A given point is specified in this space by the triplet (x, y, z) where x, y and z are fractions in the interval (-1, +1). These values are internally represented on the AGT by signed, 14-bit binary one's-complement fractions. The $\vec{X}_o$, $\vec{Y}_o$ and $\vec{Z}_o$ UNIT VECTORS are oriented in a right-handed coordinate system as diagrammed in Figure 1.

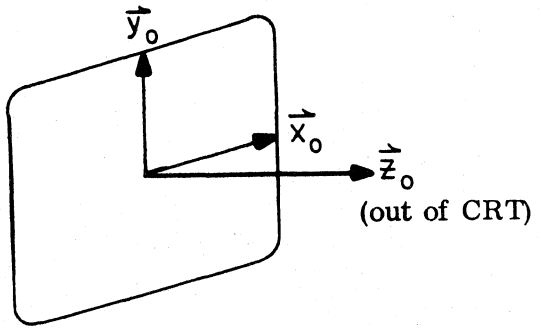


FIGURE 1. Unit Vectors in the CRT's Frame of Reference

Any image or image segment (group of image items) may be displayed with reference to a new coordinate system defined relative to the initial one by a TRANSFORM that is specified by translation, rotation, and scaling.

The resulting (embedded) coordinate space or FRAME OF REFERENCE in which the image is defined is called the IMAGE SPACE. Several levels of COMPOSED frames of reference may be defined in this way. Once the portion of an image which is embedded in a transformed image space has been displayed, the previous transform may be restored. This eliminates any accumulated errors resulting from the composition.

B. Transforms

A point (x, y, z) is displayed under a transform according to this equation:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = s\,[R] \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \vec{d}$$

Where s is a SCALE FACTOR (fraction)

R is a 3 x 3 ROTATION MATRIX
(the matrix of an orthogonal linear transformation in 3-space)

$\vec{d}$ is a DISPLACEMENT VECTOR

x', y' and z' are the coordinates as seen by the EMBEDDING SPACE. (If the embedding space is the viewing space of the CRT, these are the actual coordinates which are projected onto the screen of the CRT.)

The viewing space of the CRT corresponds to the IDENTITY TRANSFORM:

$$s = 1$$

$$[R] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\vec{d} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

[R], S and $\vec{d}$ may be specified with respect to this frame by Load-Transform image items.

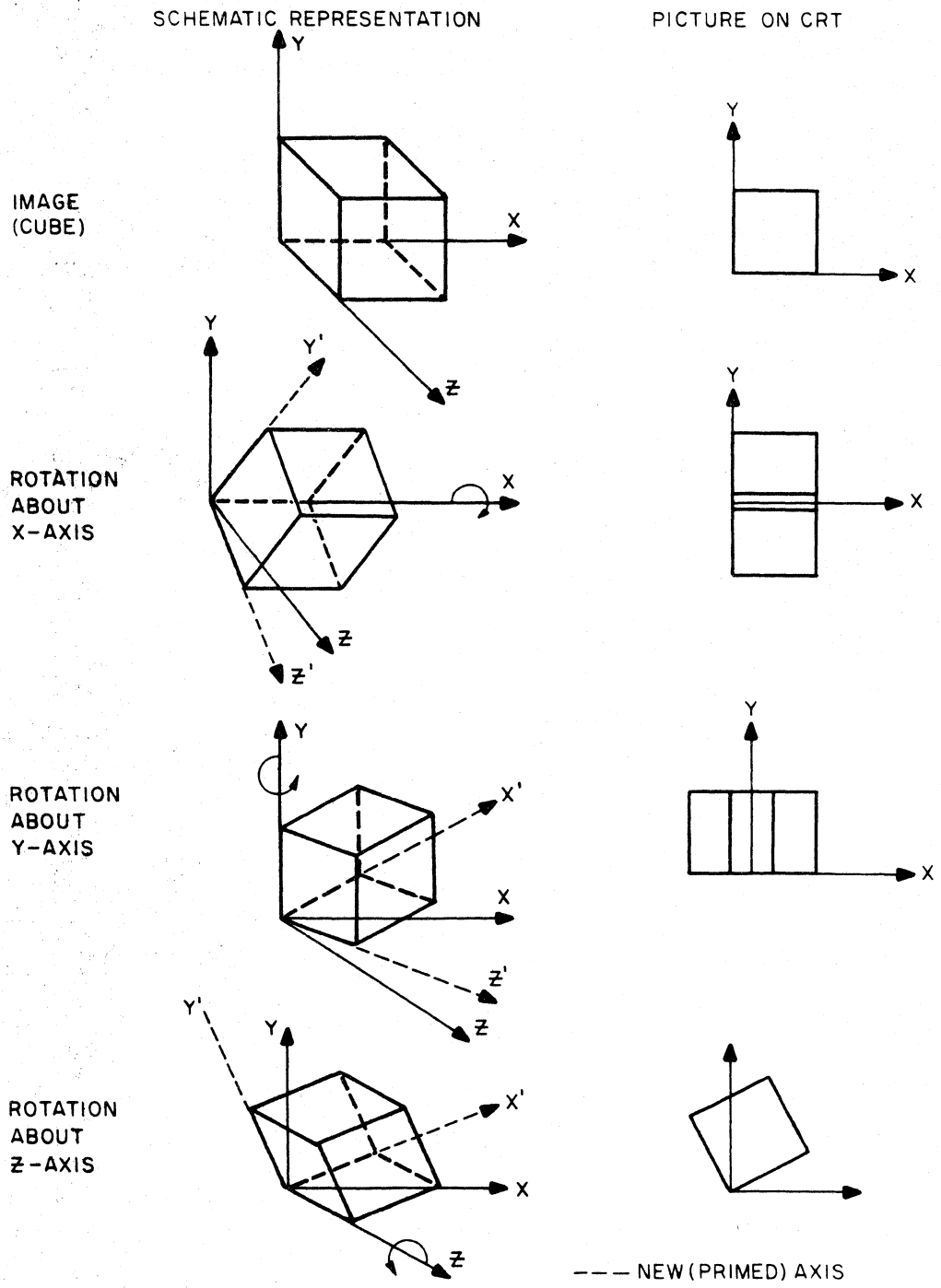


FIGURE 2. Rotations about the X, Y, and Z Axes

The transform applied may be of the following types:

1. Scale

   a. Load Scale - defines s relative to the viewing space of the CRT regardless of the embedding space.

   b. Compose Scale - scales down the three unit vectors defining the image space by the specified scale factor. Any image items displayed in this embedded image space will be scaled down by this factor. The axis definitions will remain unchanged. A negative scale factor causes reflection about all three axes.

2. Displacement

   a. Load Displacement - sets the value of $\vec{d}$ in the CRT's viewing space.

   b. Compose Displacement - displaces in its own image space the specified amount $\vec{dv}$.

$$\vec{d'} = s\,[R]\,(\vec{dv}) + \vec{d}$$

3. Rotate (not implemented in AGT/10 versions)

   a. Load Rotation - sets [R] to a rotation matrix of an angle about the specified axis in the CRT's frame of reference.

   Suppose the image displayed is a cube as in Figure 2. A rotation about one of the axes changes the definition of the other two axes with respect to the CRT. When three rotations are specified by the same image item the Z-rotation is performed first, then the Y-rotation relative to the new Y-axis, and finally the X-rotation relative to the new X-axis.

   b. Compose Rotation - rotates the entire image space the specified amount. This operation changes the definition of the other two axes relative to the current image space.

C. Images

An image is a set of items which constitute a complete segment for display via the DSPLY operator. An item (IMG) is available for including completed images as part of another image.

The "transform-specifying" items just described may be used to "place" image segments in a flexible manner. For example, boxes representing buildings could be scaled and placed on an empty lot shape for use in area studies. This can be done by load-transform items followed by IMG items which refer to a box-image. There can be several box-shapes representing the desired building types.

When viewed in this manner, the type of building at any place, the shape of all buildings of a given type, and the size or location of any one building can all be changed easily by varying only a few arguments to particular items.

Either load or compose transform items would be adequate for describing one view of the above lot.

The lot may be moved or turned, giving different views, or it may be replicated and placed repeatedly over an area representing one or more city blocks. In both cases, the buildings should be placed on the lots by compose operations so that they will "stay with" the lots as they are scaled or moved.

Once the variable parts of an image have been adjusted, the resulting image may be expanded into an inflexible image consisting entirely of element generating items for more efficient display or use by other images. This can be done by the FREEZ operator program.

D. Views

Control of the projected two-dimensional picture of an image as seen by a viewer is effected by items and statements referred to as "viewing" operations.

These affect such qualities as overall picture scale, front cut-off blanking, intensity modulation or depth cueing, and partitioning of the display surface into bounded windows.

DISPLAY OPERATOR STATEMENTS

The following operations can be performed by inputting the given statements on the teletype to AMRMX or by executing equivalent calling sequences in memory.

A. DSPLY - Start Display Program

DSPLY may be started by a statement input to the on-line monitor, AMRMX, or by executing an equivalent calling sequence in memory. The program which starts DSPLY then becomes the foreground program. The monitor statement:

DSPLY (IMAGE, RATE, ERRORI, ERRORT)!

causes the data structure starting at location IMAGE to be interpreted by DSPLY, and the corresponding two-dimensional picture to be displayed on the CRT refreshed at the specified RATE.

B. RATE Definition

The frame RATE is defined by the equation:

$$\text{FRAMES/SEC} = 120_{10}/(\text{RATE} + 1)$$

For example:

RATE	FRAMES/SEC
1	60
2	40
3	30
4	24
5	20
7	15
9_{10}	12
11_{10}	10

C. HALT - Stop Display Program

DSPLY may be stopped and control returned to AMRMX by depressing the PULSE 1 operator interrupt button. Also, typing the statement:

HALT!

to AMRMX (or executing an equivalent calling sequence in memory) stops all display.

D. SKIP - Image Error Option

In the case of an error in the IMAGE data structure, control is transferred to the ERRORI routine. The entry point SKIP is provided by DSPLY for this purpose. When executed SKIP will notify the operator (via the teletype) of the address of the faulty image item and continue interpreting without processing the remainder of the subimage containing the faulty image item.

E. ALL, CUT, BUMP - Timing Error Options

If the IMAGE cannot be interpreted in the time allotted by the specified RATE, a timing error occurs. In this case control is transferred to the ERRORT routine. The following entry points may be used for this purpose:

ALL	Interpret the entire IMAGE at a slower RATE.
CUT	Display as much of the IMAGE as possible at the specified frame RATE.
BUMP	Advance RATE enough so that the entire IMAGE may be processed, and notify the operator (via the teletype) of the new RATE at which the picture on the CRT is being refreshed.

F. DSPLY - Default Arguments

If any of the arguments are omitted in the DSPLY statement the previously specified ones are assumed. Initially, DSPLY assumes the following arguments:

Argument	Initially assumed value
RATE	2 (40 frames/sec)
ERRORI	SKIP
ERRORT	ALL

G. VIEW - Define Depth of Viewing Space (not implemented in AGT/10 versions)

The Z-dimension in the data structure refers to the depth of the image in three-space. This is analagous to the depth into the viewing space of the CRT. The Z-viewing parameters are set by the following statement:

VIEW (FRONT, BACK, DIM)!

When DIM is Ø all lines are shown at maximum intensity regardless of depth. DIM is initially assumed to be Ø.

The AGT/30 and AGT/50 have the capability of providing "depth cueing" to remove the visual ambiguity between the front and back of displayed three-dimensional images. Lines near the front of the image may be shown very bright in comparison to those in the back. When the absolute value of DIM is full scale (37777), the maximum amount of such "dimming" occurs.

FRONT and BACK specify the range in Z which is to be displayed when DIM is maximum (37777). Those portions of lines which correspond to Z-values greater than the specified FRONT are blanked. On systems with the optional Hardware Windowing Operator (HWD1) any portions of lines outside of the specified range in Z are blanked.

FRONT and BACK are specified as 5-digit octal fractions ranging from full scale forward (37777) to full scale into the CRT (4ØØØØ). Until otherwise specified they are assumed to be 37777 and 4ØØØØ respectively.

H. CLOCK -Append to CLOCK Chain

DSPLY has provision for requesting the execution of programs once per frame. Real time inputs such as the variable control dial values may be sampled in this way. The monitor statement:

CLOCK (ROUTINE, EXIT)!

or the equivalent AFORT or ADEPT calling sequence causes control to be transferred to ROUTINE once per frame. EXIT is the address of the last instruction executed by ROUTINE. CLOCK will change location EXIT to the return that ROUTINE must take. Subsequent calls to CLOCK cause new routines to be "chained" to EXIT and performed in sequence.

NOTE

ROUTINE is taken as the address of the first executable instruction of the routine to be chained, NOT as the entry-point. A subroutine can be executed on the clock chain via the monitor statement:

CLOCK (ENTRY + 1, ENTRY)!

where ENTRY is the subroutine's entry point.

Routines placed on the CLOCK chain are executed on a lower priority interrupt level (programmatic interrupt, PINT) than the vector generator and the DSPLY item-processing program. This enables such routines to be executed "simultaneously" with the DSPLY program (i.e., while the vector generator is drawing lines or processing image items automatically).

I. CLKOF - Reset CLOCK Chain

The monitor statement:

CLKOF!

causes the CLOCK chain to be reset clearing the effect of all previous CLOCK calls.

J. CLOCK Routine Restrictions

Since the routines executed on the CLOCK chain are executed on a higher priority interrupt level than the foreground program, they must not perform any JUMP'I instructions. The AR need not be restored. These routines may be interrupted by all higher priority levels. All registers except the ER will be restored; therefore, portions of the code which depend on the ER or have strick timing restrictions must be bracketed by FPRI (freeze priority interrupts) instructions. Calling ROUTINE by the CLOCK chain does not affect the time required to display IMAGE; therefore, if ROUTINE must be called once per frame, it is better to use the CLOCK chain than a "JSR ROUTINE" image item.

K. SnSET - Specify Logical Scopes 1, 2, 3 and 4

The AGT is supplied with up to four CRT's or "physical scope units." These are named physical scopes 1, 2, 3 and 4. DSPLY may select any one of four "logical scope units" for display.

They may be assigned to any of the four physical scopes by the statements:

S1SET (Physical scope unit list)!
S2SET (Physical scope unit list)!
S3SET (Physical scope unit list)!
S4SET (Physical scope unit list)!

respectively. The "physical scope unit list" consists of the appropriate physical scope unit numbers separated by commas. Until otherwise specified each Logical Scope Unit is assigned to all four physical scope units.

IMAGE DATA STRUCTURE

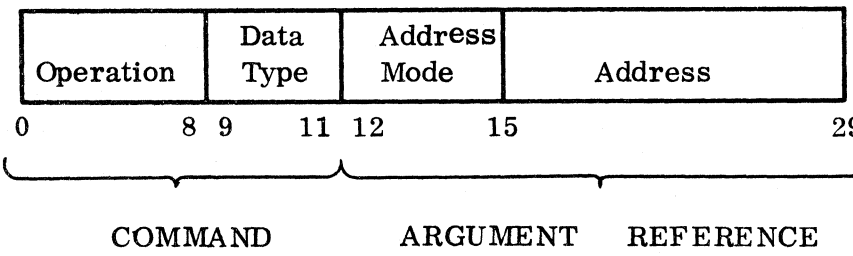
A. Image Format

An image data structure in memory consists of a two-word header followed by a list of image items which describe the image. The header is ignored by DSPLY, and usually contains the AMOS internal 6-bit code for the image NAME. The last-interpreted image item must be a RET; thus, in general an image looks like this:

```
NAME:   TEXT/NAME    /      [2 Words
        ITEM 1
        ITEM 2
          .
          .
          .
        RET                 [Last Item
```

B. Image Item Format

Image items or parts are variable in length. The first 30-bit word of each image item contains the item's command. It has the following format:



Each operation (as it is defined in the section entitled "IMAGE ITEM OPERATIONS") is associated with a number of arguments. The arguments are obtained according to the Data Type specified in the Image Command; therefore, the number of 3Ø-bit words in an image item depends upon:

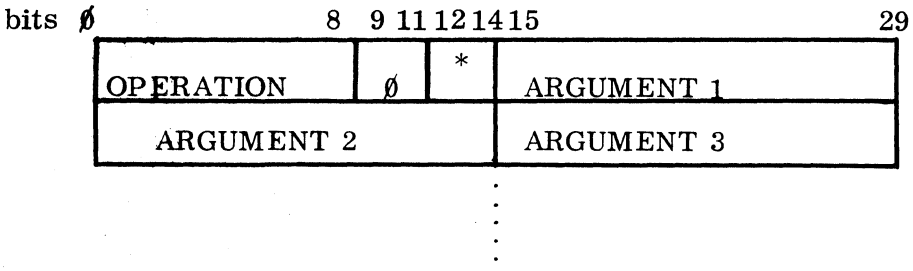
1. The number of arguments used by the Item's Operation.

2. The way in which these arguments are defined as specified by the Item's Data Type.

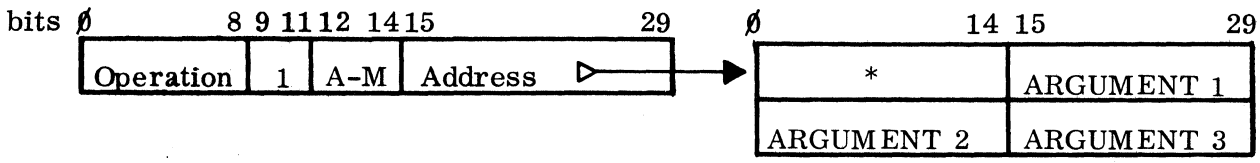
C. Data Type Definitions

The following diagrams illustrate, for all Operations, the way in which the Data Type specifies the definition of arguments.

Immediate Half-Word Value List (Assumed)

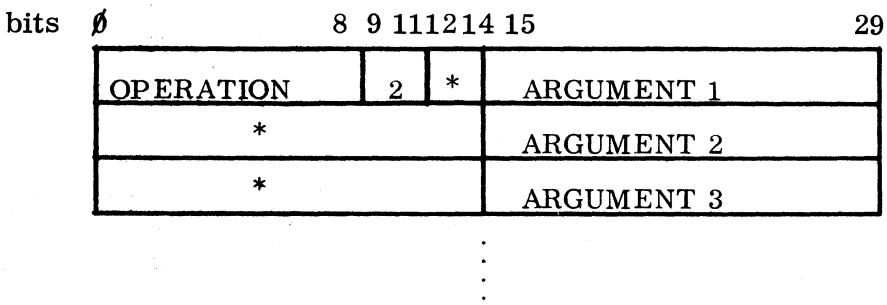


Referenced Half-Word Value List



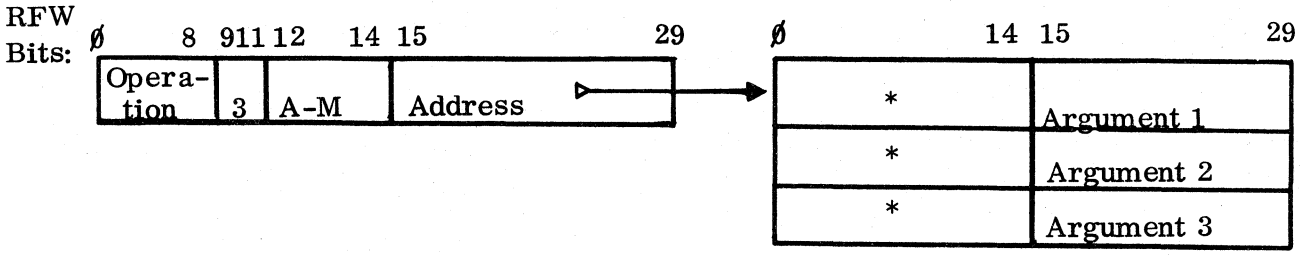
Immediate Full-Word Value List

IFW

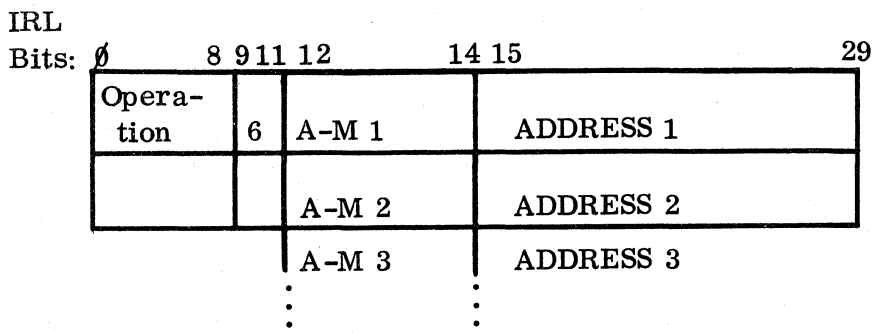


* These fields are ignored by DSPLY.

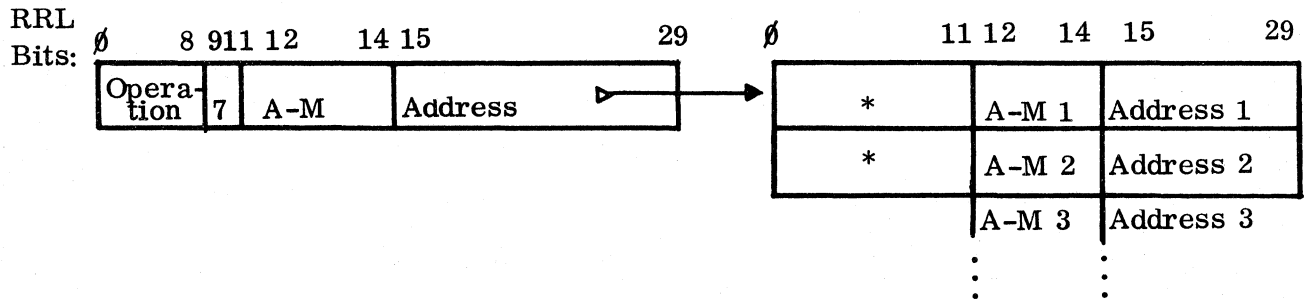
Referenced Full-Word Value List



Immediate Reference List



Referenced Reference List



* These fields are ignored by DSPLY.

D. Address Mode Definitions

The Address Mode Field of the Image Command specifies the way in which a single argument is to be found from its Address.

Immediate (Assumed IMED)	This mode (Address Mode Ø) indicates that the associated 15-bit Address Field is the value sought.
Direct DIR	This mode (Address Mode 4) indicates that the associated Address is the location of the value sought.
Indirect I	This mode (Address Mode 1) indicates that the associated Address is the first of an arbitrary long chain of indirect addresses. The first non-indirect address in the chain is the location of the value sought.
Structure STR	This mode (Address Mode 5) requires two machine words to determine its associated value. The Address Field of the first word is used as an offset; it is added to the final reference obtained from the second to locate the value sought.

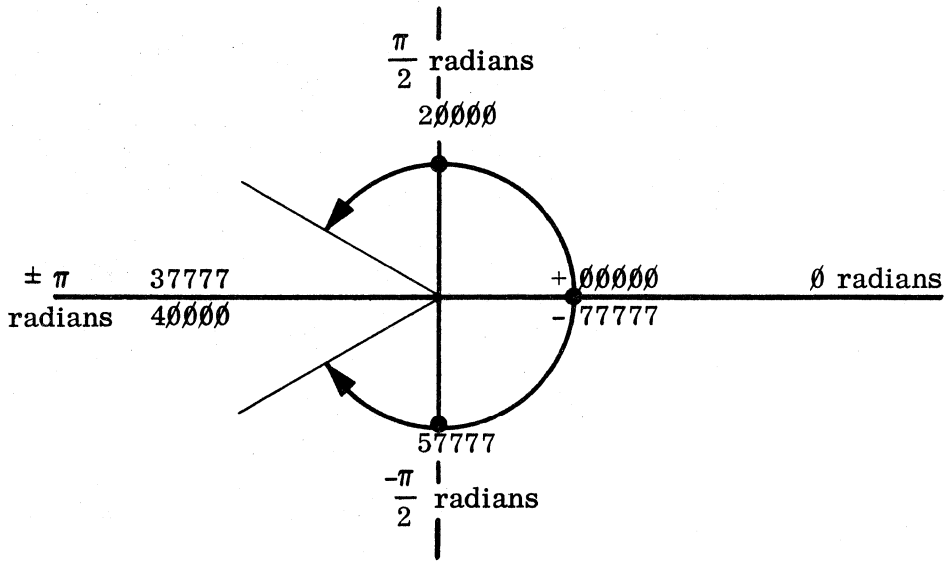
E. Argument Values

An Argument is a fifteen-bit binary value, which may be interpreted by the Image Item Command in one of the following ways:

1. integer -- a signed 14-bit one's complement binary integer.
2. address -- a pointer to another location in memory.
3. fraction -- a signed 14-bit one's complement binary fraction. Values displayed on the CRT range from 37777_8 (full scale) to 4ØØØØ (minus full scale).
4. angle value -- expressed in radians as a fraction, modulo 2π and scaled by $1/\pi$.

For example,

Angle Value, radians	Fraction	Octal Value
$\pi \pm 2\pi n$ *	1	37777 (Full Scale)
$\pi/2 \pm 2\pi n$	1/2	2ØØØØ
$Ø \pm 2\pi n$	Ø	Ø or 77777
$-\pi/2 \pm 2\pi n$	- 1/2	57777
$-\pi \pm 2\pi n$	-1	4ØØØØ



* n is an integer

IMAGE ITEM OPERATIONS

Each Image Operation is described below, along with the argument(s) it requires. These arguments are obtained from the image item according to the Data Type and Address Mode fields. In the following they are presented in parenthese following the operation's name for the sake of clarity.

The argument names are also encoded as follows:

First Letter	Meaning of 15-bit Argument
F	Fraction
I	Integer
L	Location Address
A	Angle Value (a fraction)
B	Binary Value

In the following descriptions [T] (X, Y, Z) means "the current transformation (translation, scale and then rotation) applied to the vector (X, Y, Z)."

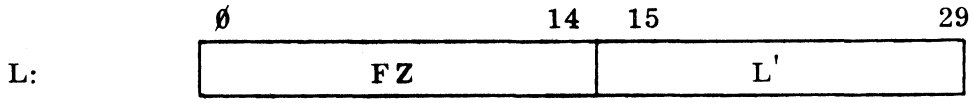
A. Element-Generating

DRAW (FZ, FX, FY) — Draws a line on the scope from the last beam position to the point [T] (FX, FY, FZ).

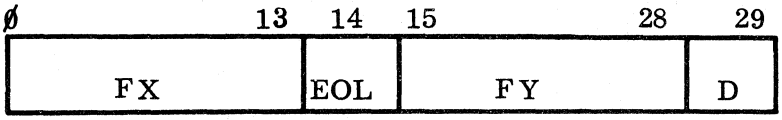
MOVE (FZ, FX, FY) — Moves the beam from any position to the point [T] (FX, FY, FZ) without displaying.

2DTBL (L) — 2D Table:

Displays a sequence of 2-D tables in which each consists of a 1-word header and consecutive 14-bit value pairs. The sequence starts at location L. The first word of each table has this format:



Where FZ is the Z-value at which this table is to be displayed, L' is the address of the next table in the sequence to be displayed (or ø if there is none). Values in each table are packed as follows:



D is the "draw" control bit. When D = 1 this vector is drawn; otherwise, the vector is blanked. EOL is the "end of list" control bit which indicates the end of the table. When it is encountered the next table is started, if there is one.

2DTF (L) — Same as 2DTBL except all vectors are short (of length less than 0.5") and one vector is displayed every 5.0 μs. Longer vector lengths result in poor end matching and low intensities. At lengths of two inches and greater, gross errors and distortions result.

2DLST (L) — Same as 2DTBL except that the picture drawn is not intended to be part of a three-dimensional image. It occupies a larger portion of the scope face. This item is useful for drawing graphs.

AGT/50 Tables

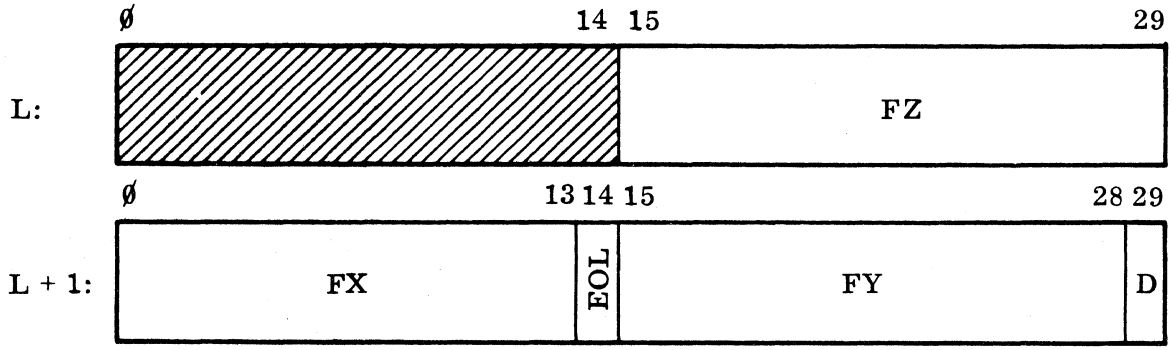
The following image items (in AGT/50 versions only) display lists of vectors in the various AGT/50 table formats. Each table consists of a one-word header followed by consecutive 30-bit table words. Each table word contains the X and Y or X, Y, and Z endpoint-values for a vector. Also contained in each table word are the draw (D) and end-of-list (EOL) control bits (see description of 2DTBL above). In tables which contain only X- and Y-values in each table word (2DTAB and 2DROT), the one-word header is used to contain the Z-value at which the table is to be displayed. In tables which also contain Z-values in each table word (2DINT and 3DTAB), the one-word header is ignored but must be present. The header and table word formats are different for each table and are presented under the individual item descriptions below. (Shaded fields are ignored.)

Note that links to additional tables are not present in the header words of the AGT/50 tables. This feature is provided in the 2DTBL, 2DTF, and 2DLST image items described above to eliminate the overhead time required for switching tables on AGT/10 and AGT/30 systems. However, the automatic table-switching facility of the AGT/50 eliminates the need for these links. In order to provide upward compatibility with AGT/10 and AGT/30 DSPLY programs, the 2DTBL, 2DTF, and 2DLST tables are also available on AGT/50 versions, but their use is discouraged in favor of the newer table formats below.

2DTAB (L)

2D Full Precision Table:

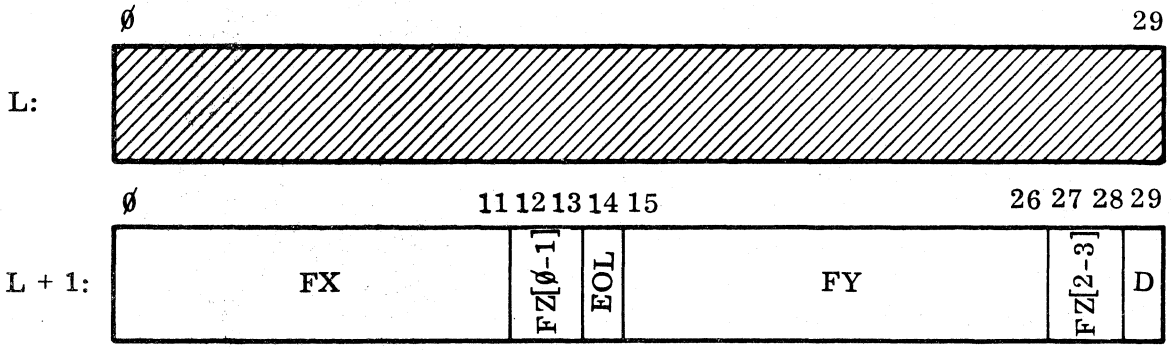
Same as 2DTBL, but with a different format for the first table word. This format allows full-precision (14-bits) of X and Y values for vectors in a plane (i.e., constant Z-value).



2DINT (L)

2D Intensity Table:

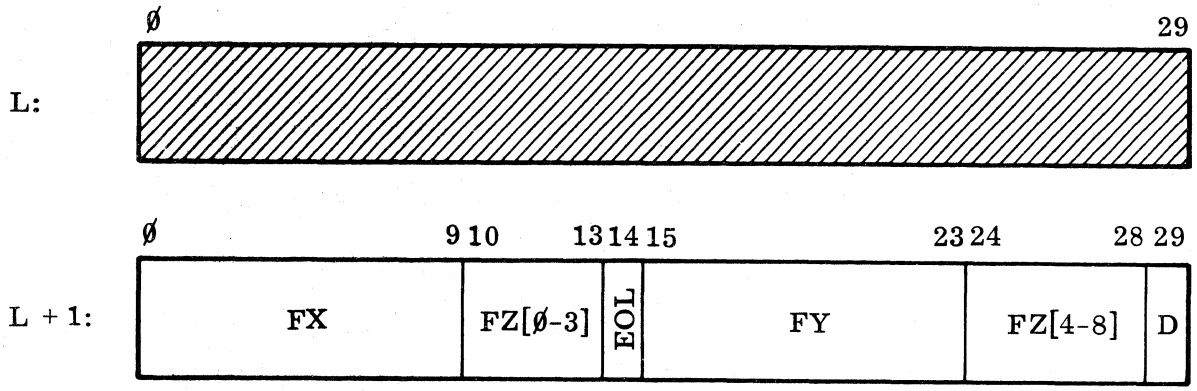
Allows 16-level intensity control (4 bits in Z) of vectors while maintaining 12-bit precision of X and Y values.



3DTAB (L)

3D Table:

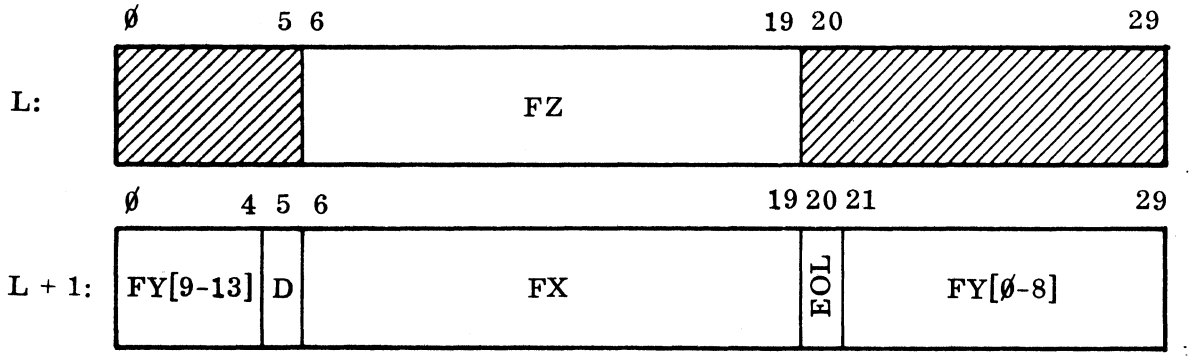
Allows three-dimensional images to be constructed with one word per vector by using 10 bits for X, 9 bits for Y, and 9 bits for Z.



2DROT (L)

2D Rotated Table:

Same as 2DTAB, but with each table word rotated six-bits to the right. If only 5 bits of precision for X and Y vectors is required, this table format may be overlapped with that of the 2DTAB image item to allow two separate lists of vectors to occupy the same area of memory.



2DTAF (L) 2DINF (L) 3DTAF (L) 2DROF (L)	Same as the above four tables except that all vectors are short (of length less than 0.5") and one vector is displayed every 5.0μs. Longer vector lengths result in poor end matching and low intensities. At lengths of two inches and greater, gross errors and distortions result.
LABL (L) * LABR (L) **	Displays text characters from a string of packed codes in consecutive words starting at location L. LABL displays the string upright with reference frame of the CRT starting at the present beam position. LABLR rotates the entire character display 90° counter-clockwise. At the conclusion of the character string, the beam is moved back to its initial position (before the string was drawn). See the section entitled "CHARACTER STRING FORMAT."

* May be deleted in DSPLY, Version 1 (AGT/10 systems without LCG1).
** Not implemented in DSPLY, Version 1.

B. Transform-Specifying

SCL (F)	Scales down the following items by scale factor F.
ROTX (A) * ROTY (A) * ROTZ (A)*	These operations rotate the image specified by the following items about the indicated reference axis (X, Y, or Z respectively) by the angle A.
RXYZ (AZ, AX, AY)*	This operation rotates as would the items: ROTZ (AZ); ROTY (AY); ROTX (AX).
DX (F) DY (F) DZ (F)	These operations displace the following image portions by the distance F along the specified local reference axis.
DV (FZ, FX, FY)	This operation repositions as would the items: DX (FX); DY (FY); DZ (FZ).

LDSCL (F)
LDRX (A)*
LDRY (A)*
LDRZ (A)*
LDRV (AZ, AX, AY)*
LDX (F)
LDY (F)
LDZ (F)
LDV (FZ, FX, FY)
	These operations set the values for a particular transformation matrix element(s). LDCSL sets the scale for the following elements. LDX, LDY, LDZ, and LDV set the origin position for follow-elements. LDRX, . . . sets the rotation matrix to give the specified rotation. The LOAD operation implies that previous orientations, scales, etc., are replaced by corresponding load items. (Any items generated are defined with respect to the CRT's frame of reference; any transforms are specified with respect to its axes.)

LDI	This operation causes the image specified by the following items to be displayed with the identity transform, i.e., in the CRT's frame of reference.

C. Viewing

LDPS (F)	Sets the overall picture scale of images in the CRT's frame of reference to the value F. Two entry-points, 3DPS and 2DPS. are provided by DSPLY for this purpose. (See section entitled "USEFUL ENTRY POINTS.")

(continuation)

* Interpreted as NUL in AGT/10 versions.

A full-scale picture scale (37777) corresponds to a 20-inch square display area centered on the screen of the CRT. As the visible display area is a 12-inch square, a smaller than full-scale picture scale is generally more useful. 3DPS is initially assumed.

LDW (FH, FV, FDH, FDV) **	Sets window parameters (horizontal and vertical) to include a field centered at the point FH, FV with horizontal and vertical dimensions FDH and FDV. **
MVW (FH, FV) **	Moves a viewer's window, centering it at the point (FH, FV) while retaining the same dimensions. **
LWS (FDH, FDV) **	Changes a window size to have new horizontal and vertical dimensions FDH and FDV. The window remains centered at its original position. **
DEPTH (F1, F2, FDIM)*	Sets the Z-cutoff and depth cueing parameters for display of following part of the image. F1, F2, and FDIM have the same meaning as FRONT, BACK and DIM in the VIEW monitor statement.

D. Control

NUL	This operation is ignored, and the DT and Argument fields may be used in a calling sequence to pass arguments to subroutines called by the JSR image item.
SAVT	Saves the currently-specified transform. This saved transform may be reinstated later by a REST operation.

* Interpreted as NUL in AGT/10 versions.
** Implemented only with the optional Hardware Windowing Operator (HWD1).

REST	This operation is used to reinstate the previously-saved transform, so that the following image items are defined in its frame of reference.
RET	This operation terminates the description of an image. If the image was processed as a part of another image's description (by an IMG command) that image's frame of reference is restored and processing resumes in its definition.
IMG (L)	This operation in the current image causes the entire image at location L to be processed relative to the current transform as part of the current image's definition.
LOOP (I)	This operation is used to initiate a loop of items whose processing is to be repeated I times. No nesting is permitted within one transform level of any one image; thus, IMG commands referencing sub-images with loops are permitted, and loops may be nested by bracketing them in SAVT-REST pairs.
ENDL	This operation is used to terminate a repeat loop. The loop count is decremented and tested and the loop repeated until the count is zero, then subsequent items are processed.
JMP (L)	This operation causes processing of the current image to branch to location L where the current description resumes.
JSR (L)	Execute machine language subroutine at location L.
SHORT*	Sets the AGT/50 "short" operation mode so that all succeeding DRAW and MOVE image items are treated as short (less than 0.5" long) vectors where one vector is displayed every 6μs.

LONG *	Resets the AGT/50 operation mode so that succeeding DRAW and MOVE image items may specify any length vectors. The minimum drawing time per vector is 9 μs, the maximum is 39 μs. Initially, the "long" mode is assumed and remains constituted until a "short" mode is encountered. Upon returning from a sub-image called by an IMG image item, the previous drawing mode is restored.

These image items use the Status Register (SR) which is described in the next section.

CJMP'C (L)	Conditional Branch in image if any of the Status Sense Bits selected by C are true. C may denote Flag 1, Flag 2, and/or Pen Detect.
WJMP'WW (L)**	Conditional Branch in image if any of the specified window boundaries have been crossed. Branch to L when any of the windowing bits [9-14] in the Status Register as selected by WW are true.
LDMB (B)	Load Mode bits of the Status Register, bits [Ø-7] from parameter B, bits [15-22].
ORMB (B)	OR Mode bits [15-22] of parameter B into bits [Ø-7] of the Status Register. Bits [23-29] of SR remain unchanged. This item allows for independent setting of the Mode bits.
ANDMB (B)	AND bits [15-22] of parameter B into bits [Ø-7] of the Status Register (Mode Bits only). Bits [23-29] of SR remain unchanged. This item allows for independent resetting of the Mode bits of SR.
LDSN (B)	Load flag bits from sign bits, bits Ø and 15 of value B into bits Flag1 and Flag 2 of SR.

* Treated as NUL image item in AGT/10 and AGT/30 versions; has no effect on AGT/50 table image items.

** Implemented only with the optional Hardware Windowing Operator (HWD1).

LDLS (B)	Load flag bits from upper and lower significance, of value B. If bits [Ø-14] of B are not Ø, set Flag 1 in SR; otherwise, reset Flag 1. Similarly, if bits [15-29] of value B are not Ø, set Flag 2; otherwise, reset Flag 2.

E. Restrictions for Subroutines Called by JSR-Type Image Items

The subroutine L may use a calling sequence to pass arguments provided that bits [Ø-8] of the arguments are Ø (NUL-operation image items). The next image item is temporarily replaced by DSPLY and stored in location JSRSV. For this reason if the JSR image item must alter or refer to the image item following it, it should alter or refer to $JSRSV.

The subroutine L must not execute any "JUMP'I" instructions. It must not generate any interrupts from the AVG. Because of its priority level it will not service any interrupts from the LCG (Character Generator), LPR (Line Printer), or the programmatic interrupt.

The subroutine L must not reset bit 5 of Destination 10 (Display Clock Hardware Control). Also, any registers in the hybrid array that are changed must be restored. (See the Section entitled "USEFUL ENTRY POINTS.")

THE STATUS REGISTER (SR)

The Status Register (SR) is a 15-bit software register that is maintained by DSPLY and may be changed or interrogated by items in the image data structure. SR may also be referenced by a conditional image item to perform a machine-language subroutine or to branch to another location in the image data structure.

A. Mode Bits of SR

Bits [Ø-7] of SR are called Mode Bits; they control various aspects of the displaying process. The Mode Bits may be set and reset by the LDMB, ORMB, ANDMB, LDSN, and LDLS image items.

B. Sense Bits of SR

Bits [6-14] of SR are called Sense Bits; they describe various display and core conditions. The Sense Bits can be sensed by the conditional image items CJMP'CC, CJSR'CC, WJMP'WW*, WJSR'WW* and used to alter the image processing path.

STATUS REGISTER FORMAT

The Status Register has the following format:

bits: Ø 1	2 3	4	5	6	7	8	9	1Ø	11	12	13	14
Scope Field S C	Visual Element Field VEL	PE	WE	FLAG1	FLAG2	PD	WBX1	WBX2	WBY1	WBY2	WBZ1	WBZ2
				Mode Bits			Sense Bits					

C. Explanation of Status Register Bits

SC — Scope Field

This two-bit field specifies which of the four logical scopes is selected for display. These are assigned to any of the four physical scopes by the SnSET commands. Initially Logical Scope 1 is selected.

---

* Used only with optional Hardware Windowing Operator (HWD1).

The logical scopes are selected according to the following table:

SR Bit: Ø	1	Logical Scope Selected
Ø	Ø	1
Ø	1	2
1	Ø	3
1	1	4

VEL Visual Element Field

This field sets any one of four (two on non-AGT/50 versions) visual element modes for the displaying of vectors as follows:

SR Bit: 2	3	Visual Element Mode Selected
Ø	Ø	Line
1	Ø	Dash
1	1	Point*
Ø	1	Stroke*

Line mode is the normal vector drawing mode.

Dash mode causes the intensity of lines to be modulated, producing dashed lines.

Point mode (AGT/50 only) causes only the end-points of draw-type vectors to be intensified (lines are not drawn).

Stroke mode (AGT/50 only) speeds up the vector drawing rate from 5.0 μs to 4.0 μs in the 2DTF, 2DTAF, 2DINF, 3DTAF, and 2DROF image items. Stroke mode has no effect (i.e., is treated as line mode) on other image items. This mode assumes that vectors are shorter than 0.25 in. and are drawn at or near right angles to each other; it is most useful for constructing curved lines and surfaces.

PE Pen Enable

This bit resets the PD bit, and causes the light pen to sense light from the currently selected logical scope unit when a vector beam is encountered. When the light is sensed the PD bit will be set and the "Pen-Hit Table" updated (via a hardware interrupt). When the PE bit is reset hardware light pen interrupts will be ignored.

* AGT/50 versions only.

WE — Window Enable*

Resets window bound detect bits and allows interrupts to be generated when windowing bounds are crossed.

FLAG1
FLAG2 — Image Flag Bits

These bits may be used by the image as flags to indicate any condition. They may be set or reset within the image and tested at any point by a CJMP or CJSR image item.

PD — Pen Detect

This bit is set whenever the light pen is detected while enabled and reset when Pen Enable is set. It may be tested by the CJMP'PEN and CJSR'PEN image items.

WBX1
WBX2
WBY1
WBY2
WBZ1
WBZ2 — Window Bounds Detect*

These six bits indicate whether the specified window bounds have been crossed. They are reset whenever Window Enable is set and may be tested by the WJMP'WW and WJSR'WW image items.

* Used only with optional Hardware Windowing Operator (HWD1).

USEFUL ENTRY POINTS

A. **PENHT - Read Pen-Hit Table**

This subroutine will read the nth previous entry in the Pen-Hit Table, a circular table with 16_{10} entries. The execution of the calling sequence:

```
JPSR  $PENHT
      N          Address of n
      ITA
      IMA
```

will set location ITA to the address of the image item seen by the light pen; it will set location IMA to the address of the image in which the item was seen. Entries are made in the Pen-Hit table only when the PE bit in SR is set.

B. **TRF - Current Transform Storage**

The values of the current transform entries are stored in memory starting at location TRF. These contain the actual values that are loaded into the array registers.

AGT/10 Version Transform

Location (octal)	Transform Entry (contained in bits 15-29)
TRF + 1	S - Scale factor
TRF + 2	DX - X-displacement
TRF + 3	DY - Y-displacement
TRF + 4	DZ - Z-displacement

AGT/30 and AGT/50 Version Transform

	Location (octal)	Transform Entry (contained in bits 15-29)
	TRF + 1	* R_{12} Matrix entry
	TRF + 2	R_{21} Matrix entry
	TRF + 3	S Scale factor
	TRF + 4	R_{11} Matrix entry
	TRF + 5	R_{22} Matrix entry
	TRF + 6	R_{33} Matrix entry
	TRF + 7	R_{23} Matrix entry
	TRF + 1Ø	R_{32} Matrix entry
	TRF + 11	R_{13} Matrix entry
	TRF + 12	R_{31} Matrix entry
	TRF + 13	DX X-displacement
	TRF + 14	DY Y-displacement
	TRF + 15	DZ Z-displacement
PS:	TRF + 16	** PS Overall picture scale

C. LDTRF - Load the Current Transform

This subroutine reloads the current transform stored in memory into the hybrid array hardware registers. It may be used by a subroutine called by a JSR image item but not by a foreground program or a program executed on the clock chain.

* The notation R_{AB} means "the matrix entry in row A, column B of the matrix [R]."

** PS must contain ØØØ2Ø!H in bits Ø-14.

D. LDVEW - Load Viewing Parameters (AGT/30 and AGT/50 versions only)

This subroutine reloads the array registers which specify the viewing parameters from storage in memory.

E. SR - Status Register

The Status Register bits Ø-14 are stored in entry point SR, bits 15-29.

F. DSPTB - Dispatch Table Base Address

A subroutine SUBR may be executed as an image item by assigning it an unused OPERATION code. If the code is

abcØØ$_8$!H

where a, b, and c are octal digits, location

DSPTB + [20$_8$ * a + (b c & 17)]

should contain the instruction

JPSR SUBR

SUBR should return to the next location.

G. IFA, ARGS, APTR - Process Standard Arguments

When SUBR (above) is executed, entry point IFA contains the address of the image item which called it. During the execution of SUBR, IFA must be advanced to the first word of the next image item to be processed. This may be done by either indexing IFA the proper amount or fetching standard arguments.

The subroutine ARGS accesses standard arguments according to Data Type and Address Mode fields and advances IFA to the next image item.

Calling sequence:
(AR) = n, bits 15-29
JPSR $ARGS
. Returns to next location

Result: Successive arguments may be fetched by MDAR'I'X $APTR instructions.

H. 3DPS, 2DPS - Standard Picture Scale Values

The standard 3-D picture scale (3DPS) has a value equal to $(1 + \sqrt{3})^{-1}$, which allows a full-scale cube to be completely rotated about its center within the 12-inch square visible display area of the CRT. Similarly, the 2-D picture scale (2DPS) has a value equal to $(1 + \sqrt{2})^{-1}$, which allows a full-scale square to be rotated within the visible display area. These values may be used with the LDPS image item. Also, they contain 2Ø!H in bits Ø-14, to enable insertion in location TRF+16, as described above.

I. Restrictions for Subroutines Called as Image Items

Calling Sequence:
JPSR SUBR
. Returns to next location

Results: Subroutine SUBR must advance location IFA to the next image item as described above. SUBR must not execute any "JUMP'I" instructions. It must not generate any interrupts from the AVG; it will not service any interrupts from the LCG (Character Generator), LPR (Line Printer) or the programmatic interrupt. SUBR must not reset bit 5 of Destination 1Ø (Display Clock Hardware Control). Also, any registers in the hybrid array that are changed must be restored. (See entry points LDTRF, LDVEW above.)

CHARACTER STRING FORMAT

The character strings processed by LABL are stored as lists of 30-bit words which are assumed to be in the following format:

Character Number:

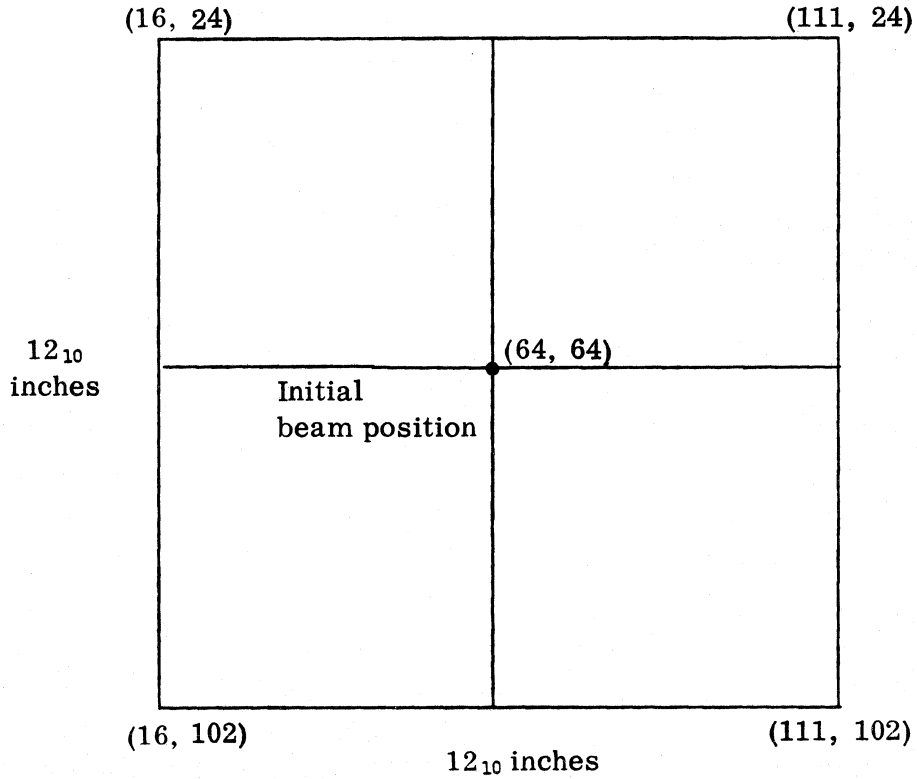
1	2		3	4	
7-bit Char.	7-bit Char.	A	7-bit Char.	7-bit Char.	B

Bit Positions: 0 6 7 13 14 15 21 22 28 29

A is a flag bit; if set it indicates the end-of-string.

B is used by the programming system and should be left 0.

Character strings are displayed starting at the last beam position and may be displayed from that position on a grid of 64_{10} lines of 128_{10} characters (smallest character size) as specified by the HT and VT control characters. The initial position of the beam is defined as (64, 64) on the grid. The center area of the grid of size 96_{10} x 40_{10} is scaled to fit within the 12_{10} inch usable display area as shown below if the beam position was initially at the center of the screen. (On the Y-axis, characters can be positioned only on even-numbered lines.)

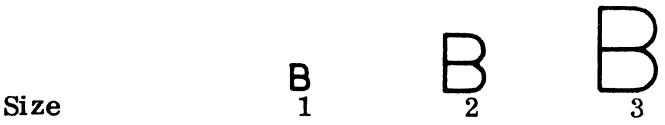


40 lines x 96 character grid

LABLR rotates the entire grid 90 degrees counter-clockwise. * This is useful for labeling vertical axes on graphs. After the character string has been displayed, the beam remains in its previous position.

Each 7-bit character may be used for control or display. The control characters can be used to position the beam, select the character size, shape, brighten and format the text (line feed, carriage return, superscripts and subscripts).

Characters may be displayed in any of the following three sizes:



At the beginning of a character string, the size is always 2. Size increase or decrease is determined by control characters.

If the italic mode is selected, characters will be drawn slanted, as italics.

The italic mode is initially off. *

If the vertical spacing mode is selected (by using the Space Switch control character) characters will be spaced automatically on a vertical line.

B
B
B

The vertical spacing mode is initially off.

There are three levels of brightness. The initial level at the beginning of a string of characters is the intermediate brightness level. Brightness increase or decrease is done by control characters. *

---

* Not implemented in non-LCG AGT/10 version (DSPLY version 1).

A. Control Character Definitions and Values

The available control characters are the following:

7-Bit Code	Mnemonic	ADEPT Code	Control Operation
ØØØ	NUL	@ N	No action
Ø1Ø	BS*	@ I	Italics control. Complements the italic state.
Ø11	HT	@ X	Position X. The next 7-bit character specifies the horizontal grid position for succeeding characters.
Ø13	VT	@ Y	Position Y. Similar to Position X but specifies vertical grid position.
Ø12	LF	@ F	Line feed - move beam down to next line.
Ø15	CR	@ C	Carriage return - reset X to position 16_{10}.
Ø21	DC1		Space switch. Complements the vertical spacing state.
Ø22	DC2	@ S	Decrease size of character, if possible.
Ø23	DC3	@ E	Increase size of character, if possible.
Ø34	FS	@ L	Lower line one-half current character height (for subscripts), if possible.
Ø35	GS	@ R	Raise line one-half current character height (for superscripts), if possible.
Ø36	RS*	@ B	Brightness decrease, if possible.
Ø37	US*	@ D	Brightness increase, if possible.

* Not implemented in non-LCG AGT/10 version (DSPLY version 1).

B. Display Character Values

The available Display Characters are the standard 7-bit ASCII codes:

7-Bit Code	Character	7-Bit Code	Character
040	Space	100	@ (@@)
041	!	101	A
042	"	102	B
043	#	103	C
044	$	104	D
045	%	105	E
046	&	106	F
047	' (apostrophe, acute accent)	107	G
050	(	110	H
051	)	111	I
052	*	112	J
053	+	113	K
054	,	114	L
055	-	115	M
056	.	116	N
057	/	117	O
060	0	120	P
061	1	121	Q
062	2	122	R
063	3	123	S
064	4	124	T
065	5	125	U
066	6	126	V
067	7	127	W
070	8	130	X
071	9	131	Y
072	:	132	Z
073	;	133	[(@])
074	< (@()	134	\ (@ /)
075	=	135	]
076	> (@))	136	^ (circumflex) (@ V)
077	?	137	_ (underbar) (@ -)

In addition, the following extended set is available as an option:

7-Bit Code	Character	7-Bit Code	Character
140	` (grave accent)	160	p
141	a	161	q
142	b	162	r
143	c	163	s
144	d	164	t
145	e	165	u
146	f	166	v
147	g	167	w
150	h	170	x
151	i	171	y
152	j	172	z
153	k	173	{
154	l	174	| (vertical bar)
155	m	175	}
156	n	176	~ (tilde)
157	o	177	Space

OPERATION CODE VALUES

Mnemonic	Value	Mnemonic	Value
ELEMENT-GENERATING			
MOVE	1ØØ!H	2DTAB	1ØØØ!H
2DTBL	2ØØ!H	2DTAF	11ØØ!H
DRAW	3ØØ!H	2DINT	12ØØ!H
2DTF	4ØØ!H	2DINF	13ØØ!H
LABLR	5ØØ!H	3DTAB	14ØØ!H
2DLST	6ØØ!H	3DTAF	15ØØ!H
LABL	7ØØ!H	2DROT	16ØØ!H
		2DROF	17ØØ!H
TRANSFORM-SPECIFYING			
SCL	1ØØØØ!H	LDSCL	2ØØØØ!H
ROTX	1Ø1ØØ!H	LDRX	2Ø1ØØ!H
ROTY	1Ø2ØØ!H	LDRY	2Ø2ØØ!H
ROTZ	1Ø3ØØ!H	LDRZ	2Ø3ØØ!H
RXYZ	1Ø4ØØ!H	LDRV	2Ø4ØØ!H
DX	1Ø5ØØ!H	LDX	2Ø5ØØ!H
DY	1Ø6ØØ!H	LDY	2Ø6ØØ!H
DZ	1Ø7ØØ!H	LDZ	2Ø7ØØ!H
DV	11ØØØ!H	LDV	21ØØØ!H
		LDI	212ØØ!H
CONTROL			
NUL	ØØØØØ!H	LDMB	5ØØØØ!H
SAVT	3ØØØØ!H	ORMB	5Ø1ØØ!H
REST	3Ø1ØØ!H	ANDMB	5Ø2ØØ!H
RET	3Ø2ØØ!H	LDSN	54ØØØ!H
IMG	3Ø3ØØ!H	LDLS	55ØØØ!H
LOOP	3Ø4ØØ!H	CJMP'C	56CØØ!H
ENDL	3Ø5ØØ!H	CJSR'C	57CØØ!H
JMP	3Ø6ØØ!H	WJMP'WW	6WWØØ!H
JSR	3Ø7ØØ!H	WJSR'WW	7WWØØ!H
LONG	315ØØ!H		
SHORT	316ØØ!H		

Mnemonic	Value	Mnemonic	Value
VIEWING			
LDW	31ØØØ!H	LWS	312ØØ!H
MVW	311ØØ!H	DEPTH	313ØØ!H
		LDPS	314ØØ!H

FIELD VALUES

CONDITION CODES (C above)

FLAG1	4ØØ!H	PEN	1ØØ!H
FLAG2	2ØØ!H		

WINDOW BOUNDS (WW above)

WBX1	4ØØØ!H	WBX2	2ØØØ!H
WBY1	1ØØØ!H	WBY2	4ØØ!H
WBZ1	2ØØ!H	WBZ2	1ØØ!H

DATA TYPE

IHW (Assumed)	ØØ!H	RHW	1Ø!H
IFW	2Ø!H	RFW	3Ø!H
IRL	6Ø!H	RRL	7Ø!H

ADDRESS MODE

I	1!H	IMED (Assumed)	Ø!H
DIR	4!H	STR	5!H

INTRODUCTION

EDIT is an AMOS library program used to edit alphanumeric information available on punched paper tape, magnetic tape, or disk. With this routine, the console operator can, under typewriter control, perform any desired combination of the following editing functions:

"Kill" (i.e., erase or nullify) the previous contents of the text buffer in memory.

Read one or more pages of a text from punched tape into the text buffer, appending the input to a previously entered page.

Type alphanumeric information into the text buffer, appending the input to a previously entered page.

Punch out the contents of the text buffer.

Insert new lines at any place in any "page" already in the text buffer.

Delete or change specified lines or pages in the text buffer.

Display current text on the CRT scope.

Type out a selected line (or lines) of the text buffer.

Type out selected pages or the entire text buffer, with or without page and line numbers.

Write the entire text buffer onto a "scratch pad."

Replace the text buffer contents from the magnetic tape "scratch pad".

Insert or remove "page marks".

Copy the "scratch pad" to and from the magnetic tape or disk "User Volume."

Using these functions, EDIT simplifies the preparation of error-free alphanumeric text, such as symbolic programs from either typewriter input or previously prepared punched or magnetic tapes or disk.

If the text is of greater length than will fit conveniently into memory, the tape "scratch pad", contained on magnetic tape or disk, can be used to hold an up-to-date version of the text, keeping one "page" at a time in core memory and

updating the "scratch pad" when a user requests an operation that would cause the text in a changed "page" in core memory to be lost, such as bringing a new "page" into core memory. A user may, therefore, edit text of almost any length, subject only to the length of the "scratch pad."

EDIT is a relocatable library program normally residing on the system mass storage until loaded by a user control statement of the AMOS Monitor. Core memory from the end of EDIT to beginning of AMRMX is used as a text buffer. EDIT is loaded and entered using the AMOS control statement (EDIT!), and uses the same PULSE1 PRI switch setting (24ØØØ 77776). Such entry terminates any prior input/output state and establishes the control type-in mode of EDIT. Certain display parameters may be set by calling the SCALE routine in DISP. (See "TEXT DISPLAY".)

INPUT MODES

EDIT has two input type-in modes, control mode and text mode. In control mode, characters typed are interpreted as commands by EDIT; in the text mode, typed input is copied directly into the line buffer and line-by-line into the text buffer. The only exception to this is the "←" (left arrow) or RUBOUT characters (corresponding to the backspace character on the OPC) which in the text mode causes the deletion of the last input character. To go from text mode to control mode, a "←" (left arrow) or RUBOUT (backspace) character is typed immediately following a carriage return or after deleting all characters from a line. While in control mode, certain commands put EDIT in the text mode automatically. The current mode is indicated on the display scope by the words "CNTRL MODE" or "TEXT MODE".

LINE BUFFER

In the text type-in mode, a line consists of all input characters up to and including a CR (carriage return). During input, each line is held in a special line buffer area. A line may contain up to $31Ø_8$ characters.

TEXT BUFFER

In magnetic tape systems, EDIT operates on text in two different modes, the Normal mode and the Tape-Edit mode. In the normal mode of operation, all text is kept in core memory. The core memory buffer then holds one or more "pages" of typescript, separated by "page marks" and terminated by an "end-of-buffer" mark. Each contains one or more "lines" terminated by CR (carriage

return) characters. The text buffer can be loaded from the typewriter, the punched tape reader, or the magnetic tape "scratch pad." Each line within a page is identified by an octal line number, and each page in the text buffer is identified by an octal page number. The disk version of edit always operates in a mode similar to Tape-Edit in which "scratch pad" updates are automatically made.

In the Tape-Edit mode, the "scratch pad" area contains the text in the form of one of the following.

In the Tape-Edit mode, the "scratch pad" area contains the text in the form of one or more magnetic tape records. On magnetic tape systems, the "scratch pad" is contained at the beginning of magnetic tape unit 0. On disk systems, the "scratch pad" is contained on the inner cylinders of whichever disk pack was specified on entry to EDIT. These records each contain a "page" of text. A one-word record, containing the "end-of-buffer" mark is written following the last page. The "current page" (that page currently being referenced) is kept in core memory and may be modified and examined at will. When any operation is requested that would cause that page to be lost, and the current contents of that page are different from the corresponding page on the "scratch pad" (i.e., changes have been made to that page in core memory), an update operation will take place before the requested operation. The action of an update will occur in one of two modes (Single and Multi-Tape) specfied by the user.

In the Single-Tape-Edit mode, a tape update takes place by reading in text from the "scratch pad" until the core buffer is full, writing out a specified number of pages, deleting these pages from the core buffer and reading again. This operation is repeated until all pages from the "changed" page to the last page have been recopied onto the tape.

In the Multi-Tape-Edit mode, a two tape update takes place in which the text on the "scratch pad" is copied onto the user specified "copy tape", substituting the "changed" page for the corresponding page on the "scratch pad" and then re-copying the "copy tape" back to the "scratch pad". This operation requires the use of the "scratch pad" area of two magnetic tapes, but is usually faster than the one tape update, especially with a long text. The disk system version of EDIT does not have a corresponding mode.

In both types of Tape-Edit modes, there is normally only one page in memory, unless "page marks" have been inserted, in which case the next tape update would cause the separate pages to be written out on the "scratch pad." With respect to the current page in core memory, the line references are the same as in the normal mode.

TEXT DISPLAY

The display scope is used primarily to provide instantaneous access to the text. The amount of text displayed is set by the user and certain commands are available to select that part of the text the user desires. Normally the text is centered about the "current line" displaying some number of lines before and after the "current line". A header line at the top of the display gives information about the current status of EDIT. This header is of the form:

mmmMODE PAGEp LINE1 ttt

where "mmm" is the mode "CNTRL" or "TEXT", "p" is the current page number,

In the Control Mode, the current line is indicated by a "→" (right arrow) on the far left side of the display. In Text Mode, the right arrow points to the line currently being inserted into the line buffer. In cases where there is more than one page currently in the core memory text buffer, the first line of each page (not page 1) is indicated by a page mark character "¶" (or "P" for disk and LCG versions) to the left of that line on the display.

Certain parameters for the display can be set up by calling the routine SCALE in DISP. The Command form:

SCALE ((t1, t2, t3, t4, t5), c, b, a)!

typed as a monitor control statement will enter DISP and set the five display tabulation increments to t1, t2, t3, t4, and t5. These arguments represent the number of spaces between each tab setting. If it is desired to change just one or two of the tab settings, leaving any of the four arguments zero will cause their particular setting not to be changed. The normal setting is 7, 15_8, 12_8, and 10_8.

1. A value in "a" will set the character size. The value "a" is S/4 where the S scale factor multiplies the character size increments (normally 4).

2. A value in "b" will set the increments for character and line spacing. The value "b" is of the form D_x in the upper half and D_y in the lower half where D_x and D_y are the character spacing and line spacing increments respectively (normally 400_8 and 1500_8).

3. A value in "c" sets the number of lines displayed before and after the current line.

If in any of the three command forms, one or more arguments ("a", "b", or "c") are omitted, no action will be taken on that argument. For example, desiring only to change the number of lines normally displayed before and after the current line one might type:

SCALE (,7)!

This command is not valid in disk and character generator versions of EDIT.

CONTROL INPUTS

Many EDIT commands require input of a numerical argument. This may be provided as an octal integer, or any of the characters "L", . (period), ' (single quote), or ! (exclamation point), explained below, or it may be any expression (EXP) made up of integers and these special characters, separated by "space" or "+" (plus) and "-" (minus). Certain commands may take two arguments; these appear before the command letter, separated from each other by a comma.

The slash ("/") and all alphabetic command letters except G, H, W, S, X, and Z require a "terminator" before action on the command is taken. This "terminator" is a tab or a carriage return typed immediately after the command letter. If part or all of a command was typed (prior to the "terminator") by mistake, a "?" (question mark) may be typed to wipe out the command. Some commands acknowledge that the operation requested has been accomplished or that action is required by the operator by typing the character BELL.

Characters in addition to octal digits used to represent value are:

L has the value of the number of lines in tne current page of the text buffer; in other words, it is the number of the last in the current page.

. (period) represents the number of the line most recently referred to by an EDIT command.

' (single quote) represents the number of the current page.

! (exclamation point) represents the number of the last page in the text buffer.

If an "=" (equal sign) is typed following an expression, EDIT will type out the expression as an octal number.

In the following descriptions, EXP is the octal value of the expression typed in the given context. Where two expressions are assumed, their values are designated as EXP1 and EXP2, and are separated from each other by a comma. When the operation specified by an EDIT command cannot be performed, a typed error response is produced, indicating the nature of the error encountered. The commands available in EDIT are:

/ EXP1, EXP2 / (list - types out lines EXP1 through EXP2 of the current page and returns to the control input mode. If no "EXP1", sets it equal to EXP2. The operation can be aborted by depressing IC [ø].

A A (Append) - enters the text type-in mode, and adds the subsequent typed input to the end of the current page on a line-by-line basis. This operation will create new text if buffer is empty.

A EXP A (Append from "User Volume") - causes the next file EXP on the "User Volume" to be appended to the current "scratch pad" and returns to control input mode. This command is valid only in Tape-Edit modes, and will cause an update operation if in Tape-Edit mode and the page in memory has been changed. In the Multi-Tape-Edit mode, this operation will cause a copy of the "scratch pad" to be made on the user specified "copy" tape at the end of the operation.

A EXP1, EXP2 A (Append from Tape) - same as above but reads file EXP2 starting at record (i.e., page) EXP1.

B B (Back to AMRMX) - returns control to the AMOS Monitor This command will cause an update operation if in Tape-Edit mode and the page in memory has been changed.

C EXP1, EXP2 C (Change) - deletes lines numbered EXP1 through EXP2 from the current page of the text buffer, enters the text input mode, and inserts the subsequent typed input into the text buffer, on a line-by-line basis in place of the deleted lines. If no "EXP1", sets EXP1 equal to EXP2. The number of lines inserted need not be equal to the number of lines deleted.

D EXP1, EXP2 D (Delete) - deletes lines numbered EXP1 through EXP2 from the current page of the text buffer, changing subsequent line numbers accordingly. If no EXP1, sets EXP1 equal to EXP2. Returns to control input mode.

E E (Single-tape-edit) - sets EDIT to work in the single-tape-edit mode with the "scratch pad" on the tape unit Ø, and returns to control input mode. Invalid in disk version.

E EXP E (multi-tape-edit) - sets EDIT to work in the multi-tape-edit mode with the "scratch pad" on tape unit Ø and the "copy tape" on tape unit EXP (1, 2, or 3) and returns to control input mode. Invalid in disk version.

E EXP % E (multi-tape-edit and copy) - same as above but causes an immediate copy of the "scratch pad" to be made on the "copy tape". Invalid in disk version.

E -E (Normal) - causes EDIT to establish the normal mode where all text is kept in the core text buffer, and returns to control input mode. Invalid in disk version.

F EXP F (Feed Blank Tape) - types a BELL character and enters a three-second wait period to allow the operator time to turn on the punch unit. Then punches EXP number of null characters (feed holes only) on the paper tape punch unit. At the end of the punch operation, a BELL character is again typed and a three-second wait period entered to allow the operator time to turn off the punch unit. Control mode is then entered. This operation may be aborted by depressing IC [Ø].

G EXP G (Advance Display) - causes the "current line" to be advanced by EXP. If no EXP or if EXP = Ø, advances by number of lines set by user.

G EXP1, EXP2 G (Advance and Set) - causes "current line" to be advanced by EXP2 and sets the number of lines, advanced by "G" along or backspaced by "H" alone, to EXP1.

H EXP H (Backspace Display) - causes the "current line" to be backspaced by EXP. If no "EXP" or if EXP = Ø, backs up display by number of lines set by user.

I EXP I (Insert) - enters the text type-in mode, and inserts the subsequent typed input into the text buffer, on a line-by-line basis, before line no. EXP of the current page. No material is lost from the text buffer, although line numbers are changed to accommodate the insertion.

J EXP J (Jump to Page) - sets current page number = EXP, and returns to control input mode. If no EXP, set page 1 current. This command will cause an update operation if in Tape-Edit mode and the page in memory has been changed.

K EXP1, EXP2 K (Kill) - deletes pages EXP1 through EXP2 from the text buffer, and returns to the control mode. If no EXP1, sets it equal to EXP2. If EXP2 is negative, deletes the entire text buffer. For example, -K "Kills" the entire text buffer. This command will cause an update operation if in Tape-Edit mode.

M -M (Memorize) - if EDIT is in the normal mode, writes the entire text buffer on the "scratch pad" area of tape Ø, and returns to control input mode. This command will cause an update if in the Tape-Edit mode and the page in memory has been changed, following which the text on the "scratch pad" is copied onto the end of the user specified "User Volume" and the file number typed out.

N EXP N (Next) - increments the current page no. by EXP, and returns to the control input mode. If no EXP, increments by 1. This command will cause an update operation if in Tape-Edit mode and the page in memory has been changed.

O O (Omit) - deletes the page mark from the end of the current page and returns to the control input mode.

P EXP1, EXP2 P (Punch) - causes EDIT to type a BELL character and enter a three-second wait period to allow the operator time to turn on the punch unit. Pages EXP1 through EXP2 are then punched in the following form. As the page printer operates simultaneously with the punch unit, the punch operation will also generate a page listing of the selected text, and will generate a paper tape which can be listed as off-line as well as read-in for text input by EDIT. Tab characters in the text cause the punching of a Horizontal Tab character (non-printing) followed by enough SPACE characters to position the printed output to the next tab setting. After the Carriage Return character at the end of each line, a Line Feed character is punched; and after the last line of each page, enough line feed characters are punched to position the paper to the next page (11 inch pages).

After each page, sixteen null characters are punched to signify the end-of-page. Following the last page, an End of Message character (300_8) is punched. A three-second wait loop is then entered to allow the operator time to turn off the paper tape punch unit. If no "EXP1" is given, it is set equal to "EXP2". If "EXP2" is also not given, the entire text buffer (entire "scratch pad" if in Tape-Edit mode) is punched. This command will cause an update operation if in Tape-Edit mode and the current page has been changed. This operation can be aborted by depressing IC [ø].

Q EXP1, EXP2 Q (Annotate) - causes text to be appended to lines EXP1 through EXP2. The operation is as follows: Line EXP1 is inserted into the line buffer replacing the carriage return by a TAB. EDIT then enters "text" mode and the following typewriter input until a carriage return is appended to the line. The line is then re-inserted into the text buffer. The next line is inserted in the same manner and this operation is continued through the EXP2. If no "EXP1", sets it equal to EXP2. If no "EXP2", causes annotation of the entire page. Lines containing only a carriage return (CR) are not annotated.

R R (Read) - types a BELL character and waits for the operator to turn on the paper tape reader unit. The input paper tape is then read and appended to the contents of the text buffer ("scratch pad" if in Tape-Edit mode) until an End-of-Message character [CTRL C (300_8)] is read or until IC[0] is depressed. The input text format is compatible with that generated by the punch operation. Line Feed characters are ignored. Sequences of null characters (only feed holes) signify the end of a page. Horizontal tab characters cause the input of a TAB character and all following SPACE characters are ignored until the occurrence of a non-SPACE character. At the end of the operation, a BELL character is typed and a three-second wait period entered to allow the operator time to turn off the reader unit. This command causes an update operation if in Tape-Edit mode and the current page has been changed. At the end of this operation, a copy of the "scratch pad" will be made on the user specified "copy" tape if in Multiple-Tape-Edit mode.

S EXP S (Set current line) - causes the EXP to be set as the current line. If no "EXP" or if "EXP" = ø, sets current line = 1.

T T (Tag) - establishes "tag" mode, and returns to the control input mode. In the tag mode, each line of text output on the typewriter is labelled with its octal page number and line number separated by a period.

T -T (Untag) - establishes the normal "no tag" type-out, and returns to the control input mode.

U EXP1, EXP2 U (Unload) - types out pages EXP1 through EXP2, and returns to the control input mode. At the end of each page, enough Line Feed characters will be typed to position the paper in the typewriter to the next page. If no "EXP1", sets it to equal EXP2. If no "EXP2", types out the entire text buffer (the entire "scratch pad" if in Tape-Edit mode). This command will cause an update operation if in Tape-Edit mode and the page in memory has been changed. The operation can be aborted by depressing IC [ø].

V EXP V (Divide) - inserts a page mark before line number EXP of the current page and returns to the control input mode.

W W (Reset Vertical) - sets the number of lines displayed before and after the "current line" to the value set by argument "c" in the setup entry from AMRM.

W EXP W (Set Vertical) - sets number of lines displayed before and after the "current line" to be EXP.

W EXP, W (Set Character Size) - sets the character size to EXP (1, 2, or 3) on disk and character generator versions of EDIT.

W EXP, W (Set Horizontal) - sets the horizontal display cutoff to full screen if EXP = 0 or to half screen if EXP $\neq$ 0 on non-disk and non-character generator versions of EDIT.

W EXP1, EXP2 W (Set Character Size or Horizontal, and Set Vertical) - sets the character size or the horizontal display cutoff, as explained above, to EXP1, and sets the number of lines displayed before and after the "current line" to EXP2.

X X (Backspace Half-Frame) - backspaces the "current line" one-half frame (i.e., the top line displayed becomes the "current line").

Y -Y (Yank) - kills (deletes) the entire text buffer and reads into the text buffer from the "scratch pad" area on mag tape unit ∅, and returns to the control input mode. This command is only valid in the normal mode.

Y EXP, - Y (Yank) - copies file no. EXP from the specified "User Volume" to the "scratch pad", makes a copy of this on the "copy tape" if in Multi-Tape-Edit mode, and returns to the control input mode. This command is only valid when in the Tape-Edit mode.

Z Z (Advance Half-Frane) - advances the "current line" one-half frame (i.e., the bottom line becomes the "current line").

BACKSPACE or

RUBOUT or

← (BACK ARROW)

When entering text, will delete the last character from the current line, but if the line is empty, no further text will be accepted and subsequent characters will be taken as edit commands. When not entering text, the current line will be typed out and the reference pointer advanced.

NOTE: The commands "G", "H", and "Z" can cause the "current line" to move across the page boundaries in Normal mode but not in Tape-Edit mode.

< EXP < (¢ on the OPC) Set "User Volume" - causes the "User Volume" to be specified as tape unit EXP (EXP = 1, 2, or 3) or disk volume on disk systems where EXP is of the form "pvv_8" with p = pack (0 - 7) and vv = volume (1 - 40_8).

SUMMARY OF EDIT COMMANDS (A FUNCTIONAL BREAKDOWN)

1. To enter the editor from the AMOS Monitor:

EDIT!	Enters editor (non-disk systems).
EDIT1!	Enters editor without storing End of E-Text Mark (non-disk systems).
EDIT2!	Enters editor without clearing the change flag (non-disk systems).
EDIT(n)!	Enters editor and sets "scratch pad" on disk pack n (disk systems only).

To enter or leave the Editor in the Editor:

B	Returns to the AMOS Monitor

2. Other Monitor calls: (non-disk and non-character generator systems).

Scale ((T1, T2, T3, T4, T5), C, B, A)!

Sets Tab increments to T1 through T5. "C" is the number of lines displayed before and after the current line. "B" is the X and Y character spacing with X in left and Y in right half of word. "A" is the character scale (relative to "4") with X in the left and Y in the right half of the word. Each of the four main arguments is optional.

SNAME ("TITLE")!

Sets the title of the output file (and all text records written to title).

3. To look around while in the Editor

One page moves:	G	Advance Display
	H	Backspace Display
	S	Set current line
	X	Backspace Half-Frame
	Z	Advance Half-Frame
	W	Set and reset width of display and no. of line (and size for character generator and disk systems.)
Other page moves:	J	Jump to page
	N	Increment page number
4. To enter text:	A	Append
	I	Insert before current line

		Q	Annotate to end of line
		R	Read paper tape
		Y	Yank (Read from "user" mag tape)
5.	To change text:	C	Change (Delete and enter text)
		D	Delete
		K	Kill
6.	Paging	O	Omit page mark
		V	Divide (insert page mark)
7.	Output text:	F	Feed blank tape
		M	Memorize onto mag tape
		P	Punch paper tape
		U	Unload
8.	Set mode:	E	Establish tape edit mode
		T	Establish tag mode (line numbers on printout)

adage

3EDIT

AMOS Display Text Editor (Version 3)

Programmer's Reference Manual

June 1968

PROPRIETARY NOTICE

TRADEMARK NOTICE

TABLE OF CONTENTS

INTRODUCTION

3EDIT, version 3 of the AMOS system program, is a Display Text Editor designed for operation on the AGT/10, 4K of memory, and MTP5 or MTP8 tape unit(s). 3EDIT provides the user with the capability of creating and maintaining a text file on magnetic or paper tape. 3EDIT holds one page at a time in its text buffer, allowing insertion, deletion, and appending of new text.

COMMANDS

Control input to 3EDIT, version 3, is by a series of typed commands, consisting in general of a command letter preceeded by from zero to three arguments separated by commas. In addition, most alphabetic letter commands require their termination by the typing of a carriage return character following the command.

Arguments for the commands consist of expressions made up of terms separated by the operators "+" or SPACE (arithmetic plus), or "-" (arithmetic minus). Valid terms are octal integers and certain special characters having the values as described below.

. (period)	has the value of the number of the "current line" (described later).
$ (dollar sign)	has the value of the number of the last line on the page.

MODES

Teletype input to 3EDIT may occur in one of two modes. The first, Control Mode (represented on the display by the symbol CNTRL), signifies that typed console input is interpreted as commands to 3EDIT. The second, TEXT Mode, causes characters typed on the console (with certain exceptions described later) to be inserted into the text buffer.

A third mode, READ Mode, is in effect when paper tape is being read into the text buffer. Certain commands given in CNTRL Mode place 3EDIT into the TEXT or READ Mode. CNTRL Mode is re-established when the specified operations indicated by the command are completed.

DISPLAY FORMAT

The display consists of a header line indicating the current mode and current line number, and a text display of up to 21_{10} lines from the text page. The center line is indicated by a right arrow character at its left. The line number of the center line is used as the "current line" number. Certain special commands typed in CNTRL Mode affect the "current line" number. These commands are as follows:

G	Advance current line by 1.
H	Decrease current line by 1.
Z	Advance current line by 10_{10}.
X	Decrease current line by 10_{10}.
expS	Set current line to line exp.

If in advancing the current line number there are not enough lines left in the page, the last line will be set as the current line. If in decreasing the current line number there are not enough lines available to move backward, line one will be set as the current line number.

These display commands are special in that they require no carriage return "terminator" to initiate their action.

In the TEXT and READ Modes, a blinking cursar character (down-arrow) is also present to indicate the position of the next character to be inserted into the text buffer.

EDITING COMMANDS

A. Delete Line(s)

expD	causes line exp to be deleted from the text buffer. Subsequent lines are renumbered accordingly.
exp1, exp2D	causes lines exp1 thru exp2 to be deleted from the text buffer.

B. Text Insertion

I	causes entry into TEXT Mode and typed input to be appended to the end of the text buffer.
expI	causes entry into the TEXT Mode and typed input inserted prior to line exp.

C. Line Modification

expM	causes entry into the TEXT Mode for subsequent insertion of typed input with the cursar character positioned just before the first character in line exp.

D. Special Format Input Characters

Certain special characters may be used in the TEXT Mode. These characters and their respective operations are described below.

BELL	end operation and return to CNTRL Mode.
RUBOUT	deletes from the text buffer the character just preceeding the cursar.
control "R"	moves the cursar backward (reverse) skipping the preceeding character.
control "E"	deletes from the text buffer the character immediately following the cursar.
control "F"	moves the cursar forwards skipping the next character.

For readability, the cursar character in TEXT Mode initiated by the "I" command is displayed with a ficticious carriage return (end-of-line) following it. This carriage return is omitted when CNTRL Mode is re-established.

TTY AND PAPER TAPE INPUT/OUTPUT COMMANDS

A. Feed Blank Tape

expF	causes exp number of null characters to be punched on the paper tape punch.

3EDIT will initially type two BELL characters and enter a three second wait period for the operator to turn on the punch unit. Another three second wait period will be entered at the end of the operation allowing the operator time to turn off the punch unit.

B. List Line(s)

expL	types line exp on the TTY printer.
exp1, exp2L	types lines exp1 thru exp2 on the TTY printer unit.

C. Punch Line(s)

expP	punches line exp on the punch unit.
exp1, exp2P	punches lines exp1 thru exp2 on the punch unit.

In both forms of the punch command, 3EDIT will type two BELL characters and enter a three second wait period to allow the operator sufficient time to turn on the paper tape punch unit. When the desired text has been punched, 3EDIT will cause the punching of 20_8 null characters and again enter the three second wait period for the operator to turn off the punch unit.

D. Read Paper Tape

R	reads text from the paper tape reader and appends it to the end of the text buffer.
expR	reads text from the reader and inserts it prior to line exp.

In both forms of the read command, 3EDIT types two BELL characters to notify the operator to turn on the reader. Initial null characters are ignored on the reader. Initial null characters are ignored prior to the first non-null character on tape. The first null character after valid input is taken as the end-of-text indicator and the read operation is terminated by the typing of three BELL characters

and the entry into a three second wait period to allow time to turn off the tape reader.

During the read operation, all line feed characters are ignored and all space characters ignored if they occur immediately following a horizontal tab character. The RUBOUT character, if read, will cause the deletion of the previous character and successive RUBOUT characters may be used, deleting characters back to the beginning of the line.

E. Typeout Octal Value

exp=	causes the typing of the lead zero suppressed octal integer having the value of exp. This command requires no carriage return "terminator".

MAGNETIC TAPE INPUT/OUTPUT COMMANDS

A. Select Tape Drive

expT	sets the tape drive for subsequent tape operations to be exp.

B. Write Text on Tape

exp3W	writes the entire text buffer on magnetic tape according to format exp3.
exp1, exp2, exp3W	writes lines exp1 thru exp2 on magnetic tape according to format exp3.

Argument exp3 is a code indicating the desired location on tape of the output record. The permissable codes are as follows:

0	appends record to "scratch pad".
1	starts new "scratch pad" (i.e., record 1)
2	appends to last file on tape.
3	starts new file in file area of tape.

At the end of the write operation, the file and record numbers of the record written are typed.

C. Read Text from Magnetic Tape

exp1, exp2Y	reads record exp1 of file exp2 and appends the text read to the end of the text buffer.
exp1, exp2, exp3Y	reads record exp1 of file exp2 and inserts the text read prior to line exp3.

ERROR MESSAGES

?	Illegal character typed in CNTRL Mode.
NOT ENOUGH MEMORY.	Text buffer storage exceeded in either "R", "I", or "M" commands. Operation is terminated.
RECORD TOO LONG.	Text buffer exceeded in reading from magnetic tape. Input text is deleted and read operation terminated.
RECORD NOT FOUND.	Specified record is not found on tape. Read operation is terminated.
ILLEGAL RECORD TYPE.	Specified record is not type "ATEXT". Read operation is terminated.
ARGUMENT ERROR.	Illegal line number for command or multiple arguments out of sequence or not enough arguments for specified command. The command operation is terminated.

INTRODUCTION

FCRD is a card reader routine used by AFORT to implement the command READ on unit 52 (see OBJPK/PRM). FCRD is loaded by OBJPK and is called directly by OBJPK. It makes use of CDRDR to implement the routine. CDRDR is called by FCRD, Version 2.

VERSIONS

FCRD may be assembled in one of two versions, specified at assembly-time. Version 1 is used when the card reader is not available to the system. Version 2 is used with a single card reader.

LOADING

FCRD is a relocatable program which is loaded by OBJPK whenever OBJPK is loaded.

UNIT ERROR

If an AFORT program specifies reading when the card reader is not available or writing on the card reader, a UNIT ERROR will occur.

AFORT CARD READER ROUTINE

Read a Hollerith Formated Card

The calling sequence:

```
JPSR        9CD1        [Entry
              .         [Normal Return
```

causes a single card to be read if $9IO = 0. The input is translated from AMOS code provided by CDRDR to AFORT internal code and put in the buffer at $9OB. In normal use $9IO and $9OB exist in OBJPK.

INTERNAL CODES

AMOS code is translated to AFORT code as follows:

TTY CHAR.	AFORT CODE	AMOS CODE	TTY CHAR.	AFORT CODE	AMOS CODE
SPACE	00	40	W	40	67
0	01	20	X	41	70
1	02	21	Y	42	71
2	03	22	Z	43	72
3	04	23	↑	44	76
4	05	24	@	45	75
5	06	25	%	46	01
6	07	26	]	47	02
7	10	27	I	50	51
8	11	30	J	51	52
9	12	31	K	52	53
...	13	32	L	53	54
...	14	00	M	54	55
...	15	03	N	55	56
...	16	11	...	56	13
...	17	12	...	57	14
A	20	41	+	60	10
B	21	42	-	61	35
C	22	43	*	62	05
D	23	44	/	63	37
E	24	45	.	64	36
F	25	46	(	65	16
G	26	47	)	66	17
H	27	50	,	67	34
O	30	57	=	70	33
P	31	60	&	71	04
Q	32	61	:	72	06
R	33	62	...	73	07
S	34	63	$	74	73
T	35	64	#	75	74
U	36	65	...	76	77
V	37	66	C/R	77	15

NOTE

"..." indicates that this internal code is unused in the AFORT system.

INTRODUCTION

FCTE is a subroutine written in ADEPT source language for the AMOS library which uses the EAU to obtain a single valued function of one (positive) independent variable, using table look-up and quadratic interpolation in a 65_{10} word table.

HARDWARE REQUIREMENTS

Any configuration of DPR2.

SOFTWARE REQUIREMENTS

The function table supplied by the calling program (such as SNCSA).

STORAGE AND TIMING

FCTE occupies less than 100_8 words of storage and requires 175 microseconds for execution.

VERSIONS

VERSION1	FCTE	Uses EAU of DPR2 for digital arithmetic.
VERSION(s)	FCTA	Same as FCTE, but using ambilogical arithmetic

USE

Calling Sequence:

AR	X (+ sign in bit Ø, value in bits [1-14])
L	JPSR $FCTA
L+1	Address of TBL (Function table)
L+2	Next instruction

Result:

AR F(x) (SIGN in bit Ø, value in bits [1-14]

where the table is of the following format:

LOCATION	BITS [ø-14]	BITS [15-29]
TBL+2J-2	ø	F(X(J))
TBL+2J-1	4(F(J+1)-F(J-1))	4(F(J+1)-2F(J)+F(J-1))

for J = 1, 2, 3,...33

where X(J) = (J-1)/32 and F(J) = F(X(J))

GENERAL

FDSK is a collection of FORTRAN callable subroutines for reading and writing data on Adage Disk Memory, DMS2.

ATTACHING DISK VOLUMES TO FORTRAN UNIT NUMBERS

Four FORTRAN I/O unit numbers (21, 22, 23, 24) are assigned to disk allowing a total of four separate I/O paths open at any one time. An entry in FDSK is provided to attach a (logical) unit number to a selected disk volume and disk pack as follows:

```
CALL SDVOL (NUNIT, NPACK, NVOL, IBUFF)
```

where NUNIT is 21_{10}, 22_{10}, 23_{10}, or 24_{10}; NPACK is the disk pack number (0 through 7); NVOL is the disk volume number (1 through 32_{10}), and IBUFF is a FORTRAN vector of length 208_{10}.

DISK FILE CONTROL

Start Output File — CALL SDOUT (NUNIT)

The above call causes FDSK to initiate a file output operation on the pack and volume associated with NUNIT. The file which is written by subsequent FORTRAN WRITE statements of this unit will be of type "DATA," have title "FDATA," and have the date, version, and revision as set by the last AMRMX date setting and VERSION statement and operation.

Start Input File — CALL SDIN (NUNIT, IEOF)

The above call causes FDSK to search the associated volume and instate the first file of type "DATA" to be read by subsequent FORTRAN READ statements. At this time, the integer variable specified by the IEOF argument is set to +0. When a subsequent READ statement is given and the file contains no more records, the integer variable is set to -1 (e.g., the READ statement following the READ statement which input the last record of the file causes the variable to be set to -1).

Close File CALL CLDIO (NUNIT)

The above call causes the "closing" of the file currently being input or output on this unit. All files "opened" by calls to SDOUT or SDIN should be "closed" by a call to this routine for proper operation of the disk driver routines. Note that the CLDIO call on an input file followed by an SDIN call on the same unit is equivalent to "rewinding" and starting the file over.

PROGRAM RESTRICTIONS

The following errors will cause run-terminating I/O error stops:

1. There may never be more than one unit "attached" to a given pack and volume at any one time.
2. A Unit must be "attached" to a pack and volume before any SDOUT, SDIN, or CLDIO calls are given.
3. An "attach" operation on a unit which is already attached and contains an open input or output file is illegal. Files must be closed before any subsequent "attach" operations or new input or output selection (on the same unit) may take place. Before FDSK is first called, all units are closed and unattached.

SUMMARY OF CALLS

Attach Unit:	CALL	SDVOL	(NUNIT, NPACK, NVOL, IBUFF)
Start Output:	CALL	SDOUT	(NUNIT)
Start Input:	CALL	SDIN	(NUNIT, IEOF)
Close Unit:	CALL	CLDIO	(NUNIT)

adage

FILE I/O

AGT DISK FILE INPUT/OUTPUT

Programmer's Reference Manual

Revision B

August 1969

TABLE OF CONTENTS

INTRODUCTION

The File I/O Package is a set of subroutines for filing and retrieving information to and from disk storage. It implements facilities for allocating the storage on disk and for maintaining the necessary indexes and directories for efficient access and input/output.

All standard system software referencing online mass storage on disk configurations use the structures, formats, and conventions of the File I/O Package. The package is available as standard in disk versions of the AMOS Monitor.

DISK FILE I/O

The Disk File I/O package provides the means for creating "volume" definitions. The disk areas assigned under these definitions are utilized to hold user and system information files.

A. Volumes

Programmed processes and procedures using standard File I/O facilities may have their input/output streams assigned to selected volumes. Efficient calls are provided for:

1. Randomly accessing any file in a volume.
2. Appending new files to a volume.
3. Deletion of files anywhere within a volume.

B. Files

Files are the units in which collected information is added to, read, or deleted from disk storage with the File I/O package. Each file is assigned fields for containing the following ID information:

1. An alphanumeric "title."
2. A sequence number to establish order in volume.

3. The type of information contained in the file.
4. An associated date, version, and revision level.
5. Flags defining:
   a. Whether the file is further subdivided into records, and
   b. if the file ID contents (lines 1 and 4) do not follow standard formats.

The above information for all files in all volumes, along with the actual disk location, is maintained in a File Directory. This allows easy searching for such identifying information without having to access the actual files.

C. Records

Files may be further subdivided into records which later can be processed serially.

FILE I/O FACILITIES

The calls implemented by the File I/O package are:

1. The Define group which provides for the creation and alteration of a disk pack's volume definitions.
2. The Open group that controls the establishment of selected volumes for use in subsequent disk operations.
3. The Look-Up group performs serial sequencing over file ID fields and instating of files for subsequent I/O accesses.
4. The I/O group directs the actual transfer of information between disk file and core storage.

A. Define

The following calling sequences govern disk allocation. For more information on arguments, see the section on arguments.

INITIALIZE

Destroys all information on the selected pack, regenerates disk format and addressing information, creates an initial empty volume and file directories.

This prepares any new or used pack for use under File I/O.

```
JPSR    $8IT    [Call
        BUSY    [Instruction executed if unit busy
        DISK    [Pack address:  unit (0-3) pack (0-1) = (0-7)
        ADR     [Address of packed 19 character or less string
                 for pack identification
        ERR     [Instruction executed if pack unavailable
        DONE    [Instruction executed when done
        .       [Location returned to once the operation has
                 been started
```

CREATE

Establishes a new volume as available on a pack for use in holding files.

```
JPSR    $8CR    [Call
        BUSY    [Instruction executed if unit busy
        DISK    [Unit and pack (0-7)
        VOL#    [ID of volume being defined (1-32)
        #CILS   [No. of cylinders to be allocated to new volume
        ERR     [Instruction executed if operation is not possible
        DONE    [Instruction executed when operation is complete
        .       [Location returned to once operation has been
                 started
```

PURGE

Removes a volume from a pack, destroys all contained files and their information, releases all disk storage which has been allocated to the volume. Major reallocation of disk contents may be undertaken by this operation.

```
JPSR    $8PU    [Call
        BUSY    [Instruction executed if unit busy
        DISK    [Unit and pack (0-7)
        VOL#    [Volume to be dropped
        ERR     [Instruction executed if operation is not possible
        DONE    [Instruction executed when operation is complete
        .       [Location returned to once the operation has
                 been started
```

CHANGE

This call increases or decreases the amount of disk storage allocated to a given volume; no information is destroyed. Major reallocation of disk contents may be undertaken by this operation.

```
JPSR    $8CH    [Call
        BUSY    [Instruction executed if unit busy
        DISK    [Unit and pack (0-7)
        VOL#    [Volume to be expanded
        ±#NEWCIL [Number of cylinders to be added to (or sub-
                  tracted from if negative) current allocation
        ERR     [Instruction executed if operation is not possible
        DONE    [Instruction executed when operation is complete
        .       [Location returned to once the operation has
                 been started
```

REMOVE

This call removes one or several files from a selected volume and deletes the file directory information for this (these) file(s). Reallocation of the selected volume's contents may be undertaken by this operation.

JPSR	$8RE	[Call
	BUSY	[Instruction executed if unit busy
	DISK	[Unit and pack (0-7)
	VOL#	[Volumes containing file(s) to be deleted
	SEQNO	[File to be deleted or pointer to list of files (-1) with number of files to be deleted in upper portion of word (multiple file list must be in increasing order)
	ERR	[Instruction executed if operation is not possible
	DONE	[Instruction executed when operation is complete
	.	[Location returned to once operation has been started

B. Open

These calls govern the establishing of selected volumes for use in subsequent disk operations. Disk operations refer to their volume by a NAME which is set when the volume is opened by a SELECT operation. For each volume opened, the caller must furnish a buffer area in which two sectors may be held (208 words).

SELECT

This call opens the specified volume for accessing and I/O. A NAME word is set to reference the volume.

JPSR	$8SE	[Call
	BUSY	[Instruction executed if unit busy
	NAME	[Address of cell for referring to volume
	DISK	[Unit and pack
	VOL#	[Volume to be opened
	BUFF	[Address of 208-word Buffer area
	ERR	[Instruction executed if operation is not possible
	DONE	[Instruction executed when operation is complete
	.	[Location returned to once the operation has been started

UNSELECT

This call closes and detaches the specified volume. The name cell referring to it and the buffer area it used are then no longer required or used. This call also closes any open output files with the same file name.

```
JPSR    $8US    [Call
        BUSY    [Instruction executed if unit busy
        NAME    [Cell referring to volume (or 0 if all are to be
                 closed)
        ERR     [Instruction executed if operation is not possible
        DONE    [Instruction executed when operation is complete
        .       [Location returned to once the operation has been
                 started
```

C. Look-Up

Look-Up calls are used to sequence through an open selected volume. Each call breaks out of the file directory the file ID information pertaining to the current file reached. Refer to NAME in section on arguments for file ID format. Any file reached in this manner is then accessible for read or skip operations. In all of these calls, the user's NAME cell is set to point to the file ID block.

FIRST

This call accesses the first file of the specified volume, instating its ID information and preparing it for reading.

```
JPSR    $8FI    [Call
        BUSY    [Instruction executed if unit busy
        NAME    [Address of cell referencing volume
        ERR     [Instruction executed if operation is not possible
        DONE    [Instruction executed when operation is complete
        .       [Location returned to once the operation has
                 been started
```

NEXT

This call accesses the next sequential file in the referenced volume (if any) and instates its ID information. The accessed file is then ready for reading.

```
JPSR    $8NE    [Call
        BUSY    [Instruction executed if unit busy
        NAME    [Address of cell referencing the volume
        ERR     [Instruction executed if operation is not possible
        DONE    [Instruction executed when operation is complete
        .       [Location returned to once the operation has
                 been started
```

LAST

This call accesses the final file in the referenced volume and instates its ID information. The accessed file is then ready for reading.

```
JPSR    $8LA    [Call
        BUSY    [Instruction executed if unit busy
        NAME    [Address of cell referencing the volume
        ERR     [Instruction executed if operation is not possible
        DONE    [Instruction executed when operation is complete
        .       [Location returned to once the operation has been
                 started
```

PREVIOUS

This call accesses the previous file (if any) of an open volume and instates its ID information. The accessed file is then ready for reading.

```
JPSR    $8PR    [Call
        BUSY    [Instruction executed if unit busy
        NAME    [Address of cell referencing the volume
```

	ERR	[Instruction executed if operation is not possible
	DONE	[Instruction executed when operation is complete
	.	[Location returned to once the operation has been started

D. Input/Output

The following calling sequences are used to process information pertaining to the currently accessed file of the specified open volume.

SKIP

This call is used to read past information in a file without transferring it to core storage.

JPSR	$8SP	[Call
	BUSY	[Instruction executed if unit busy
	NAME	[Address of cell referencing the volume
	#WRDS	[Address of number of words to be skipped (Address of Ø value if to skip 1 record)
	ERR	[Instruction executed if operation is not possible
	DONE	[Instruction executed when operation is complete
	.	[Location returned to once the operation has been started

OUTPUT

This call is used to start a new file at the end of the current volume. No data is written in the file as a result of this call. The "new ID" argument refers to a three-word block giving ID information for the new file. The first word has two flag bits indicating whether the file is structured into records, and if the next two ID words have standard or user assigned contents. The remaining two words can be used to hold a file title, type, version, revision, and date under standard format use. This call also closes any current open output files with the same file name. (See section on Table Formats for layout.)

```
JPSR    $8OU    [Call
        BUSY    [Instruction executed if unit busy
        NAME    [Address of cell referencing the volume
        NEWID   [Address of new ID block
        ERR     [Instruction executed if operation is not possible
        DONE    [Instruction executed when operation is complete.
        .       [Location returned to once the operation has
                 been started
```

APPEND

This call is used to add information to the last file of the specified volume which has been previously started by an "OUTPUT" call. If the file is structured in records, each APPEND call adds one more record to the file. Each record has associated with it (first word in its disk image) an origin and a Length field. The origin will normally be set to the core location given in the left half of the word count argument.

```
JPSR    $8AP    [Call
        BUSY    [Instruction executed if operation not possible
        NAME    [Address of cell referencing the volume
        LOC     [Address of cell pointing to buffer to be written
        #WDS    [Address of cell of number of words to write (and
                 origin if record format)
        ERR     [Instruction executed if operation is not possible
        DONE    [Instruction executed when operation is complete
        .       [Location returned to once the operation has been
                 started
```

CANCEL

This call will cause the current OUTPUT or APPEND operation to be aborted. It would be used to terminate the writing of a new file before the file has been completed. The file's contents and directory information written to this point will be dropped.

```
JPSR    $8CA    [Call
        BUSY    [Instruction executed if unit busy
        NAME    [Address of cell referencing volume being closed
        ERR     [Instruction executed if operation is not possible
        DONE    [Instruction executed when operation is complete
        .       [Location returned to one the operation is complete
```

INPUT

This call is used to input information from the accessed file of the specified volume. The number of words to be read are specified by an argument. Successful INPUT calls will input successive blocks of information from the file on disk. If there are not enough remaining words in the file, the error word will give the number read (see ERR in section on Arguments).

```
JPSR    $8IN    [Call
        BUSY    [Instruction executed if unit busy
        NAME    [Address of cell referencing volume
        LOC     [Address of cell pointing to input area
        #WDS    [Address of cell giving number of words to read
        ERR     [Instruction executed if operation is not possible
        DONE    [Instruction executed when operation is complete
        .       [Location returned to once the operation has been
                 started
```

INPUT RECORD

This call is used to input successive records from the currently accessed file (in record format) of the specified volume. The #WRDS argument gives the size of the read area. If the record exceeds it, the ERR will be performed. The LOC argument points to the address of the input area. The record read into a specified area will include as its first word the origin and length associated with it. If the LOC argument is set +0, the record will be read into core at the address given by its associated origin and the first word will be the first data word of the record.

```
JPSR    $8IR    [Call
        BUSY    [Instruction executed if unit busy
        NAME    [Address of cell referencing the volume
        LOC     [Address of address-of-input area or +0 if input
                 is at origin
        #WDS    [Address of cell giving maximum read length
        ERR     [Instruction executed if operation is not possible
        DONE    [Instruction executed when operation is complete
        .       [Location returned to once the operation has been
                 started
```

CLOSE

This call terminates the file being written, updates all pointers and directories, and empties all related buffers. A CLOSE is done automatically if a volume being written into is UNSELECTed. A CLOSE operation performed on a file currently being read will cause the scan pointers to be reset, and the file can be read again from the beginning.

```
JPSR    $8CL    [Call
        BUSY    [Instruction executed if unit busy
        NAME    [Address of cell referencing volume being closed
        ERR     [Instruction executed if operation is not possible
        DONE    [Instruction executed when operation is complete
        .       [Location returned to once the operation is complete
```

ARGUMENTS FOR FILE I/O

A. BUSY

Control is returned to this argument whenever the state of the designated disk drive does not permit a required disk operation to be initiated. Control is returned at the same priority level as the File I/O call. If the user chooses to wait until available, BUSY can be set = JUMP .-1.

B. DISK

This argument used to address physical disk packs is composed as follows:

DISK (27-28) = Unit: 0,1,2,3

DISK (29) = Pack: 0,1

C. VOL#

This argument is used as the "physical" volume number within the pack containing it. Each pack may have up to 32 volumes, each assigned a distinct VOL number from 1 to 32.

D. NAME

This argument is used as the "logical" volume address for referencing any "opened" volume in the system. After each Look-Up call, the cell referenced by NAME is set to address the accessed file's ID information in a four-word table as follows:

Word 1 (0)	=	STD: 1 if next two words in standard format
Word 1 (1)	=	RCD: 1 if file composed of standard format records (first word/rec = Origin and Length)
Word 1 (2-14)	=	File #. Sequence Number in volume
Word 1 (15-29)	=	Relative disk address in volume
Word 2 (0-29)	=	Five-character title of file
Word 3 (0-3)	=	Type of file. Standard assigned types are as follows:

0	SYMS	Absolute symbols
1	DATA	Fortran files
2	RELOC	Object programs
3	TRACF	Reactive typewriter forms
4	TEXT	Case Shift Sensitive Text

(continuation)

	5	RLSYM	Relocatable symbols
	6	PRNTR	Off line printer records
	7	ATEXT	Standard text
	10_8	BIN	Absolute file
	11_8	DUMP	Core save file
	12_8 -17_8	TYPE3-TYPE8	(unassigned)

Word 3 (4-8) = Version: numeric from 1 to 31 (0 = none)

Word 3 (9-14) = Revision: one alphanumeric or special character

Word 3 (15-29) = Date: subdivided as follows:

(16-20)	=	Year - 1964
(21-24)	=	Month ≤ 12
(25-29)	=	Day ≤ 31

The second and third words of file ID information need not contain the preceding items; they are available for any special user-use desired. When these two words are used in a non-standard way, the STD flag in word 1 should be set to zero for proper operation of system search and listing routines.

E. DONE

This instruction is executed once the requested File I/O operation has been completed. It is executed at the programmatic interrupt level (PRI level = 15) and must not release it (via a JUMP'I). When executed, the AR register is = 0 and the LC is set to where control MUST be returned to. Thus, DONE may be a subroutine call (JPSR) to a subroutine which returns (without unlocking PRI level) or a single instruction which does not destroy LC (non-JUMP or SKIP), i.e., it can set flags ARMD FL or ARMD'N FL, etc.

F. ERR

This argument is an instruction to be executed if a started File I/O operation can not be successfully completed. It is executed at the programmatic interrupt level and must not release it. When executed, the AR register contains

error-flag settings in the upper half and, where relevant, an address or count in the lower half. The LC is set at a point to which control must be returned in order to restore saved registers and resume execution in the program which was running at the time that the disk operation error was discovered. The same instructions described for DONE apply to ERR.

The errors indicated in AR are as follows:

AR[0] = 0 (Available for setting sign flags)

AR[1] = 0 if error is of the Access or Availability class
1 if error is of the Hardware class

Subsequent bits in the Access or Availability Class are:

AR[2] = Disk Storage Full (8CR, 8CH);
Disk Storage Full and File Canceled (8OU, 8AP)

AR[3] = End of Volume (8NE, 8PR, 8FI, 8LA, 8CH)

AR[4] = End of File (8IN, 8IR, 8SP) - Bits 15-29 give number of words transferred or skipped

AR[5] = End of Buffer (8IR) - Bits 15-29 give number of words remaining in record

AR[6] = Not used

AR[7] = Illegal Volume Number (8SE, 8CR, 8CH, 8PU, 8RE)

AR[8] = Illegal Command (8IR - file not in record format; 8OU - another file outputting on same volume; 8IT - pack 0 initialize requested; 8RE - file list out of order)

AR[9] = Volume Name not Selected (Look-Up, I/O, 8US); Volume Name Already Selected (8SE)

AR[10] = No File Accessed For Volume (8PR, 8NE, 8IN, 8IR, 8SP)

AR[11] = Volume Previously Defined (8CR); Volume Not Defined (8CH, 8PU, 8RE, 8SE)

AR[12] = File Sequence Number not in Volume (8RE)

AR[13] = Four Volume Names Already Selected (8SE)

AR[14] = No Output in Progress (8AP, 8CA)

Subsequent bits in the Hardware Error Class are:

AR[2] = Unit off-line

AR[3] = Read redundancy error after three re-tries

AR[4] = Sector address error

AR[5] = Cylinder address error

AR[6] = Write lockout

AR[7] = Hardware malfunction error. This error should not occur in normal operation. Bits [27-29] of AR give the particular error condition as follows:

$AR[27\text{-}29]_8$

1 Seek error condition when executing home track seek operation.

2 Command error when normal error interrupt occurs. This condition could occur if the disk unit power is dropped while an operation is in progress or if the write protect switch were turned on during a write or write format operation.

3 Error terminate interrupt occurred and no error bits on in the status word.

4 Count error (i.e., no words = 0).

5 "Fault" condition is disk unit.

NEWID

When outputting information to start a new file, this argument is used to provide the file ID information desired. The relevant items for standard ID format are the same as described under NAME. NEWID is the address of a cell pointing to a three-word block. The first word always has the following format:

Word 1 (0)	=	STD
Word 1 (1)	=	RCD
Word 1 (2-29)	=	Not used

The remaining two words of the block may be used by the user for file identification of his choice or may follow the standard file ID format as follows:

Word 2 (0-29)	=	Title
Word 3 (0-3)	=	Type
Word 3 (4-8)	=	Version
Word 3 (9-14)	=	Revision
Word 3 (16-29)	=	Date

If the standard file ID format is followed, the STD bit in word 1 should be set to "1"; otherwise, to "0."

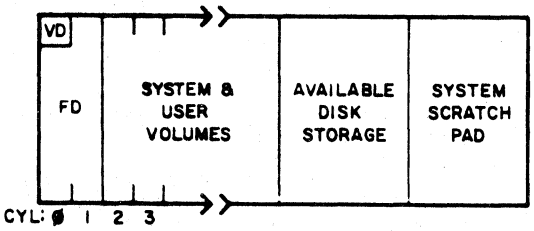
See NAME write-up for further description of contents.

FILE I/O FORMATS AND TABLES

This section will describe some of the information maintained by File I/O and its location.

A. Disk Pack

The standard File I/O disk pack has its cylinders allocated in one of the following ways;

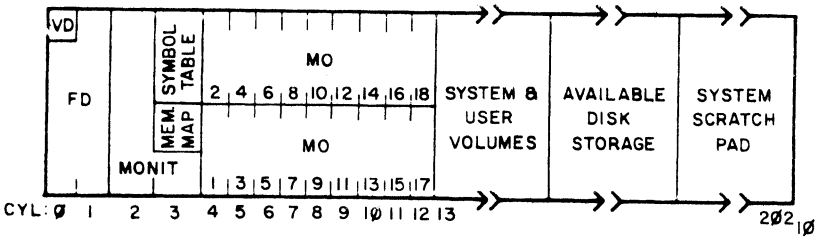


File I/O Disk Layout

Where:

VD	=	Volume Directory
FD	=	File Directory
System Scratch Page	=	System working buffer used for text by editor.

When a pack may be used on pack 0 of unit 0 under standard AMOS software systems, it must have the following format.



File I/O Layout and System

Where:

MO_i	=	Monitor Overlay i
MONIT	=	Absolute Monitor for Bootstrap Loading
Sym. Tab.	=	Current Loader Symbol Table

This format is recommended for all packs at an installation with a single disk drive.

B. Volume Directory

The Volume Directory for each pack is kept in sector 0 of cylinder 0. The first 64 words contain the Volume Definition Table with one 2-word volume ID entry for each of the 32 possible volumes. Words 64 through 95 of the Volume Directory contain the File Directory Occupancy List with a one-word FD Sector

entry for each of the 31 sectors of the File Directory. Words 96 and 97 give the pack's current cylinder allocation. Word 99 contains the pack's creation date. Words 100-103 contain the pack's descriptive string (19 or fewer characters). Refer to Appendix B.

1. Volume ID Entry

The i'th word pair of the Definition Table has the following information for the i'th volume:

Word 1 (0-14)	=	Pointer to FD entry for last file ID of volume
Word 1 (15-29)	=	Pointer to FD entry for first file ID of volume
Word 2 (0-14)	=	Number of cylinders assigned to volume
Word 2 (15-29)	=	First cylinder of volume

2. FD Sector Entry

The i'th word of the FD Occupancy List contains a count of the number of file ID's in the i'th sector of the File Directory, and the pointer to the first free entry in that sector.

3. Cylinder Allocation

The first word of the cylinder allocation contains the "Last Available Cylinder" number (plus 1) which is the current lower bound of the scratch pad.

The second word contains the first available cylinder number which is the end of assigned disk storage.

C. File Directory

The File Directory occupies all sectors of the first two cylinders, except for the Volume Directory. Each of the 31 sectors of the FD contains twenty "bins," each "bin" being five words in length. The bins are used to hold file ID's.

Forward and backward referencing links chain each file ID to the following and preceding file ID's of its volume. All empty bins of an FD sector are linked into a chain of "available" bins.

The first word of each FD sector contains a pointer to the first "bin" on the available chain.

The next twenty five-word blocks are the bins containing either file ID's or available chain links. Each has one of the following formats:

1. Available Bin

Word 1 (0-14)	=	Sector address ($*2^7$) of current FD sector
Word 1 (15-29)	=	Pointer to next available bin (or 0 if last)
Words 2,3,4,5	=	Not used (zero value)

2. File ID

Word 1 (0-14)	=	Pointer to previous file ID of volume
Word 1 (15-29)	=	Pointer to next file ID of volume
Word 2 (0)	=	STD: standard ID flag
Word 2 (1)	=	RCD: record-formatting file flag
Word 2 (2-14)	=	File sequence number within volume
Word 2 (15-29)	=	File's 1st sector address related to 1st cylinder in volume
Word 3 (0-29)	=	Title of file
Word 4 (0-3)	=	Type of file
Word 4 (4-8)	=	Version
Word 4 (9-14)	=	Revision
Word 4 (15-29)	=	Date
Word 5 (0-11)	=	Number of sectors in file
Word 5 (12-29)	=	Number of words in file

The "pointers" used as links in the FD have the following format:

Cylinder	=	Leading 4 bits (can only be = 0 or 1)
Sector	=	Next lower 4 bits (0-15)
Word	=	Low order 7 bits

NOTE

Within the VD and FD sectors, all cylinder addresses and values (i.e., first cylinder in volume and number of cylinders in volume) are actually stored as sectors. That is, the cylinder value appears in the high order 11 bits of the 15-bit value field and the low order 4 bits are zero.

APPENDIX A

SYNOPSIS OF CALLING SEQUENCES FOR FILE I/O PACKAGE

The following calling sequences used by the File I/O package governs disk allocation:

DEFINE		
Initialize	8IT	(BUSY, DISK, ADR, ERR, DONE)
Create	8CR	(BUSY, DISK, VOL#, CYLS, ERR, DONE)
Change	8CH	(BUSY, DISK, VOL#, ±#CYLS, ERR, DONE)
Purge	8PU	(BUSY, DISK, VOL#, ERR, DONE)
Remove	8RE	(BUSY, DISK, VOL#, SEQNO, ERR, DONE)
OPEN		
Select	8SE	(BUSY, NAME, DISK, VOL#, BUFF, ERR, DONE)
Unselect	8US	(BUSY, NAME, ERR, DONE)
LOOK-UP		
First	8FI	(BUSY, NAME, ERR, DONE)
Next	8NE	(BUSY, NAME, ERR, DONE)
Last	8LA	(BUSY, NAME, ERR, DONE)
Previous	8PR	(BUSY, NAME, ERR, DONE)
I/O		
Skip	8SP	(BUSY, NAME, #WRDS, ERR, DONE)
Output	8OU	(BUSY, NAME, NEWID, ERR, DONE)
Append	8AP	(BUSY, NAME, LOC, #WDS, ERR, DONE)
Cancel	8CA	(BUSY, NAME, ERR, DONE)
Input	8IN	(BUSY, NAME, LOC, #WDS, ERR, DONE)
Input Record	8IR	(BUSY, NAME, LOC, #WDS, ERR, DONE)
Close	8CL	(BUSY, NAME, ERR, DONE)

APPENDIX B

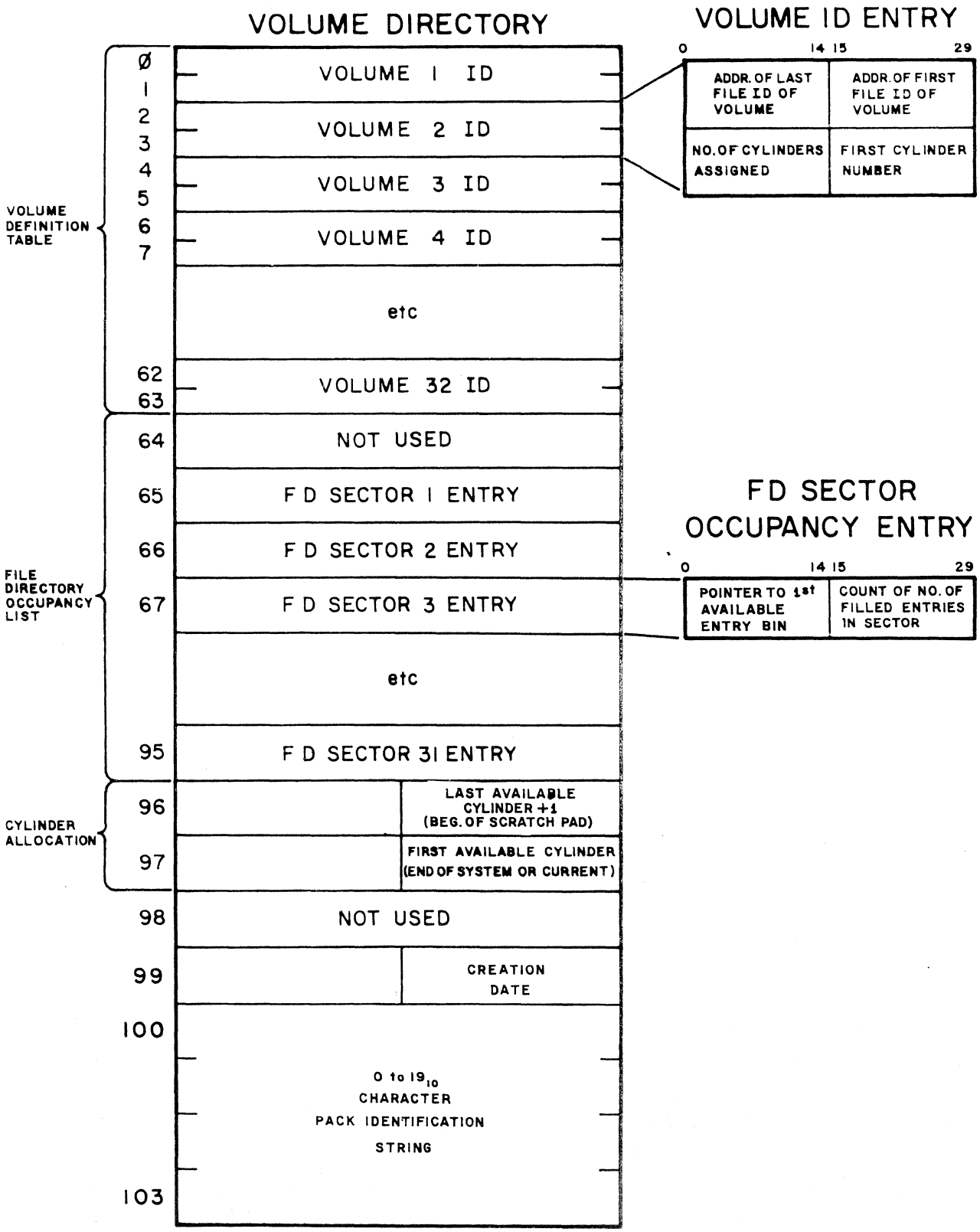
FILE I/O PACK DIRECTORY FORMATS

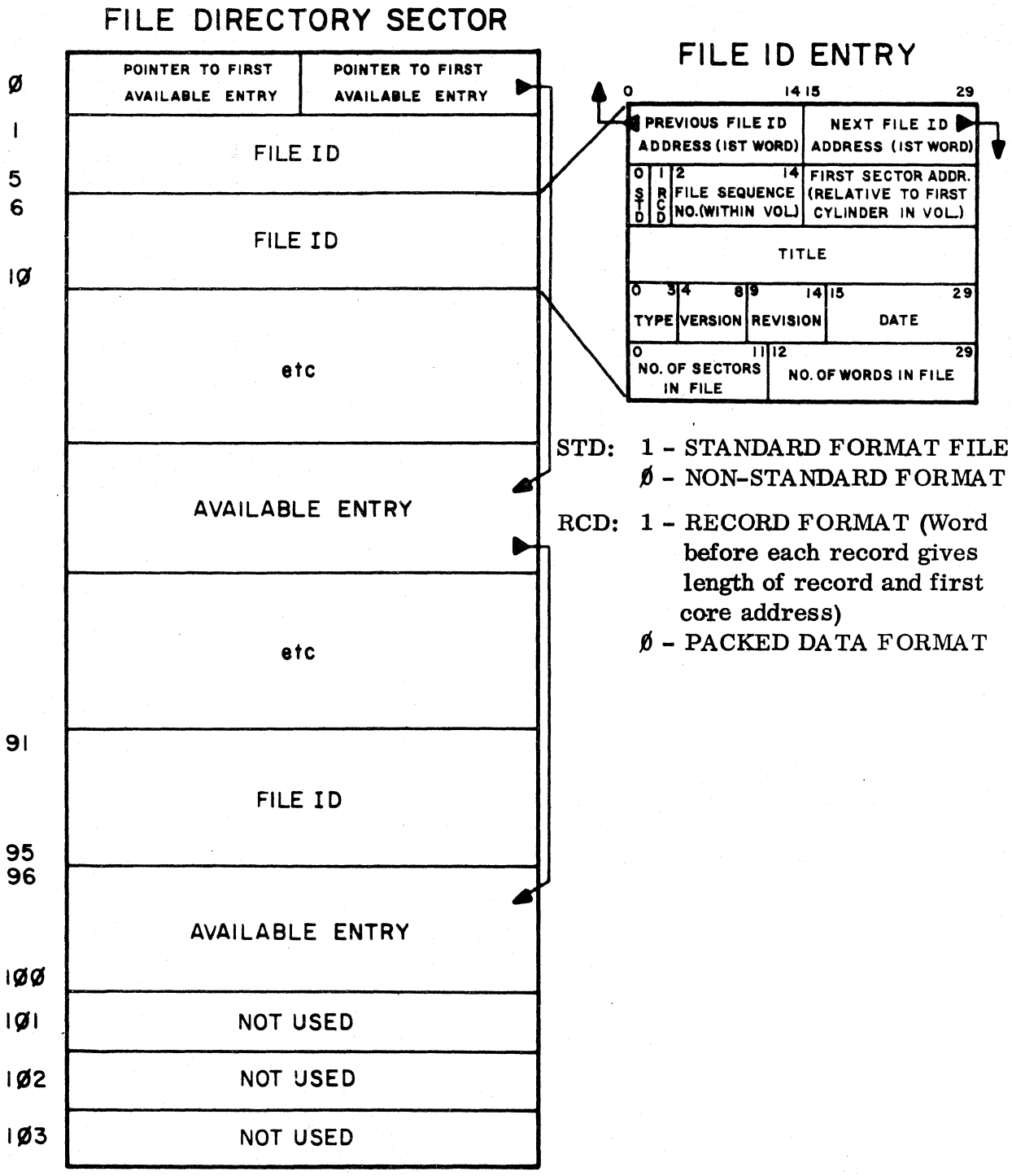
CYLINDER 0

SECTOR Ø	VOLUME DIRECTORY
SECTOR 1	FILE DIRECTORY SECTOR 1
SECTOR 2	FILE DIRECTORY SECTOR 2
SECTOR 3	FILE DIRECTORY SECTOR 3
SECTOR 4	FILE DIRECTORY SECTOR 4
SECTOR 5	FILE DIRECTORY SECTOR 5
SECTOR 6	FILE DIRECTORY SECTOR 6
SECTOR 7	FILE DIRECTORY SECTOR 7
SECTOR 8	FILE DIRECTORY SECTOR 8
SECTOR 9	FILE DIRECTORY SECTOR 9
SECTOR 10	FILE DIRECTORY SECTOR 10
SECTOR 11	FILE DIRECTORY SECTOR 11
SECTOR 12	FILE DIRECTORY SECTOR 12
SECTOR 13	FILE DIRECTORY SECTOR 13
SECTOR 14	FILE DIRECTORY SECTOR 14
SECTOR 15	FILE DIRECTORY SECTOR 15

CYLINDER 1

SECTOR Ø	FILE DIRECTORY SECTOR 16
SECTOR 1	FILE DIRECTORY SECTOR 17
SECTOR 2	FILE DIRECTORY SECTOR 18
SECTOR 3	FILE DIRECTORY SECTOR 19
SECTOR 4	FILE DIRECTORY SECTOR 20
SECTOR 5	FILE DIRECTORY SECTOR 21
SECTOR 6	FILE DIRECTORY SECTOR 22
SECTOR 7	FILE DIRECTORY SECTOR 23
SECTOR 8	FILE DIRECTORY SECTOR 24
SECTOR 9	FILE DIRECTORY SECTOR 25
SECTOR 10	FILE DIRECTORY SECTOR 26
SECTOR 11	FILE DIRECTORY SECTOR 27
SECTOR 12	FILE DIRECTORY SECTOR 28
SECTOR 13	FILE DIRECTORY SECTOR 29
SECTOR 14	FILE DIRECTORY SECTOR 30
SECTOR 15	FILE DIRECTORY SECTOR 31


VOLUME DIRECTORY
VOLUME ID ENTRY
0
14 15
29
ADDR. OF LAST FILE ID OF VOLUME
ADDR. OF FIRST FILE ID OF VOLUME
NO. OF CYLINDERS ASSIGNED
FIRST CYLINDER NUMBER
VOLUME DEFINITION TABLE
Ø
1
2
3
4
5
6
7
VOLUME 1 ID
VOLUME 2 ID
VOLUME 3 ID
VOLUME 4 ID
etc
62
63
VOLUME 32 ID
FILE DIRECTORY OCCUPANCY LIST
64
NOT USED
65
F D SECTOR 1 ENTRY
66
F D SECTOR 2 ENTRY
67
F D SECTOR 3 ENTRY
etc
95
F D SECTOR 31 ENTRY
FD SECTOR OCCUPANCY ENTRY
0
14 15
29
POINTER TO 1st AVAILABLE ENTRY BIN
COUNT OF NO. OF FILLED ENTRIES IN SECTOR
CYLINDER ALLOCATION
96
LAST AVAILABLE CYLINDER +1 (BEG. OF SCRATCH PAD)
97
FIRST AVAILABLE CYLINDER (END OF SYSTEM OR CURRENT)
98
NOT USED
99
CREATION DATE
100
0 to 19$_{10}$ CHARACTER PACK IDENTIFICATION STRING
103



STD: 1 - STANDARD FORMAT FILE
Ø - NON-STANDARD FORMAT

RCD: 1 - RECORD FORMAT (Word before each record gives length of record and first core address)
Ø - PACKED DATA FORMAT

INTRODUCTION

The FLSTR AMOS File Lister is a relocatable routine in the AMOS system library having two subroutines which can list files or records (depending on configuration) at the system teletypewriter or graphics display scope.

VERSIONS

FLST1 - For use with 8K systems
FLST2 - For use in systems with more than 8K and a graphics facility

SOFTWARE REQUIREMENTS

FLST1 - AMRM (appropriate Version for hardware configuration used)
FLST2 - AMRM (appropriate Version for hardware configuration used)
EDIT - DISP - FONT

DESCRIPTION

The following input statements to the AMOS Monitor

LISTF (TAPE, START FILE)!

or

LISTR (TAPE, START FILE)!

where

TAPE is the requested tape drive number: Ø, 1, 2, or 3
START FILE is the starting file
or RECORD!H FILE numbers.
(If RECORD is omitted, it is assumed to be 1. If START FILE is omitted, FLSTR lists the entire tape).

causes the specified TAPE to be rewound and scanned, starting at START FILE, for record header information which is appended to EDIT's text buffer for display, editing, listing, etc. (Version 2) or typed directly on the system console (Version 1).

Output is by logical files or by records, respectively, under the headings

FILE RECORDS NAME ORIGIN LENGTH TYPE VERS REV DA/MON/YR
or RECORD

FILE is the logical file number as contained in the first header word.

RECORDS (LISTF entry only) is the actual number of records on the file (not the last one's record number).

RECORD (LISTR entry only) is the record's actual record number.

NAME is the file's internal character-code name.

ORIGIN is the specified origin of the record or file.

LENGTH is:
(LISTF entry) the sum of all specified record lengths within the logical file.
(LISTR entry) the specified record length.

TYPE is the file or record's specified type.

VERS is the version number of the file.

REV is the revision level of the file.

DA/MON/YR is the date the file was created.

TERMINATION AND RE-ENTRY

FLSTR continues to scan the tape until the second file mark, which indicates the logical end of tape has been read, or until the operator has depressed IC [ø].

If LISTF or LISTR is re-entered while still in memory, FLSTR will append text to that already existing in the text buffer.

ERROR MESSAGES

If there is not enough unused core storage, FLSTR types "NO MEMORY AVAILABLE" and returns to the caller. If the requested START FILE is not found, FLSTR types "FILE NOT FOUND", rewinds the tape, and returns. If there is not enough storage to list all the files requested, FLSTR types "NOT ENOUGH MEMORY", and returns, retaining in the text buffer all information it has thus far accumulated.

INTRODUCTION

FNSIO is one of a set of AMOS system programs in the subroutine library which may be called by either programs or image descriptions to interface on-line terminal I/O devices with the calling program or image description. Using the function switches or pedals of the FNS1-P1/P2 subsystem, the programmer may enter selected values, execute routines and control the indicator lights of the function switch array. The FNS1-P2 has a light associated with each function key which is controlled by the program rather than by the associated switch. FNSIO samples the input from the FNS1-P1/P2 subsystem at the frame rate specified by the user's call to the CLOCK facility in the DSPLY operator. DSPLY then implements sampling of frames at the specified rate.

HARDWARE REQUIREMENTS

AGT/10, 30, or 50 having an FNS1-P1/P2 subsystem.

SOFTWARE REQUIREMENTS

FNSIO requires the CLOCK facility provided by the DSPLY operator.

STORAGE & TIMING

FNSIO occupies less than 450_8 words of storage.

Time/frame for no change = 80 μsec.
Time/frame for change = 600 μsec. plus time of any subroutines executed.
Time/call to FNSIO = 100 μsec.

VERSION

FNSIO exists in one version for the FNS1-P1 and FNS1-P2 subsystems.

INITIALIZATION

The FNSIO program defines the three entry points: FNSIO, FNSK, and FNSR.

The calling sequence:

```
JPSR    $CLOCK
        $FNSK
        $FNSR
```

causes the FNS1 Function Switches sampling program, FNSK, to be chained to the CLOCK subroutine calling list with a return instruction being put in FNSR. FNSK causes any requested operations, "OP", to be carried out upon the detection of a switch changing from a "reset" to "set" or "set" to "reset" status.

USE

The calling sequence:

```
JPSR    $FNSIO
        SW
        OP
        VALUE
```

causes the setting up of a table of procedures, specified by "OP", upon the sensing of a change in a selected function switch, specified by "SW". The operation, "OP", may take an argument, "VALUE". The available operations are the following:

OP = Ø ignore the specified switch.
1 set the location specified by "VALUE" to +1 if the specified switch is set.
2 execute the routine with entry at "VALUE" when the specified switch is set.
3 set the location specified by "VALUE" to -1 if the specified switch is set.
4 turn on light specified by "SW".
5 turn off light specified by "SW".

INTRODUCTION

FONT is an ADEPT program consisting of a character instruction execution table and display coordinate lists for all characters in the AMOS character set. The coordinate lists are made available to DISP, the display package for AMOS EDIT, and to other user routines. FONT consists of four versions with the configuration dependence outlined in the Hardware Requirement section.

HARDWARE REQUIREMENTS

Version 1 - OPC and OSD-1

Version 2 - TTY and OSD-1

Version 3 - OPC and OSD-2

Version 4 - TTY and OSD-2

THE CHARACTER SET

The FONT character set is dependent on console characteristics (OPC or TTY). The displayed set according to the octal AMOS internal character code is as follows:

Octal	OPC	TTY	Octal	OPC	TTY
ØØ	°	[	22	2	2
Ø1	%	%	23	3	3
Ø2	¢	]	24	4	4
Ø3	!	!	25	5	5
Ø4	&	&	26	6	6
Ø5	*	*	27	7	7
Ø6	:	:	3Ø	8	8
Ø7	_	\	31	9	9
1Ø	÷	+	32	;	;
*11	TAB	TAB	33	=	=
12	?	?	34	,	,
13	"	"	35	-	-
14	'	'	36	.	.
*15	C/R	C/R	37	/	/
16	(	(	*4Ø	SPACE	SPACE
17	)	)	41	A	A
2Ø	Ø	Ø	42	B	B
21	1	1	43	C	C

Octal	OPC	TTY	Octal	OPC	TTY
44	D	D	62	R	R
45	E	E	63	S	S
46	F	F	64	T	T
47	G	G	65	U	U
5Ø	H	H	66	V	V
51	I	I	67	W	W
52	J	J	7Ø	X	X
53	K	K	71	Y	Y
54	L	L	72	Z	Z
55	M	M	73	$	$
56	N	N	74	#	#
57	O	O	75	@	@
6Ø	P	P	76	±	↑
61	Q	Q	*77	BKSP	←

The characters marked with an asterisk (*) by their octal values (11, 15, 4Ø and 77) do not correspond to normal displayed characters. The entries for these special characters in the instruction table contain non-normal instructions (see section on Instruction Table).

In addition, the fetch instructions and character lists are available for three additional characters -- down arrow ("↓"), right arrow ("→"), and page mark ("¶").

CHARACTER LIST FORMAT

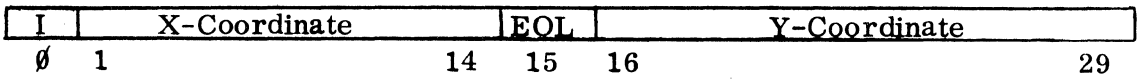
All characters (except right arrow and page mark) in an unscaled list are constructed with a maximum size of $24Ø_8$ in the x direction and $4ØØ_8$ in the y direction with the Ø, Ø reference origin in the lower left-hand corner of the character. As the coordinate is rotated one bit left before use, the resultant values are twice the maximum figures stated above.

The word format for the OSD-1 character lists is as follows:

B	X-Coordinate	EOL	Y-Coordinate
Ø	1 14	15	16 29

where EOL is the end-of-character flag present in the last vector of each character, and B is the blanking flag, indicating, if on, that the vector is not to be displayed. The first vector in each character is assumed to be blanked, and the blanking flag is not present in the first vector.

The word format for OSD-2 character list is:



where EOL again specifies the end-of-character. However, bit Ø, which indicates blanking in the OSD-1 list, indicates intensification of the vector for the OSD-2 list. Bit Ø, is never "1" on the first vector of each character.

INSTRUCTION TABLE

Contained in FONT is a 100_8 word instruction table, with entry name DCH15, in the order of the AMOS internal 6-bit code. The instructions for the four non-displayed characters (codes 11, 15, 40 and 77) are:

Code (and Table Position)	Instruction	
11	JUMP	$DCH10
15	JUMP	$DCH07
40	JUMP	$DCH14
77	JUMP'I'X	$DCHAR

These "JUMP" instructions provide special character re-entry into DCHAR, the DISP display character subroutine which uses FONT.

The address field of the other instructions references the first vector of its appropriate character. The instruction portion is dependent on the OSD version. With OSD-1, the instruction portion is "MDAR" and with OSD-2, "MD10'B."

The three additional special characters, down arrow, right arrow, and page mark, each have one instruction word containing an "MDAR" ("MD10'B" for OSD-2) addressed to the first vector of the character. The entries for these special characters are:

Character	Entry
Down Arrow	DC26X
Right Arrow	DC21X
Page Mark	DC40X

In addition to these entries and to DCH15 noted above, the first vector of the entire coordinate list and the last vector of the list are entries defined DC241 and DCHXX respectively for use by the coordinate list scaling routine in DISP.

GENERAL

The FREEZ Graphics Operator program transforms a display image into its equivalent sets of X, Y, and Z coordinates. This new coordinate set is a different image that will produce the instantaneous picture that was on the CRT screen; thus, the FREEZ operator transforms an image, made up of many different types of image commands, into a set of coordinates which is equivalent to the picture that the complex image had created. This transformation causes the loss of the identities of specific subimages within the image.

The resulting transform is stored in a specified core buffer. The transformed images generally occupy more core storage than the original, but since the image description of the transform consists entirely of packed MOVE and DRAW items, 2DTBL items, or LABL items, it requires minimal software interpretation for CRT display. The effect is a very efficient operation of the AVG1 and the LCG1 subsystems.

HARDWARE REQUIREMENTS

AGT/30 or AGT/50

SOFTWARE REQUIREMENTS

AMOS - Monitor
DSPLY - Operator

OPERATIONS AND SEQUENCES

This section contains the monitor control statements for the various FREEZ operations:

FREEZ("name", img, buflw, bufmx, bufov, adcer)!

or an equivalent ADEPT calling sequence will freeze the image beginning at "img", and store the transformed image into locations "buflw" - through "bufmx".
A standard two word text header will be built with the text "name". A standard RET statement will be inserted at the end of the image definition. All LABL image items will be changed to immediate addressing LABL items, and the character strings

addressed by the LABL item will be moved into the specified buffer starting at the location following the RET item (i.e., image terminator). FREEZ returns with the address of the last used buffer cell in the AR.

FREZ1 (img, buflw, bufmx, bufov, adcer)!

or its equivalent ADEPT calling sequence will have the same result as FREEZ, with the exception that it will not build the two word text header or the RET statement at the end. Also, the character strings addressed by LABL items will not be moved into the buffer region.

FREZ2 ("name", img, buflw, bufmx, bufov, adcer)!

or its equivalent ADEPT calling sequence will freeze the 3-dimensional image into a 2-dimensional projection. All vectors, except those characters drawn by LABL image items, will be translated into a 2DTBL image item that describes the entire image displayed. This form of the image can be displayed in the 2-dimensional normal mode of the AVG1 which operates nearly twice as fast as the 3-dimensional normal mode. FREEZ and FREZ1 stabilizes the image into a full 3-dimensional image of very efficient construction, while FREZ2 does the same with the image for its 2-dimensional replica, allowing even more efficient operation of the AVG1 subsystem.

ERROR ROUTINES

There are two error routine entries for FREEZ. They are:

1. "bufov", if the buffer is too small
2. "adcer", if the picture scale is too large

The user may specify any routines for these error conditions. The following routines are automatically and initially specified if omitted.

STOPF - on a buffer overflow this routine will halt the conversion process and close off freezing of the image at the point where the buffer overflowed. In addition, STOPF outputs on the teletype:

BUFFER TOO SMALL

RESCL - if the picture scale is too large to allow undistorted conversions, this routine will scale the picture by 1/2 and try the conversion a second time. RESCL will type:

ADC OVERFLOW - WILL RESCALE IMAGE AND TRY AGAIN

If the picture scale is too large, RESCL will terminate operation and type:

PICTURE SCALE STILL TOO LARGE - ABORT

INTRODUCTION

FTAP is a set of magnetic tape routines used by AFORT to implement its input-output commands. It makes use of LIBIO and MTAC as well as the AMOS monitor to implement its routines. FTAP is loaded by OBJPK and all of its routines, with the exception of STIO are called directly by OBJPK.

VERSIONS

FTAP may be assembled in any of five versions, specified at assemble-time by the number of tape units on the AGT system.

No. of Tape Units	Version	Core Requirements
None	1	5 words
1	2	320_8 words
2	3	335_8 words
3	4	352_8 words
4	5	365_8 words

LOADING

FTAP is a relocatable program which is loaded by OBJPK whenever OBJPK is loaded.

UNIT ERROR

If an AFORT program specifies a tape unit that does not exist for the version of FTAP in core, a UNIT ERROR will occur.

AFORT MAGNETIC TAPE ROUTINES

A. Read or Write a Non-AMOS Format Record

The calling sequence:

```
        JPSR    9MTn    [ Entry where n is the tape unit #
                .       [ Normal return
```

causes a record on tape unit n to be read ($9IO = Ø) or written ($9IO ≠ Ø). All input/output uses buffer at $9OB of length 34_8. In normal use $9IO and $9OB exist in OBJPK.

B. Read or Write an AMOS Format Record

The calling sequence:

```
JPSR    9MTnA    [ Entry where n is the tape unit #
          .      [ Normal return
```

causes a record on tape unit n to be read ($9IO = Ø) or written ($9IO ≠ Ø). All input/output uses buffer at $9OB of length 34_8. In normal use $9IO and $9OB exist in OBJPK.

C. Backspace a Record

The calling sequence:

```
JPSR    9BA      [ Entry
          .      [ Normal return
```

causes the tape unit to be set by the OBJPK routine $9SU and then backs up one record.

D. Write a File Mark

The calling sequence:

```
JPSR    9EN      [ Entry
          .      [ Normal return
```

causes the tape unit to be set by the OBJPK routine $9SU and then backs up one record.

E. Rewind Tape Unit

The calling sequence:

```
JPSR    9RE      [ Entry
          .      [ Normal return
```

causes the tape unit to be set by the OBJPK routine $9SU and then rewinds the tape.

F. Special Tape Input/Output

The calling sequence:

```
JPSR    STIO    [ Entry
        itype   [ Address of ITYPE
        idens   [ Address of IDENS
        itab    [ First address of ITAB
```

or the following AFORT statement will cause the results described:

CALL STIO (ITYPE, IDENS, ITAB)

Purpose: Change the external type, density and character set to allow reading and writing magnetic tapes which are compatible with other machines.

Definitions: ITYPE determines the tape parity

- Ø - no change
- 1 - BCD or even parity
- 2 - BIN or odd parity

IDENS determines the tape density in bits per inch

- Ø - no change
- 1 - 200 bpi
- 2 - 556 bpi
- 3 - 800 bpi

ITAB refers to the name of the lookup table

- 1ØØ - no table - use internal character set
- N - name of dimensioned variable (64)

Restrictions: The lookup table is a list of 64 right justified, externally coded characters written one character per word. The table must be written in the numerical sequence, 00 to 77_8, of the AFORT internal character code (see Table).

If a code appears twice on the table it will be read according to the earlier occurrence on the list.

An undefined character code will be read as 44_8 which is undefined to the fortran object package.

INTERNAL CODES

TTY Char.	AFORT Code	TTY Char.	AFORT Code
SPACE	00	W	40
0	01	X	41
1	02	Y	42
2	03	Z	43
3	04	↑	44
4	05	@	45
5	06	%	46
6	07	]	47
7	10	I	50
8	11	J	51
9	12	K	52
...	13	L	53
...	14	M	54
...	15	N	55
...	16	...	56
...	17	...	57
A	20	+	60
B	21	-	61
C	22	*	62
D	23	/	63
E	24	.	64
F	25	(	65
G	26	)	66
H	27	,	67
O	30	=	70
P	31	&	71
Q	32	:	72
R	33	...	73
S	34	$	74
T	35	#	75
U	36	...	76
V	37	C/R	77

NOTE

"..." indicates that this internal code is unused in the AFORT system.

adage

LIBIO

MAGNETIC TAPE FILE INPUT/OUTPUT ROUTINES

Programmer's Reference Manual

February 1969

CONTENTS

*NOTE: The coding covered by the description of these routines is not contained in LIBIO but in either AMRMX or AMRMX support programs. The descriptions are given for information purposes only and may be duplicated elsewhere.

INTRODUCTION

This document describes the standard AMOS Magnetic Tape formats and the tape driving routines contained in LIBIO. These routines also reside in the resident portion of the AMRMX (AMOS Resident Monitor) Versions 1, 2, 3, 4, 5, 6, 11 and 12. LIBIO is used externally in AMRMX Versions 7, 8 and 9 for driving the magnetic tape subsystems. LIBIO consists of two versions with the following hardware configuration requirements:

Version 1 -- MTP5-P1 or MTP8-P1
Version 2 -- MTP7-P1, DDC1-P1

MAGNETIC TAPE FORMAT

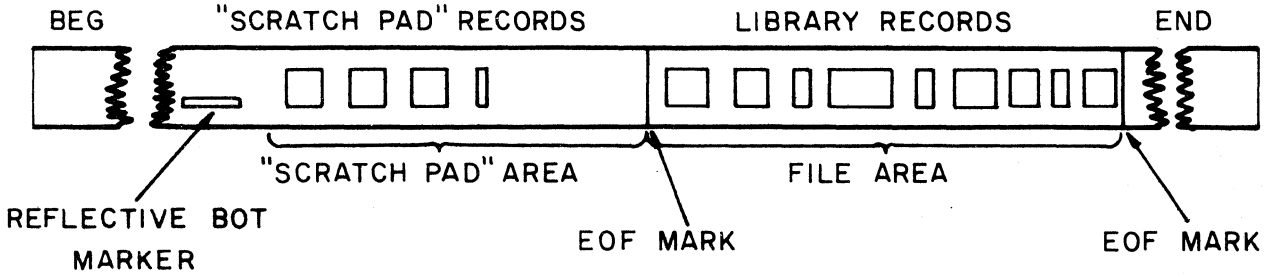


Figure 1. Standard AMOS Magnetic Tape Format

The first portion of the tape, between the reflective BOT (beginning of tape) marker and the first EOF (end of file) mark, is called the "scratch pad." This area contains text records operated on by the system text editor. The length of this area is specified when a tape is initialized. The area between the first and second EOF marks is called the file area and is the normal library file storage area for the tape. All valid (current) files on the tape are contained within the "scratch pad" area or the file area.

The "scratch pad" may be empty or may contain one ATEXT file. If a file is present, it is given the file number 10_8. Files in the file area are sequentially numbered starting with file 11_8. Each file on the tape has one or more records; each record is numbered sequentially within the file starting with record 1.

Figure 2 gives the format of individual records on standard AMOS tapes. Each word in the figure contains 30-bits (five 6-bit characters).

The record number and file number give the sequential identifiers for file and record within file.

The file TITLE consists of from one to five alphanumeric or special characters (left justified and filled with 00 characters if less than five) and is used in loading and file identification.

The number of words field gives the number of 30-bit words in the body of the record, excluding header and trailer.

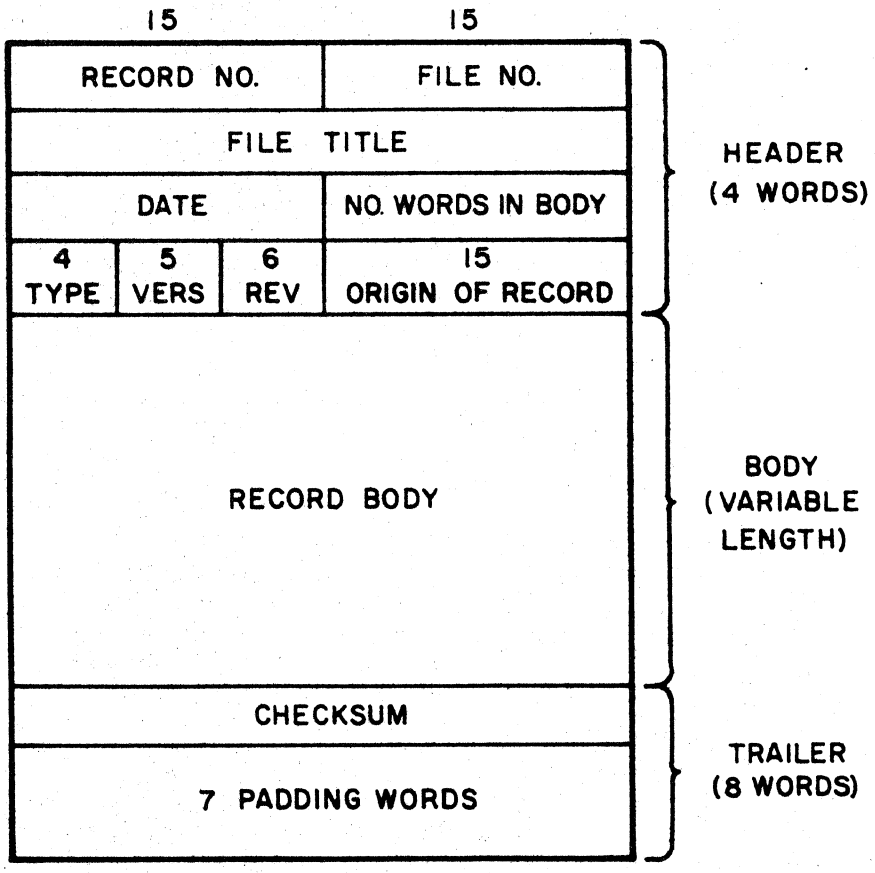


Figure 2. AMOS Tape Record Format

The date field gives the date the file was created in the following format:

Bit 0:	Unused
Bits 1-5:	Year - 1964. (i.e., 0 = 1964, 1 = 1965, 2 = 1966, etc.)
Bits 6-9:	Month (1 through 12.)
Bits 10-14:	Day (1 through 31.)

The origin field gives the core origin of the record at the time it was created. For binary files, this location will be where the record is read into memory.

The revision field gives one 6-bit alphanumeric or special character which indicates the revision level of the file.

The version field gives a version number (1 through 31_{10}, or none) for the file.

The type field gives the type of data contained in the file described below:

$Type_8$	Mnemonic	Description
00	SYMS	Absolute symbol dump
01	DATA	FORTRAN files
02	RELOC	Object programs
03	TRACF	Reactive typewriter forms
04	TEXT	Case-shift sensitive text
05	RLSYM	Relocatable symbols
06	PRNTR	Off-line printer line image records
07	ATEXT	Standard text
10	BIN	Absolute file
11	TYPE2	Unassigned -- reserved for system expansion
12	TYPE3	
13	TYPE4	
14	TYPE5	Unassigned -- available for user assignment
15	TYPE6	
16	TYPE7	
17	TYPE8	

The checksum word contains the arithmetic sum (ignoring overflows) of all words in the body of the record. When computed, the initial value of the checksum word is set to -0 before accumulation.

The padding words (which may contain any value) are present to provide a minimum number of characters in each tape record. They are ignored by the software.

SELECTION AND CONTROL ROUTINES

The subroutines described in this section control the setting of tape density, selection of tape unit for subsequent tape operations, and specification of version and revision parameters for magnetic tape output.

$DSET -- Density Selection

This subroutine sets the tape density according to a given input argument. Programmatic density selection is valid only in Version 2 of LIBIO; however, the entry point $DSET is included in Version 1 for cross-version compatability. When using Version 1 of LIBIO, the density setting is made by setting a switch on the tape control unit to 200 or 556 bpi.

All tape operation subroutines in LIBIO assume binary (ODD) parity in use and incorrect detection of EOF marks may result if BCD (EVEN) parity is used.

```
JPSR    $DSET    [Call
         code    [0=200BPI, 1=556BPI, 2=800BPI
          .      [Returns here
```

$STAPE -- Tape Selection

This subroutine selects the tape unit for subsequent tape operation and sets up a pointer to the file mark counter for the selected tape unit (see 9IXFC).

```
[   (AR) = n              n=unit number (0, 1, 2, or 3)
    JPSR     $STAPE       [Call
              .           [Returns here
```

$STDN -- Tape Selection in IC Register

This subroutine first clears out any tape control bits in the IC register, then ORs the currently selected tape unit into IC[19-20].

```
JPSR    $STDN    [Call
        .        [Returns here
```

$VERSN -- Version and Revision Parameter Setting

This subroutine sets the version and revision parameters needed by tape output operations.

```
JPSR    $VERSN   [Call
        Vers     [Version no. (-0 if no version desired)
"Rev"            [Revision character (bits 0 - 5)
        .        [Returns here
```

After the subroutine has been executed, location $TVERS will contain the version number in bits 19 - 23 and the revision character in bits 24 - 29. This subroutine is not contained in LIBIO but is included here for information.

POSITIONING ROUTINES

The subroutines described in this section control the positioning of the selected tape unit. It is assumed that the proper unit selection and density setting is made by calls to $STAPE and $DSET respectively before any of the following routines are called.

$BEAR -- Take Bearings

This subroutine is called to determine the current position of the selected tape unit. BEAR backspaces one record and if not at BOT it reads a single record header and ignores the remainder of the record. BEAR sets the file mark counter for the selected tape unit to -0 if the tape is positioned before the first EOF mark and to +0 if positioned after it. BEAR returns to the calling program with -0 in the AR if a normal record header was read or with +0 in the AR register if the tape is at BOT or if an EOF mark was read. In the former case, the record header is available for use by the calling program (core location of header data may be found in the INPUT/OUTPUT ROUTINES Section).

```
        JPSR    $BEAR   [Call
                .       [Returns here
[       (AR)=+0         BOT or EOF seen
[       or=-0           Record header read
```

$RWI, $IRWI, $WBOT, $HWBOT -- Rewind Tape

These subroutines cause the selected tape unit to be rewound to the BOT marker. In Version 1 of LIBIO, the four above entries are the same subroutine. BEAR is first called to determine the position of the tape. If the tape is beyond the first EOF mark, a high-speed rewind operation will be initiated; otherwise, a low-speed rewind is started. If IC[0] is depressed during the rewind, the rewind operation will be terminated, BEAR called again to set the current position of the tape, and control returned; otherwise, control is returned to the calling program when the tape has reached the BOT marker.

In Version 2 of LIBIO, two rewind subroutines are present. A call to $RWI or $IRWI will initiate a high-speed rewind operation and return control. A call to $WBOT or $HWBOT will initiate a high-speed rewind, wait until the tape has reached the BOT marker, and then return control. Once a rewind operation has been started, the tape cannot be stopped until it reaches the BOT marker, and any subsequent operations which cause tape movement (except for $BSX, $BEAR, or any rewind call) are illegal until the tape has reached the BOT marker.

```
        JPSR    $RWI    [Call (or $IRWI, $BOT, or $HWBOT)
                .       [Returns here
```

$BSX -- Backspace Tape One Record

This subroutine causes the currently selected tape unit to be backspaced one record. BSX may be called prior to reading a record but not immediately before a write operation as the erase head of the tape unit would not be stopped within the record gap. Note that EOF marks are counted as one record when backspacing.

```
        JPSR    $BSX    [Call
                .       [Returns here
```

$BRF -- Backspace Record and Position to Write

This subroutine is called to backspace and position the tape heads for a write operation. BRF backspaces over two records then reads the record header of the second record backed over and ignores the rest of the record. The tape is then positioned to write a record. BRF also sets location $LID to be the ID (record and file numbers) of the second record backed over, or to be 10_8 if an EOF mark was read in place of a record header. This subroutine is normally called to position for appending to the file area of a tape after reading the second EOF mark, or for repositioning to rewrite a bad record.

```
        JPSR        $BRF    [Call
        .                   [Returns here if EOF mark seen
        .                   [Header return
```

$SFM -- Position to End of File Area

This subroutine is called to position the currently selected tape unit for appending files or records to the file area of a tape. SFM scans the tape until the second EOF mark is seen, then calls $BRF to back over the EOF mark and position for writing. SFM returns with the contents of location $LID containing either the ID of the last record on the tape, or 10_8 if the file area is empty.

```
        JPSR        $SFM    [Call
        .                   [Returns here if file area is empty
        .                   [Returns here if file area not empty
```

$9IXFC -- Index File Mark (EOF) Counter

This subroutine is called to index the EOF mark counter for the currently selected tape unit. This subroutine should be called when an EOF mark is read while scanning down a tape. Subroutine 9IXFC returns with the indexed value of the counter so that the calling program can determine the tape position.

```
        JPSR        $9IXFC  [Call
                    .       [Returns here
[       (AR) = 0            EOF mark read was first one on tape
[       or ≠ 0              EOF mark read was second one on tape
```

INPUT/OUTPUT ROUTINES

The subroutines described in this section consist of input/output routines for reading and writing tape records and for writing EOF marks. It is assumed that the tape has been positioned, and that tape unit selection and density setting are made by calls to $STAPE and $DSET respectively before calls are given to subroutines in this section.

$RHD -- Read Record Header

This subroutine reads the record header information from the next record on to the currently selected tape unit, and returns to the calling program while the tape is still in motion for a decision to either read into memory the body of the record or to ignore it. While the record header is being read, the header words are stored in core memory and certain tests are made on the header information. These tests will be used by the calling program when making the read/ignore decision. The following is a list of functions performed while RHD is in the process of reading a record header:

$WORD_1$	→	TEM
$WORD_1$ ' TFID	→	IDFF
FOLD ($WORD_1$ ' TFID)	→	IDF
$WORD_2$	→	STB
FOLD ($WORD_2$ ' TNAME)	→	NAMF
$WORD_3$	→	RDATE
$WORD_3$ & 77777_8	→	NOW
if NOW > TLENG, -0	→	FITF
else +0	→	FITF
TLENG - NOW	→	NOWF
$WORD_4$	→	FST
($WORD_4$! H & 74000_8) ' TTYPE	→	FSTF, AR Register

where:

FOLD (X) = X ! H↑ X

X ' Y = X exclusive ORed with Y

X & Y = X ANDed with Y

X↑ Y = X inclusive ORed with Y

X - Y = X minus Y

X ! H = X rotated left 15-bits

X→ Y = X replaces the value of Y

$WORD_n$ = Word from tape header (n = 1, 2, 3, or 4)

Starting with the innermost, all operations are performed in sequence from left to right within each parenthetical nesting.

As RHD prepares for the possible operation of reading in the tape record, the EAU1 overflow pivot location is cleared, and the checksum accumulator word initialized to -0. The calling sequence to RHD is as follows:

```
JPSR    $RHD    [Call
.               [Returns here if EOF
.               [Returns here after header
```

If an EOF mark is detected by RHD, return will be made to the first location after the call with the tape stopped. If not, return is made to the second location after the call with the tape in motion is in the process of reading the fifth word of the tape record. The calling program must make a decision based upon prior knowledge or on information imparted by the parameters stored by RHD, and then call either $IRC to ignore the reading of the record, or $RRC to cause the record to be read into core memory.

In Version 2 of LIBIO, the calling program may execute, at most, four instructions before calling $RRC or $IRC while reading tape at 800 bpi. At lower tape densities, a proportionally greater number of instructions may be executed. In Version 1, $RRC or $IRC must be called within 50 to 60 μs after returning from RHD. Note that 800 bpi density cannot be used for Version 1 LIBIO routines.

$IRC -- Ignore Tape Record

IRC is called, following a return from $RHD or $FIND, to cause the record currently being read to be skipped and the tape unit to be stopped in the gap following the record. IRC returns control to the calling program after the record has been scanned and the tape unit stopped.

```
JPSR    $IRC    [Call
        .       [Returns here
```

$RRC -- Read Tape Record

RRC is called, following a return from $RHD or $FIND, to cause the record being scanned to be read into core memory. The address of the core buffer area is transmitted to RRC in the AR register. The number of words loaded into core memory is specified by the number of words filed in the record header. Once the record has been read the tape parity error indication is checked, and the record backspaced and re-read if incorrect. The backspacing and re-reading will continue until the record is read with the correct parity, or until IC[0] (or FNS 1) is depressed by the operator in which case the record read into memory will be accepted. Control will then be returned to the calling program. While the record is being read into core memory, RRC computes a checksum of all body words and compares it with the checksum word written on the record. An incorrect checksum will also cause a backspace and re-read operation until the record is read with the correct checksum or until IC[1] (or FNS 2) is depressed. Control is returned with the tape stopped in the record gap after the record just read.

```
[       (AR)    Read buffer origin
        JPSR    $RRC    [Call
                .       [Return here after record read
```

$FIND -- Find Record

This subroutine is called to find a particular record on the currently selected tape. The calling program must first initialize location $TFID to contain the file number of the record desired in bits 15 - 29 and the record number within the file in bits 0 - 14. FIND will determine its position on the tape and read forward to the record if the tape is positioned before the record desired,

rewind and then read forward if the tape is positioned after the file which contains the record desired. FIND will backspace the proper number of records and read forward if the tape is positioned within the file but beyond the desired record. In any case, FIND will return to the second location after the call, after finding the desired record with the tape in motion. The calling program must either call $IRC to ignore the record or call $RRC to read the record into core memory. If 800 bpi density is being used (Version 2 of LIBIO) the calling program must immediately call $RRC or $IRC (one instruction for fetching the record buffer address is allowed when calling $RRC). If the record desired is not found on the currently selected tape unit, control is returned to the first location after the call, with the tape stopped and positioned just prior to the second EOF mark.

```
[     (TFID) = Record ! H File
      JPSR      $FIND      [Call
      .                    [Returns here if not found
      .                    [Returns here after correct
                            header scanned
```

$BOUND -- Set Output Buffer

This subroutine is called to set the limits of the output record area.

```
      JPSR      $BOUND     [Call
                First      [Address of first word
                Last       [Address of last word
                .          [Returns here
```

```
[     ($SRCHF) = First
[     ($SRCHL) = Last
[     ($BNDF)  = $SRCHF
[     ($BNDL)  = $SRCHL
```

This subroutine is not included in LIBIO, but is included for information.

$WREC -- Write Record

This subroutine is called to write a record on the currently selected tape

unit at its present position. The area of core memory to be written as the body of the record is specified by the last $BOUND call. The checksum of the area of core memory is computed and used for the record being written. The header words are created by the following procedure:

if WFF < 0 or LID[0-14] = 0, (LID + 1) [15-29]	→	LID
LID + (1 ! H)	→	LID
LID	→	$WORD_1$
TNAME	→	$WORD_2$
TDATE [15-29]	→	$WORD_3$ [0-14]
($SRCHL) - ($SRCHF)	→	$WORD_3$ [15-29], TLENG
TVERS [19-29]	→	$WORD_4$ [4-14]
TORIG [15-29]	→	$WORD_4$ [15-29]
TTYPE [15-18]	→	$WORD_4$ [0-3]

Once the header information has been processed, WREC writes the record containing the header, body from (SRCHF) through (SRCHL), and trailer onto the tape. If a parity error condition exists after writing, WREC will back over the incorrect record, erase a section of tape, and rewrite the record. This operation continues until the record has been written properly, or until the operator depresses IC[0] (or FNS 1) which causes the parity error condition to be ignored. Control is returned to the calling program after the record has been written, with the tape stopped in the gap following the record just written. Before writing the record, WREC checks for write-lockout on the tape unit being used. If a write-lockout condition exists a message to that effect will be typed, and the operator must enable the writing of the unit and depress IC[0] (or FNS 1). WREC will then again attempt the tape write operation. In the process of removing the write-lockout condition, if the operator moves the position of the tape in the tape unit, the record will not be written at the proper position on the tape.

```
JPSR    $WREC    [Call
         .       [Returns here
```

$WFM -- Write File Mark (EOF)

This subroutine writes an EOF mark at its current position on the currently selected tape. After writing the EOF mark, WFM backs over it for proper subsequent operation of BEAR and subroutines which call BEAR. WFM does not check for WRITE ENABLE before or after initiating a file mark write operation.

```
JPSR    $WFM    [Call
        .       [Returns here after writing
                 EOF mark
```

INTRODUCTION

MTAC is a set of magnetic tape routines to supplement those existing in the AMOS Monitor. The combination of the two is designed to provide all tape dependent functions to every program used under the system.

VERSIONS

Version MTAC1 is used with the MTP5 and MTAC2, with the DDC1 mag tape subsystem.

LOADING

MTAC is a relocatable program which is loaded into memory automatically by the Monitor when one of its entry points is requested by another program being loaded. It may also be loaded by typing:

START ("MTAC1", UNIT)!

where unit is the tape unit number (Ø, 1, etc.).

SUPPLEMENTAL MAGNETIC TAPE ROUTINES

A. Magnetic Tape READ

The calling sequence:

```
        JPSR    MTARC
                N       ° Number of words to be read
                ADDR    ° Starting address
                UNIT    ° Unit number (Ø, 1, etc.)
RET1:           .       ° File mark return
RET2:           .       ° Normal return
```

causes (at most) N words of the next record on the specified UNIT to be read into memory starting at location ADDR. If a parity error is read, the program backs over the record and tries to read again, until the operator depresses IC [Ø]. In this case, MTARC exits via RET2 with the tape unit positioned after the offending record.

B. Magnetic Tape WRITE

The calling sequence:

```
        JPSR    MTAWC
                N       ° Number of words to be written
                ADDR    ° Starting address
                UNIT    ° Unit number (Ø, 1, etc)
RET1:           .       ° Write lockout return
RET2:           .       ° Normal return
```

causes a record N words in length to be written on the specified UNIT (Ø, 1, etc.) starting at location ADDR. If a parity error is written, MTAWC repositions the unit, erases a patch of tape, and tries to write again. If the operator depresses IC [Ø], the program will exit via RET2, positioned to write the same record. If write lockout is sensed, the program will position to write again and exit via RET1.

NOTE: MTAWC assumes that the tape is already positioned to write. MTAWC does not write records in TOPS2 format; it writes only the block of N words.

C. Erase SCRATCH Pad

The calling sequence:

```
        JPSR    ERSCP
        .       UNIT    ° Unit number (Ø, 1, etc.)
RET1:           .       ° Write lockout return
RET2:           .       ° Normal return
```

causes the SCRATCH pad on the specified UNIT (Ø, 1, etc.) to be erased in this way: ERSCP first rewinds the unit, then searches forward to the first file mark, backs over it, and erases to BOT.

D. Erase at BOT

The calling sequence:

```
        JPSR    ERBOT
                UNIT    ° Unit number (Ø, 1, etc.)
                .       ° Returns to next location
```

causes the tape to be rewound and prepared to write a record a few inches after the BOT. On the MTP5 system, this involves erasing about 3½ inches of tape. (The erase is automatic at the start of a write operation on the DDC system).

E. Erase TAPE

The calling sequence:

```
        JPSR    ERAS
                UNIT    ° Unit number (Ø, 1, etc.)
                N       ° Number of feet
RET1:           .       ° Write lockout return
RET2:           .       ° Normal return
```

causes N feet of tape to be erased on the specified UNIT, and then backs up the tape unit enough so that it is positioned to write on clean tape.

F. Write a FILE Mark

The calling sequence:

```
        JPSR    WRFM
                UNIT    ° Unit number (Ø, 1, etc.)
                .       ° Returns to next location
```

causes a FILE mark to be written on the specified unit.

G. Skip a RECORD

The calling sequence:

```
        JPSR    RSKIP
                UNIT    ° Unit number (Ø, 1, etc.)
RET1:           .       ° File mark return
RET2:           .       ° Normal return
```

causes the specified UNIT to skip over one record, which must be more than 2Ø characters in length, unless it is a file mark.

GENERAL

AMOS Tape to Printer Routines (MTPRT) list text files, file and record header listings, and octal record dumps for the LPR1 Line Printer.

CALLING SEQUENCES AND CODES

Calling Sequence

MTPRT (TAPE, FIRST, LAST, CODE, SYMS)!
where:

1. TAPE is the selected tape number (Ø, 1, 2, or 3).
2. FIRST is the first file selected.
3. LAST is the last file selected (or = FIRST if omitted).
4. CODE is the code for selecting the type of listing desired.
5. SYMS is the symbol file number of the type "SYMS" for the text listing (or Ø if symbol listing is not desired).

Definition of Codes

1. In code types Ø, 1, 2, no symbol files will be printed if SYMS is specified as equal to zero. If SYMS is non-zero, and multiple files are listed, then file SYMS will be printed as symbols for the first text file, SYMS + 1 for the second, etc.

2. In code types 3, 4, and 5, FIRST and LAST can be used to specify particular records in the selected file by giving one or both the following format:

RECORD! H FILE

where RECORD is the record in file FILE to be used as the FIRST or LAST argument.

Listing Format

The following listing formats can be generated by the codes:

Ø - Text listing from FIRST through LAST with symbol files SYMS, SYMS + 1, etc., and no tag listing

1 - Text listing from FIRST through LAST with symbol files SYMS, SYMS + 1, etc., and tag listing in ADEPT format

2 - Text listing from FIRST through LAST with symbol files SYMS, SYMS + 1, etc., and tag listing in ASMT3 format

3 - File header listing from file FIRST through LAST. If FIRST is zero, listing starts at first record on tape. If LAST is also zero, listing continues through end of tape.

4 - Record header listing from file FIRST through LAST. If FIRST is zero, listing starts at first record on tape. If LAST is also zero, listing continues through end of tape.

5 - Octal record dump with header information from file FIRST through LAST. If FIRST is zero, listing starts at first record on tape. If LAST is also zero, listing continues through end of tape.

NOTE

For operator convenience, several of the above calling sequences may be used on one MTPRT call in the form:

MTPRT ((T1, F1, L1, C1, S1), (T2, • F2, L2, C2, S2), etc., (Tn, Fn, Ln, Cn, Sn))!

Many different operations can be specified, and MTPRT will process each sub-argument list in sequence.

adage

OBJPK

AFORT OBJECT PACKAGE

Programmer's Reference Manual

Revision B

March 1969

TABLE OF CONTENTS

INPUT/OUTPUT OPERATIONS

GENERAL

Data is input from an external medium to internal memory or output from internal memory to an external medium by the Input/Output Routine. This routine converts data between the 6-bit external codes and internal FORTRAN 6-bit character codes. Data is processed in either the formatted or unformatted mode.

Formatted data is converted to real, integer, or Hollerith data as specified in a Format Statement.

Unformatted data is processed as 6-bit internal characters, packed five characters per computer word.

A program specifies an I/O operation to the Input/Output Routine by means of an I/O Initiate Call. The Argument List Call is required only when a list is associated with the input/output function. In general, an I/O operation is specified to the input/output routine in the following sequence:

I/O Initiate Call
Argument List Call
Argument List Call
.
.
.
Argument List Call
I/O Terminate Call

The I/O Initiate Call defines the operation type; each Argument List Call supplies the program data applicable to the operation; the I/O Terminate Call terminates the operation.

Data is operated upon in the form of external records. The I/O Initiate Call establishes a new record. Other records may be established according to the Argument List Calls or the specifications in the Format Statement. The I/O Terminate Call terminates any incomplete records.

External Device	Record Terminator
Magnetic Tape	Tape Gap
Paper Tape	Carriage Return Character
Typewriter	Carriage Return Character

Maximum Record Size

120 Six-Bit Characters

A record may be smaller than the maximum size if it has a proper record terminator. An error condition is created if a record is larger than the specified 120 character maximum size.

INPUT/OUTPUT INITIATION

Input/output operations are initiated by a coding sequence composed of three elements:

Unit Specifications
I/O Initiate Call
Data Mode Specification

Unit Specification

The device and unit number to be selected are declared by the Unit Specification. The number is placed in the A-register. Local unit numbers, and the equivalent devices specified, are shown in the following table:

Logical Unit Number	Device
1	Magnetic Tape 0
2	Magnetic Tape 1
3	Magnetic Tape 2
4	Magnetic Tape 3
21, 22, 23, 24	Logical Disk Volume
50	Console Typewriter
51	Paper Tape
54	ASCII Core Buffer

I/O Initiate Call

An I/O Initiate Call specifies an I/O operation in one of the following forms:

```
JPSR 9RD        [Input Data
JPSR 9WR        [Output Data
```

The input data and output data operations normally require an Argument List. If none is given, data is passed only as specified by a Hollerith type format specification. If this type of format is not specified, the result is a blank record for output or a record skipped on input.

The three auxiliary I/O operations given below do not use the Data Mode, Argument List Call, or I/O Terminate Call:

```
JPSR  9RE      [Rewind Magnetic Tape
JPSR  9BA      [Backspace Magnetic Tape One Record
JPSR  9EN      [Write File Mark on Magnetic Tape
```

Data Mode Specification

The data mode is supplied in the form of a parameter following either a JPSR 9RD or a JPSR 9WR. If this parameter is zero, the data is unformatted; otherwise, this parameter specifies the address of a Format Specification.

Application of the I/O Initiate Call is demonstrated in the following examples. An input operation is specified by the following sequence of instructions:

```
MDAR'F      51      [ Select Paper Tape
JPSR        9RD     [ Call Read Routine
```

An output operation is specified by the following sequence of operations:

```
MDAR'F      50      [ Select Typewriter
JPSR        9WR     [ Call Write Routine
0           0       [ Unformatted Output
```

A rewind magnetic tape operation is specified by the following instruction set:

```
MDAR'F      1       [ Select Tape Unit 0
JPSR        9RE     [ Call Rewind Routine
```

A backspace magnetic tape operation is specified by the following instruction set:

```
MDAR'F      2       [ Select Tape Unit 1
JPSR        9BA     [ Call Backspace Routine
```

A write file mark on magnetic tape operation is specified by the following instruction set:

```
MDAR'F      4       [ Select Tape Unit 3
JPSR        9EN     [ Call End Of File Routine
```

ARGUMENT LIST CALLS

The I/O Argument List Call is an optional declaration, and is required only if there is a list associated with the I/O function. If used, it is composed of three elements:

Argument Size
Argument Type
Argument Address

Argument Size

Argument size specifies the number of words for the argument. This number is placed in the A-register.

Argument Type Call

The argument type call initiates the call to the linkage routine in the form:

```
JPSR    9Ik
```

where: k specifies the argument type: 1 = integer, and 2 = real.

Argument Address

The argument address specifies the address of the argument. All arguments in the list must be of the same mode and must have the same argument size. The parameter is of the form:

```
0    L1
0    L2
  .
  .
  .
0    Ln
```

where: L_2^* is the address of the argument.

The use of the I/O Argument List Call in conjunction with the I/O Initiate Call is demonstrated in the following example:

```
MDAR        50      [ Select Typewriter
JPSR        9WR     [ Call Write Routine
0           F       [ Address of Format Specification
MDAR'F      5       [ Five Words for Each Argument
JPSR        9I1     [ Argument List is of Type Integer
0           J       [ Argument Address
0           K       [ Argument Address
0           L       [ Argument Address
```

The Argument List in this example would cause the data at locations J through J + 4, K through K + 4, and L through L + 4 to be typed on the typewriter according to the format specification in location F.

INPUT/OUTPUT TERMINATION

The I/O Terminate Call provides the necessary linkage to complete the I/O operation and terminate any incomplete records.

I/O Terminate Call

The I/O Terminate Call initiates a call to the linkage routine in the form:

```
JPSR    9ND
```

The sequence of instructions required then, for an I/O operation not using an Argument List, can be demonstrated in the following example:

```
MDAR'F  51      [ Select Paper Tape
JPSR    9WR     [ Call Write Routine
0       A       [ Address of Format Specification
JPSR    9ND     [ I/O Terminate
```

where: A is a Hollerith type Format Statement, e.g., (5HABCDE).

A sequence of instructions required for an I/O operation using an Argument List is shown below:

```
MDAR'F  50      [ Select Typewriter
JPSR    9WR     [ Call to Write Routine
0       F       [ Address of Format Specification
MDAR'F  5       [ Five Words for this Argument
JPSR    RI1     [ Argument List of Type Integer
0       J       [ Argument List Address
JPSR    9ND     [ I/O Terminate
```

A sequence of instructions for an I/O operation requiring an Argument List of different sizes and data types is shown below:

```
MDAR'F  50      [ Select Typewriter
JPSR    9RD     [ Call Read Routine
0       F       [ Format Address
MDAR'F  1       [ One Word Per Argument
JPSR    9I1     [ Type Integer
```

```
0         J       [Argument List Address
0         K       [Argument List Address
MDAR'F    1       [One Word Per Argument
JPSR      R12     [Type Real
0         R       [Argument List Address
0         S       [Argument List Address
MDAR'F    20      [Twenty Words per Argument
JPSR      9I2     [Type Real
0         T       [Argument List Address
JPSR      9ND     [I/O Terminate Call
```

FORMAT CONTROL

Formatted data is described by a format descriptor of one of the following forms:

I	Integer Data
F	Fixed Point Real Data
E	Floating Point Real Data
H	Hollerith Data
X	Blank Data
/	Record Terminator

NOTE

In the following examples of data types, b represents a blank character.

Integer Data

Integer data is input/output with an I/O Initiate Call and an Argument List Call to the routine 9I1. The format specification must be of the form Iw; where w is the width of the external field.

Integer Input. -- The external value is right-justified to the width w. All blanks are treated as zeros. The value may be preceded by a + or - sign character. If no sign is indicated, the value is assumed to be positive. Any other non-numeric character in the field is an input error. The maximum numeric value allowed for an integer number is 536870911. An input value exceeding this limit causes an input error.

Examples of integer input are shown on the following list:

Input Format	External Characters	Internal Value	
I6	b12345	+12345	
I7	b+12345	+12345	
I7	b-12345	-12345	
I8	bbbbbbbb	0	
I8	bbb12b45	+12345	
I9	536870911	+536870911	
I9	536870912		ERROR
I8	123-45		ERROR
I8	ABCDE		ERROR

Integer Output. -- The characters output are right-justified in the field width n. Negative numbers are preceded by a minus sign. If the specified width of the field is smaller than the number of characters required for a number, an * character is placed in the least significant character position.

Examples of an integer output are shown in the following list:

Output Format	Internal Number	Characters Output
I8	+12345	bbb12345
I7	-12345	b-12345
I6	+12345	b12345
I5	+12345	12345
I5	-12345	-123*
I4	+12345	123*
I3	-12345	-1*
I2	+12345	1*
I1	-12345	*
I0	- FORMAT ERROR	
I5	0	bbbb0

Real Data

Real data is input/output with an I/O Initiate Call and an Argument List Call to the 9I2 routine. The format specifications must be of the form Fw.d or Ew.d; where w is the width of the external data field and d is the number of decimal places.

Real Input. -- The external value is right-justified to the width w, with d decimal places. For type E format, the two least significant characters are considered to represent the decimal exponent. All blanks are treated as zeros. The number may be preceded by a + or - sign character. If a sign is not indicated, the number is considered to be positive. The external data field may contain the decimal point character (.), which determines the position of the decimal point and overrides the d specification.

The character E in an internal field specifies a decimal exponent of up to two characters. The exponent may be preceded by a + or - sign. If an E is present, and a decimal point is not specified, the decimal place is considered to be d positions to the left of the E character. Any non-numeric character in the data field other than those described above causes an input error.

Real numbers must lie in the range $2^{-127} \leq R \leq 2^{127}$ ($0.588 \times 10^{-38} \leq R \leq 0.588 \times 10^{+38}$ approximately). On input, numbers outside the range will produce an input error. When input, any real number can be considered to be of the form I.E±n, where I is a decimal integer and n is the decimal exponent. If I is in the range 100000000X ≤ I ≤ 536870911X (where X is a string of decimal digits), the X digits will be dropped and the exponent increased by the number of digits in X. (e.g., 1234567897234.E+2 will be treated as 123456789.E+6).

Examples of real data input are shown in the following list:

Input Format	External Characters	Internal Value
F6.3	b12345	+.12345E+2
F6.3	-12345	-.12345E+2
F6.3	1234.5	+.12345E+4
F6.3, E6.3	1.2E04	+.12000E+5
F6.3, E6.3	1.20+4	+.12000E+5
F6.3, E6.3	bbbbbb	+.00000E+0
F6.3, E6.3	212E+4	+.21200E+4
F6.3, E6.3	212E-4	+.21200E+4
E6.3	b12345	+.12300E+45

Input Format	External Characters	Internal Value
E6.3	-12345	-.12300E+45
E6.3	1.2345	+.12300E+46
E6.3	b12345.	INPUT ERROR
F6.3, E6.3	12E+39	INPUT ERROR
F6.3, E6.3	12.E5.	INPUT ERROR
F6.3, E6.3	ABC	INPUT ERROR

Real Output. -- The characters output are right-justified in the field width w. Negative numbers are preceded by a minus sign. If the size of a number exceeds the field width, the least significant character is replaced by an asterisk (*). Formal type E must allow four character positions for the exponent, and one character position for the decimal point.

Examples of real data are shown in the following list:

Output Format	Internal Number	Characters Output
E12.4	-123456.	b-0.1235Eb06
E12.4	+.012346	bb0.1235E-01
E11.4	+123456	b0.1235Eb06
E11.4	-.000123	-0.1230E-03
E10.4	-123456.	-.1235Eb06
E9.4	+123456.	.1235Eb06
E9.4	-123456.	-.12*Eb06
E9.4	0	.0000Eb00
F6.0	0	bbbb0.
F6.3	+.123456	b0.123
F6.3	-.123456	-0.123
F6.3	+1.23456	b1.235
F6.3	-1.23456	-1.235
F6.3	+12.3456	12.346
F6.3	-12.3456	-12.3*
F6.3	+123.456	123.4*
F6.3	-123.456	-123.*

Hollerith Data

The Hollerith format descriptor is of general form: n H k where: n is an integer, H is the format descriptor, and k is a string of n Hollerith characters within the Format Specification.

Hollerith Input. -- For input, n characters from the external field are placed into the string k in the format.

Hollerith Output. -- The integer n specifies the number of k characters that will be transmitted from the format to the external field. If the character string k contains more than n characters, characters starting with n + 1 will be interpreted as format descriptor characters, with resulting errors.

The X Format Descriptor

The X format descriptor is in the general form: n X
where: n is an integer and X is the format descriptor.

X Format Input. -- With the X format input, n characters are skipped in the external field.

X Format Output. -- The integer n specifies the number of blank characters to be transmitted. Examples of H and X output are shown in the following list:

Format Descriptor	Characters Output
10X	bbbbbbbbbb
3X	bbb
X	b
12HADDb(A,B*D)b	ADDb(A,B*D)b
5HXYZ	XYZbb
4HXYZ	XYZb
3HXYZ	XYZ
2HXY	XY
1HX	X
HXYZ	FORMAT ERROR

Examples of H and X format input are shown in the following list:

Format Description	External Field	Resultant Format
6HABCDEF	123456	6H123456
4Hbbbb	ABCD	6HABCD
4H1234	bbbb	4Hbbbb
3HABC,2X,1HA	123456	3H123,2X,1H6

Record Terminator

The / format descriptor causes termination of a record. On input, any remaining unprocessed data is ignored and further data is input from the next record. On output, the record is output, and further data in the I/O operation is placed in the next record.

INPUT/OUTPUT ERROR MESSAGES

Error conditions that occur during execution of an input/output operation cause one of the following messages to be typed on the typewriter.

1. FORMAT ERROR - This message indicates the format is in error.

2. MODE ERROR - This message indicates the mode of the argument, integer or real, does not agree with the format specification.

3. DATA ERROR - This message indicates the input data is in error. The error may be that the value is too large or the file contains an illegal character.

4. UNIT ERROR - This message indicates a logical unit number outside the range 1 to 51 has been specified.

5. TAPE ERROR - This message indicates a non-recoverable magnetic tape failure has been encountered. After the message is typed, execution of the program is terminated and control is transferred to the exit routine.

CHARACTER CODES

The following table provides a listing of the external and internal character representations used in AFORT:

Character	Magnetic Tape and Internal Code	External Code
0	01	20
1	02	21
2	03	22
3	04	23
4	05	24
5	06	25
6	07	26
7	10	27
8	11	30
9	12	31

Character	Magnetic Tape and Internal Code	External Code
A	20	41
B	21	42
C	22	43
D	23	44
E	24	45
F	25	46
G	26	47
H	27	50
I	50	51
J	51	52
K	52	53
L	53	54
M	54	55
N	55	56
O	30	57
P	31	60
Q	32	61
R	33	62
S	34	63
T	35	64
U	36	65
V	37	66
W	40	67
X	41	70
Y	42	71
Z	43	72
+	60	10
-	61	35
*	62	05
/	63	37
.	64	36
(	65	16
)	66	17
,	67	34
=	70	33
.	72	06
%	73	01
$	74	73
#	75	74
C/R	77	15
SPACE	00	40

UTILITY ROUTINES

Utility routines are contained as part of the Operating Routines loaded with the loader. The utility routines perform basic functions necessary for execution of most FORTRAN programs. Following is a list of these routines and their functions:

9DO	Performs the incrementing and testing of a DO loop index. Receives parameters for increment, index, limit, and start of loop.
9CG	Performs the computed GO TO statement. Receives the index value in the A-register and the transfer locations as parameters.
9IF	Performs the IF statement. Receives the expression value in the A-register and the transfer locations as parameters.
9ST	Performs the STOP statement.
9PA	Performs the PAUSE statement.
9TI	Types the TRACE of an integer value.
9TR	Types the TRACE of a real value.
9TG	Types the TRACE value of a real IF statement.
9TS	Types the TRACE of a statement number.
ERFUN	Sets the arithmetic error designator bits.
EXIT	Performs the termination of execution of a program.

These routines are primarily used to implement compiled statements and are not compatible with normal subroutine usage. For further description of their behavior and calling sequences refer to Adage Doc. OBJPK/SMM.

INTRODUCTION

PRIO is an AMOS system program written in the ADEPT assembly language. PRIO provides ADEPT and AFORT with the necessary interface to I/O devices to obtain source text input and to output object machine code in relocatable format.

SOFTWARE REQUIREMENTS

Version 1 - AMRMX versions 1, 2, 3, 4, 5, 6, 11, or 12.

HARDWARE REQUIREMENTS

Hardware requirements are specified by the appropriate version of AMRMX being used.

PRIO occupies approximately 440_8 locations of core memory.

PROCESSOR I/O ROUTINES

A. Text Input

The "current" input text to be processed by all AMOS processors is taken from an ATEXT file in the file area or the scratch pad of the currently assigned input tape.

The routines by which processors access the current text are:

1. INITI - Initialize Input prepares (or reprepares in the case of multiple passes) the "current text" for input from the beginning of the first page. This subroutine initializes the next input routine, ICH, to start scanning the text in the scratch pad area of the scratch tape (unit 0).

Calling Sequence

```
[Scratch pad unit 0 = text to be input
        JPSR        INITI       [Call
                                [Return
```

2. ICH - Input Character subroutine fetches successive characters of the current input text into AR[24-29]. As each page is exhausted, the next one is fetched via FPB.

Calling Sequence

```
JPSR    ICH    [Call
               [Return
```

Results

(AR[24-29]) = next character of current input text.

3. FPB - Fill Page Buffer subroutine is called by INITI and occasionally by ICH in order to enter the next page of the current text and reset ICH to scan it.

Calling Sequence

```
JPSR    ICH    [Call
               [Return
```

NOTE

If processor output is currently being generated on the same tape as the text input tape, FPB will first close the library file area, and later reposition the tape for subsequent output after it has found and read the next text input page.

B. Object Output

All object output generated in the AMOS System is appended to the user-file area of the currently assigned System Tape. The I/O Package provides facilities to:

1. INITO - Initialize Output subroutine prepares the currently assigned system tape to accept a set of relocatable files comprising object code generated by a language text processor. This includes positioning of the system tape, establishing the write parameters (the title is typed out), and resetting the output routines, buffers, pointers, and counts.

Calling Sequence

```
JPSR    INITO    [Call
                 [Return
```

2. OAB - Output Absolute, ORE - Output Relative, OCM - Output Common, and OSP - Output Special

These subroutines load the contents of (AR) into the output buffer, and set the appropriate matrix code. For OSP, (AR) must have the subcode in bits 0-5.

Calling Sequences

[(AR) = string to be output

```
JPSR    OAB     [Call
  .               [Return
JPSR    ORE     [Call
  .               [Return
JPSR    OCM     [Call
  .               [Return
JPSR    OSP     [Call
  .               [Return
```

3. SOM - Set Output Matrix subroutine loads the contents of (Call + 1) [28-29] - 3 into the next spot in the output matrix and preserves RA. Called by one of the output (AR) routines OAB, ORE, OCM, or OSP, this routine sets the appropriate matrix code and then calls OWD to output (AR).

Calling Sequence

[(AR) = string to be output

```
JPSR    SOM     [Call
          1     [i = 0, 1, 2, 3
  .             [Return
```

4. OWD - Output Word subroutine loads the contents of (AR) into the output buffer. When the buffer is full, it is written, then pointers and counters reset.

Calling Sequence

[(AR) = string to be output

```
JPSR    OWD     [Call
  .             [Return
```

5. RSOT - Reset Output Routines

This subroutine initializes the object-code outputting routines. It clears the output buffer to -0s, resets the current entry pointer to the first word, and resets the count of relocation-types in the buffer.

Calling Sequence

```
JPSR        RSOT        [Call
  .                     [Return
```

6. SOT - Set Object Output Tape Parameters

This subroutine instates the "title" and "type" parameters for use by the magnetic tape write routines when called by object output routines.

Calling Sequence

```
JPSR        SOT         [Call
  .                     [Return
```

7. SETN - Set Output Name

This subroutine obtains a right-justified blank-filled name and left-justifies it, filling with nulls, and then establishes it as the "title" for any subsequent object records output.

Calling Sequence

```
JPSR        SETN        [Call
                        [Return
```

8. DROP - Delete Past and Suspend Further Output for Current File

This subroutine insures that any output for the current program will be deleted, and that any subsequent output generated will not be written.

Calling Sequence

```
JPSR        DROP        [Call
                        [Return
```

9. TOT - TERMINATE OUTPUT

This subroutine is used by processors (Compilers, Assemblers) to terminate the outputting of one object program and initialize the output routines for outputting the next one as a new file.

Any remaining room in the buffer is fulled with type "END" object records until the buffer gets written and reset. If the current program has been DROPPED and not removed from the output tape, it will be removed on the first output call.

The DROP flag is reset and the last-record-count is cleared, forcing the next write to start a new file.

Calling Sequence

(AR) = length of object program

```
JPSR        TOT        [Call
                       [Return
```

10. CLOSE - CLOSE TAPE LIBRARY FILE

This subroutine is used to close the user-file area after having added object program files to it. The system tape is selected, and the terminal file mark is generated.

Calling Sequence

```
JPSR        CLOSE      [Call
                       [Return
```

11. NOMOR - Terminate Output and Close Library

This subroutine is used by the processors to both terminate the object program output and close the user file area.

Calling Sequence

```
JPSR        NOMOR      [Call
                       [Return
```

INTRODUCTION

The RADC subroutine is used in the AGT/10 with the AMC1-P1,P2 comparator option. The comparator is used by RADC to convert an analog input into a 10-bit digital value. RADC is called by those versions of RVCD, RJSB, and RADT which operate on the AGT/10.

USE

Select a CHANNEL(S) to be digitized by loading the proper multiplexor bit(s) and then specify:

```
L:        JPSR    $RADC
L + 1:        Returns here
```

This will digitize the sum of the input device(s) selected and will return with the value in the AR[15-24](1's complements for negative numbers, sign extended).

STORING AND TIMING

RADC occupies less than 40_{10} words of core and requires 120 μsecs to execute and retrieve a 10-bit value.

NOTE: RADC destroys the current transformation of the array. For this reason, it must not be processed asynchronously with any program which uses the array (such as DSPLY). Any programs which call RADC must not be chained on the CLOCK routine of the DSPLY operator.

GENERAL

RADT is an AMOS library routine which is used to read the X and Y coordinates on the ADT (Analog Data Tablet) to determine the pen-up/pen-down condition of the ADT stylus tip, and to detect the "pen-depressed" position of the stylus tip.

CALLING SEQUENCE AND USE

When used with DSPLY, RADT may be appended to the CLOCK chain and executed once per frame by typing the statement:

CLOCK (RADTK, RADTR)!

Result:

The tablet is sampled once each time the picture on the CRT is refreshed.

RADTX: X value in bits 15-29 (Sign extended bits 0-14)
RADTY: Y value in bits 15-29 (Sign extended bits 0-14)

The X and Y values range from -37777 (minus full scale) to 37777 (full scale). The center of the data tablet is defined as the origin (Ø, Ø). When the pen stylus is not positioned on the data tablet, the flag ADTF1 is set to -1, and the values of RADTX and RADTY do not change; otherwise, ADTF1 is set to +Ø. When the tip of the stylus is depressed hard enough to engage the "pen-depressed" switch, the flag ADTF2 is set to -1; otherwise, ADTF2 is set to +Ø.

NOTE

RADT may be called once per frame with a "JSR RADTR" image item or at any time by executing a "JPSR RADTR" instruction in memory.

To obtain the four variable values in an AFORT program, use the calling sequence:

CALL RADTV (I1, I2, I3, I4)

where: I1 is RADTX, I2 is RADTY, I3 is ADTF1, and I4 is ADTF2. RADTX and RADTY are integers ranging from +16,384 to -16,384. The RADTV subroutine presumes that RADT is on the CLOCK chain or has been called with a CALL RADTR.

DESCRIPTION

RANK is a relocatable AMOS library routine which may be used to read characters from any of four ANK-P1 Keyboard sybsystems. RANK may also be interfaced with the AMRMX and EDIT system programs so that they may receive input from the alphanumeric keyboard by the program RANKC.

REQUIREMENTS

RANK requires the EAU subsystem. It also requires the implementation of the programmatic interrupt at the lowest priority level and must be used with the corresponding version of AMRMX.

Version 1 of RANK is for systems with only one ANK; Version 2 of RANK is for systems with up to 4 ANK-Keyboards.

USE

A. Initiate Input Routine

The calling sequence:

```
JPSR    RKCHn       n = 1 for Version 1
        DONE        n = 1, 2, 3, or 4 for Version 2

                    Returns to next location
```

Enables keyboard n to accept an input character. This subroutine is "open". That is, it first initiates the keyboard character routine and then returns to the location after the calling sequence. It executes instruction DONE after the character has been input. With the 7-bit character is the 7-bit reversed ASCII character (compatible with the TTY input characters) contained in AR bits [22-28]. If the instruction DONE causes control to be returned to the next location when it is executed, the foreground environment will be restored. If RKCHn is called again before the character has been accepted, the program will wait until the previously specified DONE instruction is executed.

B. Wait for Input Routine

The calling sequence:

```
JPSR    RKCWn       n = 1 for Version 1
        CHAR        n = 1, 2, 3, or 4 for Version 2

                    Returns to next location
```

Waits for keyboard n to input a character. The result is the 7-bit reversed ASCII character contained in AR bits [22-28] and also in the location referenced by CHAR, which may be an indirect chain. Thus, RKCWn may be called by AFORT programs.

DESCRIPTION

RANKC is a relocatable AMOS library routine used to interface the RANK program (Read Alphanumeric Keyboard) with AMRMX and EDIT. Using RANKC, AMRMX and EDIT may receive control statements from an ANK1 or ANK2 Alphanumeric Keyboard subsystem.

USE

The calling sequence:

```
JPSR    SYSC
        n
```

Where: n = Ø, 1, 2, 3, 4

This call causes RANK to interface with the basic teletype input of AMRMX thus allowing EDIT, as well as AMRMX, to receive input from keyboards 1, 2, 3, or 4. When n is Ø, the input function is restored to the TTY. If n is omitted, it is assumed to be Ø

When Version 1 RANK is used, arguments of n = 1, 2, 3, or 4 all cause input to be from the one alphanumeric keyboard on the system.

HARDWARE REQUIREMENTS

RANK may be used on an AGT with an EAU subsystem and a programmatic interrupt. It links with the AGT Monitor (AMRMX, Version 11 or 12) and the program RANK (Version 1 or 2).

NOTE:

The call:

```
JPSR    SYSC
        n
```

is the equivalent to Monitor statement SYSC(n)!

INTRODUCTION

RCD is a set of relocatable subroutines which are used to read punched cards on the CDR1 card reader. RCD contains both open routines for flexibility and closed routines for user convenience. Each call to RCD will cause a single card to be read.

VERSION

RCD exists in only one version.

SOFTWARE REQUIREMENTS

RCD makes use ot the AMRMX routines to link subroutines at the PINT priority level.

HARDWARE REQUIREMENTS

RCD requires the CDR1-P1 subsystem. RCD occupies 505_8 words of core storage.

TIMING CONSIDERATIONS

The CDR1-P1 provides interrupts for each character every 2.4 ms. The longest the computer will be held at the card reader interrupt level for each character is:

Code	Time
-Ø	15Ø μs
Ø	35Ø μs
1	39Ø μs

In all cases RCD must be allowed to operate at its priority level at least 15% of the time

The done or error instructions, which require less than 1ØØ μs at the CDR1 level to be set up, are completed at the PINT level.

CARD READER ROUTINE

A. RCD - Read Card (Open Call)

The calling sequence:

```
JPSR    RCD          [ Entry
          .          [ Busy return
        TABLE        [ Location of table
        CODE         [ Conversion code
        DSAB INST    [ Instruction if machine not enabled
        ERR INST     [ Instruction if error
        DONE INST    [ Instruction when done
          .          [ Normal return
```

will initiate the reading of a single card into a table starting at TABLE. The program will return through the normal return and the reading will continue on an interrupt basis. If the RCD program is in operation at the time of the call, it will return through the busy return and no action will be taken.

RCD will read the following codes:

Code	Card Code Conversion
-Ø	Binary input. Each column on the card is interpreted as two 6-bit binary characters. One card will fill a table of 40_8 words.
Ø	Hollerith input. Each column on the card is translated into the standard AMOS code according to the table in this document (or a table which has been substituted by SRCDT). One card will fill a table of 20_8 words.

Code	Card Code Conversion
1	Hollerith input. Does the same as Ø above except the 11-8-2 and 12-8-2 punches are replaced by the 11-Ø and 12-Ø punches respectively.

The DSAB INST is executed if RCD is called when the card reader is turned off or not enabled. The instruction is executed with a ONE in the AR. RCD will return immediately through the normal return with the contents of AR after the DSAB INST still in the AR.

If the AR is left negative by the DSAB INST (e.g., -DSAB INST = ARAR'N'F) the program will: 1. Set up the instructions to read a card when the card reader is enabled; 2. Retain the "busy" state to prevent additional RCD calls; and 3. Return immediately through the normal return.

The ERR INST will be executed if an error is detected by the CDR1 or an illegal code is detected in the Hollerith translation. The instruction is executed at the PINT priority level with one of the following codes in AR:

10_2 error detected by CDR1.

100_2 illegal code character.

The DONE INST is executed when the card read is complete. The instruction is executed at the PINT priority level with a Ø in AR.

NOTE

Care should be taken to make sure that the DSAB INST, ERR INST, and DONE INST do not contain instructions which will release their interrupt level and that they return to their calling sequence.

B. RCDC - Read Card (Closed Routine)

The calling sequence:

```
JPSR    RCDC     [ Entry
        TABLE    [ Location of table
        CODE     [ Conversion code
        ERR      [ Return if error
         .       [ Normal return
```

will read a single card into a table starting at TABLE. CODE will be as in RCD.

The program will not return until either an error is detected or the card has been read. If the error return is taken, the error code will be in AR as follows:

1	CRD1 Off Line or Not Enabled
10_2	Error Detected by CDR1
100_2	Illegal Code Character
1000000_2	RCD Busy

C. SRCDT - Set RCD Code Table

The calling sequence:

```
JPSR    SRCDT     [Entry
        ADDR      [Address of new code table
          .       [ Return
```

will cause a table of 20_8 words starting at ADDR to be loaded into the RCD code table. The code words are packed four 6-bit characters, left justified per word, according to the following chart. If ADDR = -ø, the standard AMOS code will be restored.

	ZONE PUNCHES					
ø	NONE	ø	11	12_{29}		
ADDR:					1	BIT PUNCHES
					2	
					3	
					4	
					5	
					6	
					7	
					9	
					none	
					8 2	
					8 3	
					8 4	
					8 5	
					8 6	
					8 7	
					8	

CHARACTER SET

TTY CHAR.	AMOS CODE	CARD PUNCH	TTY CHAR.	AMOS CODE	CARD PUNCH
[	00	12-8-2*	space	40	none
%	01	0-8-4	A	41	12-1
]	02	11-8-2*	B	42	12-2
!	03	12-8-7	C	43	12-3
&	04	12	D	44	12-4
*	05	11-8-4	E	45	12-5
:	06	8-2	F	46	12-6
	07	0-8-2	G	47	12-7
+	10	12-8-6	H	50	12-8
tab	11	12-8-4	I	51	12-9
?	12	0-8-7	J	52	11-1
"	13	8-7	K	53	11-2
'	14	8-5	L	54	11-3
C/R	15	0-8-6	M	55	11-4
(	16	12-8-5	N	56	11-5
)	17	11-8-5	O	57	11-6
0	20	0	P	60	11-7
1	21	1	Q	61	11-8
2	22	2	R	62	11-9
3	23	3	S	63	0-2
4	24	4	T	64	0-3
5	25	5	U	65	0-4
6	26	6	V	66	0-5
7	27	7	W	67	0-6
8	30	8	X	70	0-7
9	31	9	Y	71	0-8
;	32	11-8-6	Z	72	0-9
=	33	8-6	$	73	11-8-3
,	34	0-8-3	#	74	8-3
-	35	11	@	75	8-4
.	36	12-8-3		76	11-8-7
/	37	0-1		77	0-8-5

*CODE 1 will replace 12-8-2 and 11-8-2 with 12-0 and 11-0 respectively.

GENERAL

Read Relocatable Symbols subroutine implements the Monitor Statement READS. It causes local assembly symbols (of type RLSYM) saved in the file "TITLE" to be defined and properly relocated.

CALLING SEQUENCE AND USE

Calling Sequence:

```
JPSR READS
"TITLE"        [TITLE of requested file
TAPE           [Tape unit number = Ø, 1, etc.
                 (SYSTN if omitted)
Ø              [End argument-list
•              [Returns to next location
```

READS eliminates any local tags defined in the external symbol table and selects the tape unit, then finds the requested file. If it is not found, READS then types the message "FILE NOT FOUND" and returns to the Monitor.

The READS subroutine reads the file into the external symbol table. (See DUMPS abstract for a definition of the format of the file.) The true relocation constant is computed from the ENTRY value in the new file (M) and its value in the Monitor's symbol table (X), provided that the program has been loaded. If it has not been loaded, READS will cause it to be loaded and the program will continue. The relocation constant has the value (X-M). If the source program specified no entry points, it has the value 12Ø. All relocatable symbols will have their values incremented by the value of the relocation constant.

INTRODUCTION

The Retrieve Graphics Operator enables selection and reading of tape or disk files from the System Library. The input, consisting of relocatable records in a type "IMAGE" file, may have been generated by a SAVE operation. The image segment(s) in the files are reinstated in the current core resident BUILD Temporary Library Table. The additional segment(s) may be used in subsequent BUILDing operations.

DESCRIPTION

The following operation:

$$\text{RETRV}\left(\text{"title"},\left\{\begin{matrix}\text{unit}\\ \text{or}\\ \text{vol}\end{matrix}\right\}\right)!$$

where:

title	=	The name (title) of the relocatable IMAGE file to be read.
unit	=	The input tape unit or, if omitted, the current "system tape".
vol	=	The input volume number if unput is to be from disk. If omitted, the current "system volume" is assumed.

Causes the image segment(s) in the input file to be read (appended) into BUILD's Temporary Library Table.

VERSIONS

Version 1	=	AGT Tape System
Version 2	=	AGT Disk System

CORE REQUIREMENTS

500_8 words

GENERAL

RJSB is one of a set of AMOS system programs in the subroutine library that interfaces on-line terminal I/O devices with the user's calling program. RJSB samples the input from the JSB1-P1, P2 subsystem. In Version 2 RJSB may be chained to the CLOCK facility in the DSPLY operator, thereby continually sampling the joystick at the specified frame rate.

The RJSB program defines five external symbols: entry points RJSB, RJSBV, and the three input addresses, JSBX, JSBY, JSBZ.

In addition, Version 2 of RJSB defines entry points JSBK and JSBR.

USE (Versions 1 and 2)

The calling sequence:

```
JPSR    $RJSB
```

causes all dials to be sampled and their values stored in locations JSBX, JSBY, and JSBZ respectively. Do not use this call if the RJSB program is chained to the clock.

Version 2 only:

```
JPSR    $CLOCK    [CLOCK (JSBK, JSBR)!]
        $JSBK
        $JSBR
```

causes the RJSB program to be chained to the CLOCK subroutine of the DSPLY operator. This will cause the dials to be sampled once per frame and the results to be stored in JSBX, JSBY, and JSBZ.

To obtain the values JSBX, JSBY, and JSBZ from an AFORT program, use the following call:

CALL RJSBV (A1, A2, A3)

where: A1 is JSBX, A2 is JSBY, and A3 is JSBZ.

JSBX, Y, and Z are integers between 16,384 and -16, 384. The RJSBV subroutine presumes that RJSB is on the CLOCK chain or has been called with a CALL RJSB.

HARDWARE REQUIREMENTS

AGT with GHA1 or GHA2.

Version 1 - for AGT/10, JSB1-P1, AMC1-P1

Version 2 - for AGT30 or AGT/50, JSB1-P2

SOFTWARE REQUIREMENTS

Version 1 - RADC, Read analog digital with comparator

Version 2 - DSPLY operator, if RJSB is to be chained to the CLOCK

STORAGE AND TIMING

Version 1 - RJSB together with the RADC routine, occupies less than 100_{10} locations in core. To digitize the three input variables requires 380 μs of time.

Version 2 - RJSB occupies less than 60_{10} locations in core. To digitize the three input variables requires 95 μs of time.

GENERAL

The RVCD program is one of a set of AMOS system programs in the subroutine library that interfaces on-line terminal I/O devices with the user's calling program. RVCD samples the input from the VCD1-P1/P2 subsystem and may be chained to the CLOCK facility in the DSPLY operator, thereby continually sampling the dials at a frame rate specified by the operator.

Versions 1 and 2 of the RVCD program define nine external symbols: entry points RVCD, RVCD1, RVCD6, and the six dial addresses, VCDA, VCDB, VCDC, VCDD, VCDE, VCDF.

In addition, Version 2 defines entry points VCDK, VCDR.

CALLING SEQUENCE AND USE (Versions 1 and 2)

The Calling Sequence:

```
JPSR    $RVCD
```

causes all the dials to be sampled and their values stored in locations VCDA through VCDF, respectively. Do not use this call if the RVCD program is chained to the clock.

Version 2 only:

```
JPSR    $CLOCK      [CLOCK (VCDK, VCDR)!]
        $VCDK
        $VCDR
```

causes the RVCD program to be chained to the CLOCK subroutine of the DSPLY operator. This will cause the dials to be sampled once per frame and the results to be stored in VCDA - VCDF.

To obtain the values VCDA - VCDF from an AFORT program, use either of the following calls:

CALL RVCD1 (AA, A1) causes the contents of dial AA (1-6 VCDA - VCDF) to be loaded into variable A1.

CALL RVCD6 (A1, A2, A3, A4, A5, A6) will load variables A1 - A6 with VCDA - VCDF contents respectively. VCDA - VCDF are integers between +16,384 and -16,384. The RVCD1 and RVCD6 subroutines presume that RVCD is on the clock chain or has been called with a CALL RVCD.

HARDWARE REQUIREMENTS

AGT with GHA1 or GHA2

VCD1-P1 or P2

AMC1-P2 (Version 1 only)

VERSIONS

Version 1 - AGT/10, VCD1-P1, AMC1-P2

Version 2 - AGT/30 or AGT/50, VCD1-P2

SOFTWARE REQUIREMENTS

Version 1 - RADC

Version 2 - DSPLY operator, if RVCD is to be chained to the clock.

STORAGE AND TIMING

Version 1 - The RVCD and RADC subroutines occupy less than 100_{10} locations in core. To digitize the six dials requires 750μs of time.

Version 2 - RVCD occupies less than 90_{10} locations in core and requires 161μs to digitize the six dials.

INTRODUCTION

The SAVE Graphics Operator enables the filing of image items into the System Library on either mag tape or disk. These image items may comprise one or more segments in the BUILD Temporary Library Table or may be any complete image descriptions in core. The output consists of relocatable records in a type "IMAGE" file.

DESCRIPTION

The following operation:

$$\text{SAVE}\left(\text{"title"},\ \left\{\begin{matrix}\text{unit}\\ \text{or}\\ \text{vol}\end{matrix}\right\},\left(\left\{\begin{matrix}\text{image name}\\ \text{or}\\ \text{segment name}\end{matrix}\right\},\ \ldots\right)\right)\ !$$

where:

title	=	the name (title) of the relocatable Image file to be output.
unit	=	the output tape unit number or, if omitted, thecurrent "system tape".
vol	=	the output volume number if output is to be on disk. If omitted, the current "system volume" is assumed.
image name	=	the entry point name (or location) of a complete image description (with 2 word header, etc.)
segment name	=	the name of a segment currently defined in BUILD's Temporary Library Table (done by means of a BUILD PUT operation).

Causes the output of a file on either tape or disk of type "IMAGE".

These files may subsequently be read from the library on tape or disk by LOAD or RETRV operations.

VERSIONS

1 - For use under Mag Tape Operating Systems
2 - For use under Disk Operating Systems

STORAGE REQUIREMENTS

The SAVE operator occupies less than 1K of storage.

INTRODUCTION

SCCPY is an ADEPT language program for copying TEXT or ATEXT files on magnetic tape. SCCPY is primarily used on one-tape systems for copying text files from and to the "Scratch Pad" area of the tape.

OPERATION

The following statement is input to the AMOS Monitor, or given by appropriate machine language:

SCCPY (RECORD, FILE, INPUT, OUTPUT, DENS1, DENS2)!

where RECORD is the first record of the input file.

FILE is the input file number.

INPUT is the input tape number (0, 1, 2, or 3).

OUTPUT is the output tape number (0, 1, 2, or 3, with - for output to scratch pad).

DENS1 is the input tape density (0 for 200 bpi, 1 for 556 bpi, or 2 for 800 bpi).

DENS2 is the output tape density (0, 1, or 2 as above).

NOTE

Density arguments are ignored on MTP5/8 systems.

This statement causes the specified text FILE to be written on the output tape starting with the specified RECORD. If the specified OUTPUT tape number is preceded by a minus sign (-), output is to the "scratch pad" (file 10_8) of the output tape. Otherwise, the output is to the end of the output tape. Where the arguments are the same as the above, except that when output to the "scratch pad" is specified, input records replace the contents of the existing "scratch pad."

The following statement:

SCCPA (RECORD, FILE, INPUT, OUTPUT, DENS1, DENS2)!

is equivalent to the first statement except that when output to the "scratch pad" is specified, input records are appended to the existing contents of the "scratch pad."

When finished, the program writes a file mark on the output tape if output has not been to the "scratch pad," and rewinds both tapes. If output has not been to the "scratch pad," the program types FILE = N, where N is the output file number. On MTP-7 systems, magnetic tape density will be left at the output density, DENS2. The program then returns to the caller.

GENERAL

AMOS Set Date Routine, SDATE, is an AMOS library program written in the ADEPT assembly language used for setting the current date cell in the AMOS Resident Monitor.

CALLING SEQUENCE

SDATE is called by the Monitor control statement

SDATE!

or by a subroutine call in a user program of the following form:

```
LL:       JPSR      $SDATE
LL+1:     Returns here
```

USE

On entry, SDATE types the following message:

TYPE-IN DAY OF MONTH...

The operator should then type in the day of the month, a decimal integer from 1 to 31, terminated by a carriage return character.

Next, SDATE will type:

TYPE-IN SYMBOLIC MONTH...

The operator should then type the symbolic month, terminated by a carriage return character. The entire month word need not be typed as long as the characters typed are sufficient to uniquely specify the desired month. For example, "J(C/R)" is insufficient as the months "JANUARY," "JUNE," and "JULY" all begin with "J." The sequence "JA(C/R)" is sufficient to specify "JANUARY," while "JUN" and "JUL" must be present to specify "JUNE" or "JULY" respectively.

After the month has been typed in, SDATE will type:

TYPE-IN YEAR (4 DIGITS)...

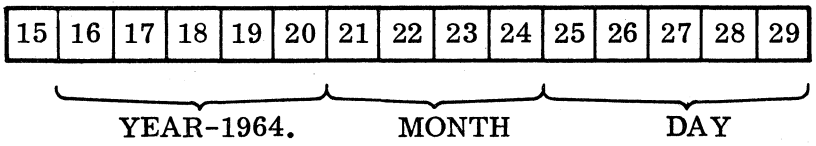
The operator should then type in the year terminated by a carriage return character. Permissable years lie in the range 1964 through 1995.

SDATE will then evaulate the octal code for the date type in, check for any errors, place the date in the Monitorls date cell, and type the following message:

EVALUATED OCTAL DATE IS 00000

where "00000" is the five digit octal date code representing the bits 15-29 of the word placed in the Monitor's date cell. Control is then returned from SDATE.

The octal date code generated by SDATE has the following structure:



DESCRIPTION

SNCOS is a relocatable AMOS Library subroutine for computing both the sine and cosine of a specified angle (expressed in radians) using the second-order Taylor Series approximation:

$$SIN(X+\Delta X) \approx SIN(X)\ (1-\Delta X^2)+COS(X)\Delta X$$
$$COS(X+\Delta X) \approx COS(X)\ (1-\Delta X^2)-SIN(X)\Delta X$$

as well as the trigonometric relations:

$$SIN(-X) = -SIN(X)$$
$$COS(-X) = COS(X)$$

REQUIREMENTS

SNCOS requires the EAU subsystem; it occupies 166 (octal) locations in memory.

TIMING

SNCOS computes both the sine and the cosine of the given angle in (at most) 178μ sec.

CALLING SEQUENCE

(AR) = $X = \frac{\text{angle value}}{\pi}$ (radians) in bits [15-29]

```
JPSR    SNCOS
  .             Returns to next location
```

RESULTS

(AR) = SIN(πX) in bits [Ø-14], Ø in bits [15-29]
SINE: Ø!H SIN(πX)
COSN: Ø!H COS(πX)

DECRSIPTION

SNCSA is a standard Adage library subroutine written in ADEPT source language. This subroutine uses FCTA to obtain the functions SIN (π X) and COS (π X) for the argument range -1. LT. X. LT. +1, using table lookup and interpolation. SNCSA contains two subroutines, SINA and COSA.

Calling Sequences:

	SINA			COSA	
AR:	X(sign in bit 0, value in bits 1-15)				
L:	JPSR	$SINA		JPSR	$COSA
L+1:		Next instruction			

EXECUTION TIMES

SINA -223 μs, COSA -248 μs

SUBROUTINES REQUIRED

FCTA or FCTE.

HARDWARE REQUIRED

FCTA

Two DAC channels
Two ACE subsystems
One CMP subsystem
One ADC3 or ADC5 subsystem

FCTE

One Extended Arithmetic Unit (EAU)

USAGE

Except for AR, no other "live register" contents are changed by the subroutine.

RESULT

AR SIN (π X) COS (π X) (Sign in bit 0, value in bits 1-29)

GENERAL DESCRIPTION

STALL is a subroutine which runs under the AMOS monitor. The purpose of this routine is to allocate data storage to a program at execution time. This allows a program to have variable length data buffers, or all of available memory, if necessary. If the program terminates and returns to the monitor, its storage area is released and made available to other programs.

There are two entry points to the storage allocation routine, STALL and ALL. STALL is used when a specific number of cells is desired, and ALL is used when all of available memory is desired.

REQUIREMENTS

When using STALL, it is assumed that a proper version of the AMOS Resident Monitor, corresponding to the system configuration being used, is present in core.

STALL

CALLING SEQUENCE

```
JPSR    $STALL    /Call
          N       /(Number of cells requested)
          .       /Error return when not
                       available
          .       /Normal return
```

RESULTS

(AR) = Starting address of N-word block reserved from available storage if possible.

ALL is used when all possible available storage is desired. Four cells are left available to permit entering control statements if program is interrupted.

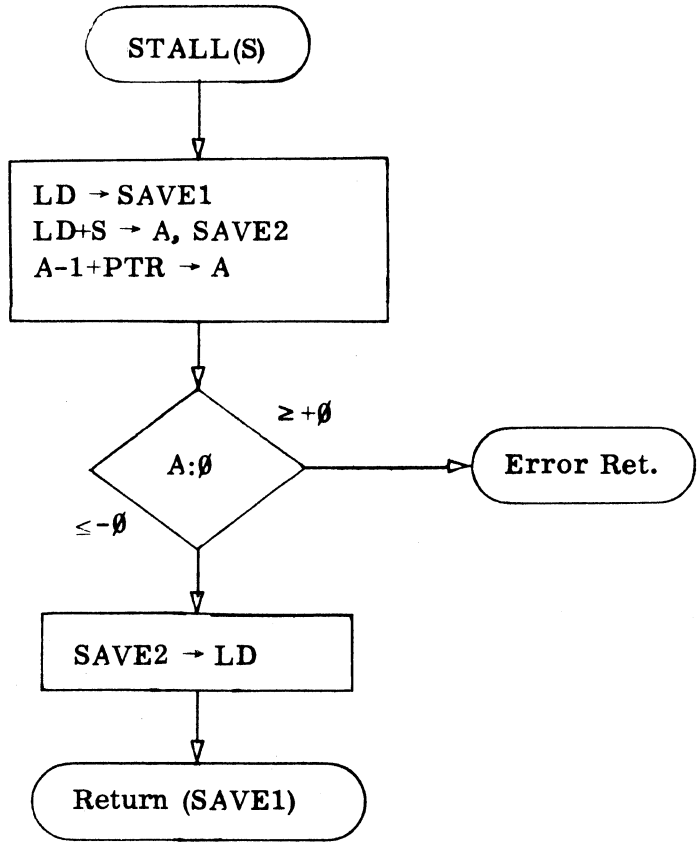
ALL

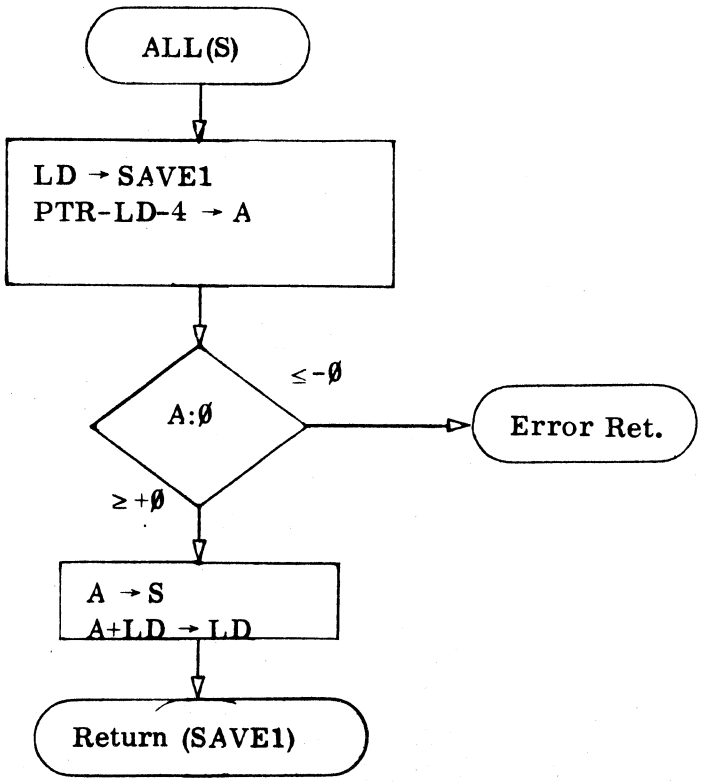
CALLING SEQUENCE

```
JPSR    $ALL    /Call
          N     /(Set to size of block re-
                    served)
          .     /Error return for ≤4 avail-
                    able cells
          .     /Normal return
```

RESULTS

N words of storage reserved (all but 4 of available core).
(AR) = Base address of block reserved when possible.


STALL(S)
LD → SAVE1
LD+S → A, SAVE2
A-1+PTR → A
A:Ø
≥ +Ø
Error Ret.
≤ -Ø
SAVE2 → LD
Return (SAVE1)


ALL(S)
LD → SAVE1
PTR-LD-4 → A
A:Ø
≤ -Ø
Error Ret.
≥ +Ø
A → S
A+LD → LD
Return (SAVE1)

INTRODUCTION

This routine permits AGT display output to be recorded on the GDR1 under operator or programmed control.

GENERAL

WGDR implements the following facilities:

1. Contents of the display screen may be exposed and printed.
2. Successive partial picture displays may be exposed and a final composite of the separate exposures printed.
3. The exposures may be timed by frame counts or by a mechanical timer under operator control.

USE

WGDR can be used by the following online monitor statements or the equivalent calling sequences:

GDRX (BUSY, TIME)!	Expose current display as per TIME
GDRP (BUSY, TIME)!	Expose display as per TIME, then print

Where:

BUSY is the instruction transferred to when the GDR1 is not available.

TIME is the number of frames to expose (or Ø if manual timer run-out is to be used).

When a frame count is used to time exposure, the routine GDRR must be called once per frame. This may be done under DSPLY by an image item:

```
JSR     GDRR
```

or by the clock chain with the monitor statement:

CLOCK (GDRK, GDRR)!

If the user chooses to wait while the GDR1 is busy, the BUSY argument may use the entry point GDRWT.

REQUIREMENTS

WGDR occupies less than 40_{10} cells.

GDRR takes less than 44 μseconds per frame.

WGDR requires no other software, but is designed for easy use with the DSPLY operator under the AMRMX monitor.

CORPORATE OFFICES AND MANUFACTURING PLANT

Adage, Inc.
1079 Commonwealth Avenue
Boston, Massachusetts 02215
Telephone (617) 783-1100
TWX No. 710-330-0141

SALES OFFICES

NORTHERN
1079 Commonwealth Avenue
Boston, Massachusetts 02215
Tel. (617) 783-1100

1300 Route 46
Parsippany, New Jersey 07054
Tel. (201) 335-0900

17500 West Eight Mile Road
Southfield, Michigan 48075
Tel. (313) 358-3393

SOUTHERN
818 Roeder Road
Silver Spring, Maryland 20910
Tel. (301) 589-1221

Braniff Airways Building
Exchange Park
Dallas, Texas 75235
Tel. (214) 358-3161

WESTERN
6151 West Century Boulevard
Los Angeles, California 90045
Tel. (213) 776-6610

680 Beach Street
San Francisco, California 94109
Tel. (415) 771-3577

adage