pdp11

MUMPS-11

Language Reference Manual

Order No. DEC-11-MMLMA-D-D

digital

MUMPS-11

Language Reference Manual

Order No. DEC-11-MMLMA-D-D

digital equipment corporation · maynard. massachusetts

First Printing, November 1972
Revised: October 1973
October 1974
February 1976

The postage prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

5/77-17

CONTENTS

CONTENTS (CONT.)

CONTENTS (CONT.)

TABLES

PREFACE

The MUMPS-11 Language Manual is a reference manual designed to provide the MUMPS-11 programmer with complete and easily accessible information about all aspects of the MUMPS-11 Language. New users should refer to the Introduction to MUMPS-11 Language tutorial manual.

Chapter 1 describes the elements of the language including: the character set, programming modes, program structure, data modes, numbers, strings, literals, constants and variables.

Chapter 2 describes how to form expressions in MUMPS-11 and how they are evaluated.

Chapter 3 describes each MUMPS-11 Command including: its syntax, arguments, meaning and examples of use. This chapter is arranged for quick reference; each command begins on a new page with the command name in large bold type in the upper corner. Chapter 4 describes each MUMPS-11 Function including: its syntax, arguments, meaning and examples of use. This chapter is presented in the same format as Chapter 3.

Appendices provided include: Glossary of Terms, Character Set, Error Messages, Symbol Usage and Conversion Tables.

Associated Documents include:

Getting Started With MUMPS-11
DEC-11-MMGSA-A-D

Introduction to MUMPS-11 Language - Tutorial Manual
DEC-11-MMLTA-C-D

MUMPS-11 Programmer's Reference Card
DEC-11-MMPCA-C-C

MUMPS-11 Programmer's Guide
DEC-11-MMPGA-D-D

MUMPS-11 Operator's Guide
DEC-11-MMOPA-D-D

ACKNOWLEDGMENT

MUMPS-11 is an integrated system comprised of an interactive programming language, a data management facility and a multi-user time sharing executive, developed by Digital Equipment Corporation for the PDP-11. Meditech Corporation contributed to the original development of MUMPS-11. Interpretive Data Systems, Inc., assisted in the Version 4 developments.

The language is a dialect of MUMPS (Massachusetts General Hospital Utility Multi-Programming System) which was developed at the Laboratory of Computer Science at Massachusetts General Hospital and is supported by Grant HS-00240 from the National Center for Health Services Research and Development.

FOREWORD

MUMPS-11 is an interactive single language, multi-user time-sharing system that allows access to a common data base. The capabilities of the system are heavily oriented towards string manipulation using a high level language. The system relieves the user of any concern for programming peripheral devices or for structuring data bases in the traditional sense.

Language processing by the system is in every sense interpretive. Each line of code undergoes identical processing each time it is executed (intermediate code is not generated). The MUMPS application programmer is relieved of all the burdens associated with driving peripheral equipment or the programming of assembly language. He may concentrate his energies to the analysis aspects of his problem. His major problems are concerned with developing proper logical hierarchy for his data base, and developing efficient logic for his data processing requirement.

The MUMPS language is supported by a stand-alone operating system. In addition to implementing the MUMPS language and providing all operating system capabilities, the system affords the user a unique data base structure and access method. Data which is referred to symbolically is automatically stored and linked in a tree structure. The physical allocation of mass storage for the tree structured data base is accomplished by the operating system. The data base thus created is available to other users in the system.

DOCUMENT CONVENTIONS

Symbol — Definition

bve — A Boolean valued expression is an expression which is interpreted as either TRUE or FALSE depending on whether its result is a non-zero or zero value respectively. The standard value for TRUE created by MUMPS is -.01.

gvar — A global variable is a variable which is an element of a global.

lvar — A local variable is a variable which is temporary and resides in the user's partition.

nve — A numeric valued expression is an expression which, when evaluated, yields a numeric result within the range of valid MUMPS-11 numbers.

pnam — A program name consists of any legal identifier, the first character of which may be a % to indicate a library program name.

spn — Any valid Step or Part number.

sve — A string valued expression is an expression which results in a string of ASCII characters which does not exceed maximum string length of 132 characters.

svl — A string variable or literal (a more specific case of sve).

↵ — Universal symbol for line terminator. Line terminators for terminals are either Carriage Return or ALTMODE.

␣ — A single space.

{ } — Fields described within braces are optional.

| | — Vertical bars are used to contain a list of options among which a single choice must be made.

(Continued on next page)

Symbol	Definition
UPPER CASE/ lower case characters	Upper case characters indicate elements of the language which must be used exactly as shown. Lower case characters indicate user supplied elements (sve, nve, etc.) or letters in a command name which may be omitted.
,...	The punctuation characters ,... are used to indicate optional continuation of a command argument list in the form of the last specified argument.
UNDERLINING	All examples showing keyboard (i.e., Direct Mode) input are underlined.

CHAPTER 1

ELEMENTS OF THE LANGUAGE

A MUMPS-11 program is a sequence of symbolic statements which the MUMPS Language Interpreter translates for execution by the operating system.

A command is the basic unit of expression in the MUMPS Language. Commands have one or more elements. The first is a mnemonic which characterizes or symbolizes the action to be performed. Examples: GOTO, SET. The other elements in a command are called arguments, and they specify the objects of the action to be performed.

There are six functional categories of MUMPS commands.

Category	General Function
Assignment Commands	Assign values to symbolic representations.
Control Commands	Govern the sequence in which commands are executed.
Input/Output Commands	Direct the input and output of data to and from the various devices in the hardware environment.
Editing Commands	Permit the examination, modification, and storing of programs.
Debugging Command	Facilitate the creation and maintenance of MUMPS programs.
Timesharing Commands	Permits database timesharing protection.

The format and syntax of MUMPS commands are described in detail in Chapter 3.

1.1 CHARACTER SET

All MUMPS-11 programs are constructed of symbolic characters which form the elements of the language. MUMPS programs use the 64-character graphic subset of ASCII, along with the special control characters listed in Table 1-1. Characters that are used as data may be selected from the entire 128-character ASCII set. Ordinarily, however, those listed in Table 1-1 will appear in language elements only. Appendix B lists the ASCII character set.

Table 1-1
Special MUMPS Control Characters

Name	Function
NUL	Line Terminator (internal)
Carriage RETURN	Line Terminator (external)
ALTMODE	Line Terminator (external)
Line Feed	Line Terminator (external)
Form Feed	Line Terminator (external)
Vertical Tab	Line Terminator (external)
DEL(Rubout)	Delete Character (prior to typing terminator)
CTRL U	Delete Line (prior to typing terminator)
CTRL O	Suppress output to terminal
CTRL C BREAK Key	Sign-on signal by interrupt current operation

1.2 PROGRAMMING MODES

There are two operating modes available to the programmer: Indirect Mode and Direct Mode. Indirect Mode is the mode in which MUMPS executes a stored program. Most MUMPS commands may also be interpreted outside the context of a stored program. In Direct Mode, such commands are executed immediately after entry from a terminal, much like the operation of a desk calculator. Direct Mode is used when creating, modifying, or storing MUMPS programs.

1.3 PROGRAM STRUCTURE

A MUMPS program consists of one or more uniquely numbered lines of commands and arguments, and comments. These lines, called Steps, are terminated by either Carriage RETURN, ALTMODE, or ESCape (symbolized in this manual by a ↵). Each program Step is stored in the user's memory partition for subsequent execution in Indirect Mode. The general format for a step is:

{Step Number␣} {command␣arguments}␣...{;comment}↵

A line of commands not having a Step Number is called a command line and is executed in Direct Mode immediately after it is entered from a terminal. Neither a Step nor a command line may contain more than 132 characters. Specific rules for command syntax are provided in Chapter 3.

1.3.1 Step Numbers

A Step Number is used to identify each line of a MUMPS program. Step Numbers establish the fundamental sequence of program execution. Within a given Part (Section 1.3.2), each line of a program is executed sequentially in ascending Step Number order (assuming, of course, that no Control Commands (Section 3.3) were encountered).

A Step Number is a positive number in the range 0.01 through 327.67. The fractional part of a step number must be non-zero (e.g., 1.00, 198.00, etc., are illegal). Unless explicitly stated in the appropriate command argument, user program execution begins with the Step having the lowest non-zero integer. Step numbers in the range 0.01 through 0.99, though normally used to contain program comments, can contain executable commands; however, control must be explicitly transferred to these steps via arguments to the commands: GOTO, DO, CALL, OVERLAY, and START.

Examples of valid Step Numbers are:

34.87
1.01
.76
08.88

1.3.2 Part Numbers

All Steps having a common integer base form a Part. Parts are used to form program modules, each module specifying a particular procedure within a program. A program may have one Part or many Parts, as the programmer desires. Each Part is a distinct entity with regard to program execution. Execution control is limited to those steps within a Part, and Control Commands such as GOTO, DO, OVERLAY, etc., must be used to effect transfer of control outside of a Part. All Steps in a Part may be collectively referenced by the Part Number alone.

For example:

The series of Steps:

2.01
2.04
2.10
2.87
2.99

can be referred to as Part 2. The command GOTO␣2 would cause all Steps in Part 2 to be executed.

1.4 DATA MODES

MUMPS-11 interprets all data in one of two ways: either as numeric quantities, such as might be used for calculation, or as strings which simply impart their inherent symbolic meaning, such as names and addresses.

1.4.1 Numbers

Numbers in MUMPS are signed, fixed-point, two-place decimal quantities in the range ±21474836.47 [or $-(2^{31}-1)/100 \le n \le (2^{31}-1)/100$].

On input from a terminal or other device, numeric strings used arithmetically which are outside the specified range, are flagged with the following error messages:

MXNUM Integer portion too large
MINIM Fractional portion more than 2 places

Numbers that are stored internally as intermediate results during processing must also conform to the specification, except that decimal fractions are truncated to two places. (No error is created within the system.)

On output, a sign is printed only for negative quantities. Integer quantities are printed without decimal point and trailing zeroes in the fractional part. Decimal fractions are printed with a single leading zero in the form: 0.nn.

Examples of legal numbers are:

```
      .8
     0.25
   100.00
     -.01
   025.
-73256
```

1.4.2 Strings

A string is any contiguous sequence of legal MUMPS characters which is to be considered a single identifiable entity of data. Examples of strings are:

```
HELLO. MY NAME IS:
55 seconds
2,000,345,876,743.4738501
When in the course of human events...
@$#%¢764908!1PoutSFCerhcmAdAtwhS
```

1.5 IDENTIFIERS

An identifier is a string consisting of one to three alphanumeric characters. Identifiers are formed from the characters 0 - 9, the upper case alphabetics A through Z, and the percent (%) character. The first character must be either an alphabetic character or the % character. Remaining characters may be either alphabetic or numeric characters (the % character is legal in the first character position only). Identifiers are used as symbolic names for variables and programs as described later. Identifiers for System Library Programs and Library Globals, however, must use % exclusively as the first character.

Examples of identifiers are:

```
TST     %B6     M
A4R     %11     %X
ZZ0
```

1.6 EXPRESSING DATA VALUES

Program data values may be expressed in several ways in a MUMPS program. The basic units - string literals, numeric constants, and variables - represent single entities of data having either string or numeric values. Literals and constants cannot be altered during a program's execution; variables have whatever values are currently assigned to them. New values may be computed from known values of these data elements using MUMPS commands, functions, and operators which are described in succeeding chapters.

1.6.1 Literals

A literal is used to specify a string of characters which does not change from one execution of a program to the next. A literal may comprise any valid string of characters enclosed in quotation marks (""). A literal may not contain any of the following characters:

Quotation Mark	CTRL C	Line Feed
Carriage RETURN	DEL (Rubout)	Form Feed
ALTMODE	NUL	Verticl Tab
CTRL U		CTRL O

Examples of literals are:

```
"1234.1098+="
"THE ANSWER IS:"
"G$536svfjri'&PPkl;"
```

1.6.2 Constants

A constant is used to express a numeric quantity which does not change from one execution of a program to the next. A constant may consist of any valid MUMPS number (+21474836.47).

Example of constants are:

```
   23.90
     .08
00578.99
  -37.69
```

1.6.3 Variables

A variable is a symbolic representation of a logical storage location. Unlike literals and constants, variables are used to store data which may be altered during a program's operation. Variables may contain either numeric or string data. Numeric data must be within the legal range for MUMPS numbers +21474836.47. String data must conform to the requirements for MUMPS character strings. Variables must be assigned symbolic names which are legitimate identifiers (see Section 1.5).

Three types of variables can be created in MUMPS: Simple Variables, Subscripted Variables, and Global Variables. Variables are created, modified, and deleted using the SET, READ, KILL, and XKILL commands described in Chapter 3.

Examples of variables are:

```
A
X37
SDF
M4Z
%X
```

System Variables are a fourth type of variable in the MUMPS system. These variables, maintained by the operating system, contain general system information for use by all MUMPS programs. System variables are "read only"[1] variables and cannot be altered as can normal variables. These variables use a dollar sign ($) as the first character of their names.

1.6.4 Subscripts and Arrays

A subscript is a numeric valued expression (nve) enclosed in parentheses that is appended to a variable name to uniquely identify a data element residing under that variable name. All the subscripted variables residing under a common name are collectively referred to as an array. An array may consist of either subscripted local variables or subscripted global variables (Sections 1.6.6.2 and 1.6.7.1).

The following is an example of an array with a single level of subscripting:

```
DOG (0.1)
DOG (3.5)
DOG (34.76)
 .
 .
 .
DOG (nnnn.nn)
```

An example of a global array with multiple subscripting levels is:

```
↑ACT (1,1)
↑ACT (1,1,1)
↑ACT (1,2)
↑ACT (1,2,1)
↑ACT (1,2,2)
↑ACT (1,3)
```

The value of a subscript must be a positive number in the range: 0 through 20,971.51.[2] Subscripts may consist of constants, other

---

1. Exception: $Error is "read/write".

2. The $HIGH function (Section 4.2.6) permits an exception to this rule.

variables (which may be subscripted), and expressions (described in Chapter 2). In addition, string variables and literals (svl) may also be used for subscripting. However, the $CREATE function (described in Chapter 4) must be used to convert the string to a unique number.

1.6.5 Sparse Arrays

A sparse array is an array in which only those elements that are explicitly defined or that are required to support the array structure actually exist. Unlike other languages that may require a declaration of the maximum size of an array to preallocate storage space, MUMPS dynamically allocates storage for all arrays only as needed, thus conserving storage space. If a program defines an array which has the following elements,

```
A (4)
A (102)
A (345)
```

only these three elements actually occupy storage space. A program is penalized for occupying too much space when, indeed, there is no space left.

A local array can only have one level of subscripting.

1.6.6 Local Variables

Local variables are variables which reside in the same partition as the commands or Steps which created them. These variables are accessible only to that partition. Local variables are normally used to contain intermediate or transient data which is not to be saved from one execution of a program to the next. There are two types of local variables: simple and subscripted.

1.6.6.1 Simple Variables - A simple variable is a local variable which is not subscripted. Examples of simple variables are:

```
ABC
HAT
R45
X
%D
```

1.6.6.2 Subscripted Variables - A subscripted variable is a local variable which is followed by one subscript and can be used to form a one-dimensional array. Both subscripted variables and local variables may share a common name. Thus, it is possible for both the array ABC as well as the simple variable ABC to exist simultaneously. The programmer should exercise caution when naming variables in this way, since the KILL Command does not distinguish between the two types when no subscript is specified (see Section 3.4.14).

Examples:

```
AGE (AGE)         AGE (ABC(DEF))
AGE (2.45)        ABC (2876)
AGE (A+3.2/T)     ABC (4+B(C*F)/0.89)
```

1.6.7 Global Variables

1.6.7.1 Structure - MUMPS uses one or more disk devices as the primary data storage medium. Access to this storage is gained through the use of global variables (or global nodes). Like local variables, they are created simply by reference in a program or command line. Global variables can be either simple or subscripted. When they are subscripted, the resulting arrays are sparse arrays. Unlike local variables, global variables provide permanent storage and can be accessed by more than one user.[1] Furthermore, there is no limit to the number of levels of subscripting that can be used in forming global arrays permitting the creation of hierarchical data structures that schematically look like inverted trees. Global variables may possess either a string or a numeric data value. In addition, when used in an array, they may also serve as pointers to variables at a lower level in the tree structure.

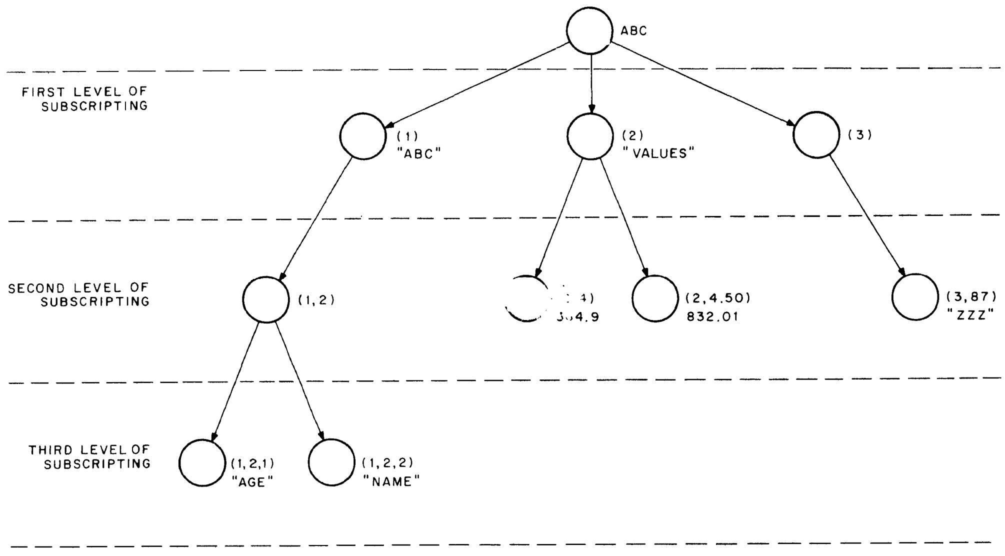
The naming conventions for global variables are the same as for local variables, except that a circumflex (^) or up-arrow (↑) must precede the name. Multiple subscripts are separated from each other by a comma. The following example should clarify this discussion.

In the array ↑ABC, assume the following elements are defined:

Variable	Contents
↑ABC	
↑ABC (1)	"ABC"
↑ABC (1,2,1)	"AGE"
↑ABC (1,2,2)	"NAME"
↑ABC (2)	"VALUES"
↑ABC (2,4)	364.9
↑ABC (2,4.50)	832.01
↑ABC (3,87)	"ZZZ"

A diagram of this array would look like this:

1. As described in MUMPS-11 Programmer's Guide.



(TOTAL NUMBER OF BLOCKS = 5, EXCLUDING THE DIRECTORY BLOCK)

Note that there are some global nodes which exist solely to point to a node which contains data. Such is the case with ↑ABC, ↑ABC (3) and ↑ABC (1,2). These nodes are defined implicitly. Other variables such as ↑ABC (1) and ↑ABC (2) contain both data and a pointer to data at a lower level. Still other variables, those at the lowest level of a branch in the tree, may contain only data. Such is the case with ↑ABC (1,2,1), ↑ABC (1,2,2), ↑ABC (2,4), etc.

A secondary feature of globals is that the top or highest node of a global, in addition to storing the global name and pointers to lower levels, can be used to store auxiliary numeric or string data, like any other variable. Thus:

```
SET␣↑ABC="THIS GLOBAL CONTAINS SALARY DATA"
```

1.6.7.2 Naked Reference - The naked reference is a facility within the language that permits the programmer to avoid excessive disk accesses during program operation. Each time a regular global variable reference is made (e.g., SET␣A=↑ABC (1,2,1)), a physical disk access is performed to bring the disk block containing that global variable into memory. Since global variables at the same level of subscripting reside in the same or a related disk block, a physical access is not always necessary when accessing globals at the same level. Using the naked reference, disk accesses are made only when the subscripting level is changed, or when a "continuation block" at the same level must be read-in to locate the desired variable.

In form, only the up-arrow and subscripts are explicitly stated; the global name is assumed from the last global reference made. The first stated subscript in the naked reference replaces the last subscript stated in the previous reference. The first stated subscript is assumed to be at the same level as that of the previous one. Thus, in the last example, if a reference to ↑ABC (2) has been made and ↑ABC (1) is to be accessed next, only the subscript need be specified as in:

```
SET␣A=↑(1)
```

Similarly, if ↑ABC (1,1,2) is to be accessed next, then:

```
SET A=↑(1,1,2)
```

is all that is required. In this case, however, a disk access is required since the subscripting level has changed.

By far the most common errors that occur in the use of global arrays stem from the incorrect use of the naked reference. The following examples illustrate some of the problems that may be encountered. These examples represent only a few of the many possibilities for producing erroneous results when using the naked reference. It is a powerful tool, but one that must be used cautiously.

Example

```
>IF $D(↑A(I,J))=0 SET ↑(J)=VAL
```

In this case, the user is testing the status of ↑A (I,J) by means of the $DEFINE function. If $DEFINE returns a zero value, that node is undefined. The user then reasons that since $DEFINE has brought him to the desired level, he is safe in using the naked variable. Incorrect!! The user has no way of knowing where the search has ended and thus cannot know the current level. For example, the search may have ended at the first level if no ↑A (I) node was defined. To be safe, the user should use the full reference as in:

```
>IF $D(↑A(I,J))=VAL
```

Of course, if $DEFINE returns a non-zero result, the use of naked variables is perfectly safe.

Example

```
2.01 SET ↑G(I.J.K)=↑(K)+1
```

This command string is legal, but the result will not be what the user desires if his intent is to increment the global specified to the left of the equal sign. The problem lies in the order of evaluation that MUMPS uses for processing a SET command string. The first side of the equal sign (=) to be evaluated is the right side. The naked reference in this case will access the level last reached, which is not necessarily the same as ↑G (I,J,K) on the left side of the equal sign.

The correct form is:

```
2.01 SET ↑(K)=↑G(I,J,K)+1
```

Further information on the structure and use of globals is provided in Introduction to MUMPS-11 Language and MUMPS-11 Programmer's Guide.

1.6.8 System Variables

A number of special "reference only" variables are defined within the system to control the flow of information and to provide system information to MUMPS application programmers and users. These variables, called System Variables, are maintained and updated by the system. They can be examined by various MUMPS commands (TYPE, SET, etc.) but, except for $Error, they cannot be directly altered by SET or READ commands. When referencing System Variables in MUMPS programs only the dollar sign and the first character after it need be used (e.g., TYPE $I). Table 1-2 defines the System Variables.

Table 1-2
MUMPS-11 System Variables

Variable Name	Description
$Address	$A is used with device I/O. When DECtape is the currently ASSIGNed device, $A contains an integer which is the address of the next character to be read or written (range of $A=0-294,911). When magtape is the currently ASSIGNed device $A contains an integer whose bit pattern displays the Magtape Hardware Status Register (drive status register for the TJU16). When the Sequential Disk Processor is the currently assigned device, $A contains either the current disk block address or the error status. When a terminal is the currently assigned device, $A contains the error status. When another processor (CPU) is the currently ASSIGNed device, the low order byte of $A contains a count of unsuccessful I/O transmission (message state only), and the high order byte describes error conditions (message and terminal state). Refer to the MUMPS Programmer's Guide, for bit assignments.
$Byte	When the Sequential Disk Processor is the currently assigned device, $B and $H contain the location (byte address) of the next character to be read or written, according to the formula. ADR = $H*256+$B Where $H = 0 or 1 (page) and $B = n, 0 ≤n≤255 (byte in page)
$Date	$Date contains the date as an integer in the form: (yy*500)+ddd where: yy = Year - 1900 ddd = Number of days since Dec 31 This value is incremented by one when the $T variable is incremented to midnight (86,400 seconds).
$Error	The system sets the contents of $E to negative nve which denotes the type of error incurred. The programmer may optionally set this variable to an spn to control his own error processing. Refer to the MUMPS-11 Programmer's Guide for details.

Table 1-2 (Cont.)
MUMPS-11 System Variables

Variable Name	Description
$Half	See $Byte.
$I/O device	$I contains an integer which is the number of the device which is currently ASSIGNed (i.e., as specified by the last argument of the last ASSIGN command issued). At log-in time, $I contains the I/O device number of that terminal (Principal I/O Device). After a System Error, or UNASSIGN and ASSIGN commands, $I is set to the number of the principal device.
$Job Status	$J contains a number, some of the bits of which specify the current status of programming mode, CTRL C/BREAK recognition, and timed READ overruns. In addition, the ASSIGN and PRINT Commands can be used to alter the bits in $J that control the reception or inhibition of CTRL C or BREAK, the updating of Library Programs and Library Globals, and the writing of memory or disk locations via the VIEW Command. Refer to the MUMPS-11 Programer's Guide for $J bit assignments.
$Location	$L contains the number of the program step currently being executed.
$Random	$R contains an integer in the closed interval 0 to 32767. The value of $R in this interval is effectively random and changes with each reference of $R.
$Storage	$S contains an integer which is the number of free byte (character) locations remaining in the user's partition.
$Time	$T contains an integer which is the number of seconds elapsed since midnight (range = 0 - 86,399). $T is incremented each second.
$Where	The system sets $W to the value of $L when an error occurs and the user had previously SET $E. If the user does not SET $E, $W contains 0.
$X coordinate $Y coordinate	$X and $Y are the x and y coordinates (output only) of the print-head or cursor position on non-mass storage I/O devices, such as the terminals, line printer and paper-tape punch. When a CPU-CPU device operating in message state is the currently assigned device, $X contains the current message number in the range 1 - 15. This number is incremented by one each time a message is transmitted successfully. When the count reaches 15, the next successful transmission resets the count to 0. $Y is not used and does not contain meaningful information.

CHAPTER 2

EXPRESSIONS

The term expression refers to the whole range of value descriptions which can be made in the MUMPS language. An expression is any legal combination of elements (operands) and operators. Expression elements include such basic language elements as literals, constants, simple variables and subscripted variables (including Global Variables and System Variables). Also included in this category are function references (defined in Chapter 4), and subexpressions, which are simply expressions enclosed in parentheses.

The following are examples of expression elements:

123.34	Constant
ABC	Simple Variable
"ABCD"	Literal
MX (5)	Local Subscripted Variable
↑XYZ (2,45.2,D)	Global Variable
$ROOT (PQR)	Function Reference
(A+B (C/D))	Subexpression

The operators in an expression serve to represent various arithmetic, string, and logical operations of the MUMPS language. All operators except minus (-) and Boolean NOT (') are binary operations and therefore require two operands. The minus (-) and Boolean NOT operators are unary operations and require one operator only. Tables 2-1 and 2-2 list the MUMPS-11 expression operators.

The following are examples of MUMPS expressions:

```
A
234.53
A*B- (C/D)
"10 CATS"@"SUP BOTTLES"
"15 DOLLARS"+AMT/NET
TOT=6.21/CD- ($ROOT (G*RT/MQ))
A=C+V&X>S-T!AMT=5
```

Table 2-1
Summary of Numeric Expression Operators

Type	Symbol	Operation
Arithmetic	+	Addition
	-	Subtraction
	*	Multiplication
	/	Division
	#	Modulo
	\	Integer Division
	-	Minus (Unary)
Relational	<	Less Than
	>	Greater Than
	=	Equality
	<= or =<	Less Than or Equal To
	>= or =>	Greater Than or Equal To
	<> or ><	Greater Than or Less Than or Not Equal To
Boolean	&	AND
	!	OR
	'(apostrophe)	NOT (Unary)

Table 2-2
Summary of String Expression Operators

Type	Symbol	Operation
Relational	[	Contains
	]	Follows
	?	Pattern Verification
	=	Equality
Concatenation	@	Concatenation

Intervening spaces between expression elements and operators are not permitted.

The following paragraphs explain the rules that govern the formation of expressions like those above and how MUMPS interprets them.

2.1 RULES FOR FORMING EXPRESSIONS

The following rules apply to the formation of all expressions:

1. Literals, constants, variables, functions, and subexpressions are expressions.
2. If A and B are expressions, then the following are expressions:
   a. A binary operator B
   b. Unary operator A
   c. (A)
3. There are no expressions except those defined by 1. and 2. above.

2.2 DATA MODES

In the MUMPS language there are essentially two types or modes of data, numeric and string. Each of these data types is defined in Chapter 1. To summarize, numeric data are signed fixed-point quantities with two decimal places and are within the range ±21474836.47. String data are simply ASCII character groupings of 132 characters or less.

Internally, MUMPS uses a third format commonly denoted double precision floating point format for storing the results of a $M function calculation. In the description on the following pages of expression evaluation, wherever a numeric to string value conversion is indicated, a floating point number is allowed, and the floating point value would be converted to a string value. However, conversion of a floating point number to a fixed decimal number is not allowed. Thus, although floating point numbers can be used with string operators, a floating point number cannot be used with arithmetic operators outside of a $M function, and care should be taken when using the equality operator with a floating point number.

All expression operators except concatenation produce numeric results. In the case of expressions which use Relational or Boolean operators, evaluation produces either a True or a False result, which is represented in numeric form as either -0.01 (True) or 0 (False).

2.3 RULES FOR EXPRESSION EVALUATION

2.3.1 Order of Evaluation

All MUMPS expressions are evaluated in strict left to right order. There is no precedence among the expression operators except that a unary minus is evaluated before a Boolean NOT when they appear as adjacent operators.

2.3.2 Setting Precedence

Additional precedence is established through the use of parentheses. Parentheses are used to form subexpressions which are evaluated as a single element of the expression in which they appear. Within parentheses, evaluation is performed as described above.

Example:

(Where: R=intermediate result)
in the expression B+C/D*E evaluation is:

$B \rightarrow R_0$
$R_0 + C \rightarrow R_1$
$R_1 / D \rightarrow R_2$
$R_2 * E \rightarrow R_3$

Adding parentheses to the same expression, B+(C/D)*E results in the division being performed prior to the addition as shown below:

$B \rightarrow R_0$
$C/D \rightarrow R_1$
$R_0 + R_1 \rightarrow R_2$
$R_2 * E \rightarrow R_3$

Additional levels of precedence can be achieved by the nesting of subexpressions.

Although there is no logical limit to the depth of nesting, there are physical limits. These are:

1. Physical line length - not more than 132 characters can be used to construct the command line in which the expression resides.

2. Size of partition in which program is running - each level of nesting uses four words of storage during evaluation.

Example:

Where: (R=intermediate result)
in the expression B+((C/D)*E), evaluation is:

$B \rightarrow R_0$
$C/D \rightarrow R_1$
$R_1 * E \rightarrow R_2$
$R_0 + R_2 \rightarrow R_3$

The precedence of evaluation between several subexpressions at the same level in an expression is also strictly from left to right.

Example:

Where: (R=intermediate result)
in the expression C-(X*Y)+W/(E-M), evaluation is:

$C \rightarrow R_0$
$X*Y \rightarrow R_1$
$R_0 - R_1 \rightarrow R_2$
$R_2 + W \rightarrow R_3$
$E-M \rightarrow R_4$
$R_3 / R_4 \rightarrow R_5$

2.3.3 Automatic Data Mode Conversion

During expression evaluation, operands are converted as required from numeric to string data and vice-versa to conform to the data mode requirements of the associated operator. This process does not, however, alter the original mode of stored data (i.e., data in variables, literals, or constants).

Numeric values are converted to the equivalent string representation for string operations. A numeric value of 123.4 would be converted to the characters: 123.4.

String values are converted to numeric values for numeric operations. All leading numeric characters in the string, including +, - and decimal point (.), are changed to the corresponding numeric quantity within the range of MUMPS numbers. The first character that does not conform to the format of a MUMPS number terminates the conversion process; the accumulated value is taken as the result. Any leading zeroes in the resulting numeric value are discarded. Strings that do not contain leading numeric characters produce a 0 result. Thus:

```
12ABC →12                   -1.52A.B+→ -1.52
ABC12 →0                    .52A →0.52
-1.011A →MINIM error        002.5X→ 2.5
```

2.3.4 Data Mode Of Results

The last operator in an expression determines the data mode of the result, either numeric or string.

Examples:

```
A/B@C →String Result
   ↑_________________
               last operator

A@B/C →Numeric Result
   ↑____
     last operator
```

2.3.5 Trailing Operator

An expression may contain a trailing operator to effect a change in the final result of evaluation from numeric to string and vice-versa. A plus (+) sign causes conversion to a numeric value and a commercial at (@) sign, which is also the concatenation operator, causes conversion to a string value.

2.4 OPERATOR DESCRIPTIONS

The following paragraphs specifically define the operations performed by each operator and show the data modes of both the operands and the results. Examples are also included where additional clarification is necessary. The mnemonics listed below are used as a shorthand notation in each description.

Mnemonic	Definition
nv	Numeric Value - this is a numeric quantity which may be either an intermediate or a final result.
sv	String Value - this is a string quantity which may be either an intermediate or a final result.
nvel	Numeric Valued Expression ELement - this is a single, identifiable numeric quantity that may be indicated by a local or global variable, a literal, or result from evaluation of a function or sub-expression.
svel	String-Valued Expression ELement - a single identifiable string quantity that may be indicated by a variable or literal, or result from evaluation of a function or sub-expression.

2.4.1 Arithmetic Operators

The arithmetic operators permit arithmetic computations to be performed. The symbols are defined as follows.

Legend:

Symbol	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
#	Modulo - When both arguments of this operator are positive, the resulting value is the remainder after integer division. More formally, A#B is defined to be A-(A\B*B) + (absolute value of B if A<0) The result is insensitive to the sign of B, and it is always positive.
\	Integer Division - The result is the same as normal division except that the decimal portion is truncated.

Binary Forms:

nv	+ - * / # \	nvel → nv	A direct evaluation of the nvel is made. The operation is performed and a numeric result produced.

nv	+ - * / # \	svel → nv	The svel is evaluated and the result is converted to a numeric value. The operation is performed and a numeric result produced.
sv	+ - * # \	nvel → nv	The sv is converted to a numeric value. The operation is performed, and a numeric result produced.
sv	+ - * / # \	svel → nv	The sv is converted to a numeric value. The svel is evaluated and the result converted to a numeric value. The operation is performed, and a numeric result produced.

Unary Forms:

When the minus (-) is used as a unary operator to indicate negation, it may prefix any element in an expression. Thus:

-nvel → nv The resulting nv has a different sign but the same absolute value.

-svel → nv The svel is converted to a numeric value then treated as above.

Examples:

Where: A=3; B=5; C=12; D=6

1. A+B- (C/D) → 6

2. A+B-C/D → -0.66

3. 3#5 → 3

4. -3#5 → 2

5. 3.4\1.4 → 2

6. "3214 MAIN ST"-802 → 2412

   Analysis:

   "3214 MAIN ST" → 3214
   3214-802 → 2412

7. 23*"CAT" → 0 Since there are no leading numeric characters in "CAT", it is evaluated as numeric 0.

Analysis:

"CAT" → 0
23*0 → 0

8. "1234ABCDE6789"/"0002POIU" →617

Analysis:

"1234ABCDE6789" →1234
"0002POIU" → 2
1234/2 →617

2.4.2 Relational Arithmetic Operators

The relational arithmetic operators permit the comparison of numeric or string quantities in an arithmetic manner. The results of expressions using these operators are either -0.01 to represent a True relation or 0, to represent a False relation.

Legend:

Symbol	Operator
<	'Less Than' comparison
>	'Greater Than' comparison
<= or =<	'Less Than or Equal To' comparison
>= or =>	'Greater Than or Equal To' comparison
< > or ><	'Greater Than or Less Than' or 'Not Equal To' comparison

Forms:

nv	< > >= <= < > ><	nvel →True or False	A direct evaluation of the nvel is made and the operation is performed. A numeric result is produced. -0.01 (True) or 0 (False).
nv	< > >= < > ><	svel →True or False	The svel is converted to a numeric value and the operation is performed. A numeric result is produced. -0.01 (True) or 0 (False).
sv	< > <= >= < > ><	nvel →True or False	The sv is converted to a numeric value and the operation is performed. A numeric result is produced. -0.01 (True) or 0 (False).

sv	< > >= >= < > ><	svel → True or False	The sv and svel are converted to numeric values and the operation is performed. A numeric result is produced -0.01 (True) or 0 (False).

Examples:

Where: A=3; B=5; C=12; D=6; X=10

1. A+B>C/D →0 (False)

   Analysis:

   A+B → 8
   8>C → 0
   0/D → 0

2. A+B>(C/D) →0.01 (True)

   Analysis:

   A+B → 8
   C/D →2
   8>2 →0.01 (True)

3. "3214 MAIN ST" <=802 → 0 (False)

   Analysis:

   "3214 MAIN ST" → 3214
   3214<=802 →0 (False)

4. 25*"CAT"< >6 → -0.01 (True)

   Analysis:

   "CAT" →0
   25*0 →0
   0< >6 → -0.01 (True)

5. "23TSV"+"XYZ"<"78.04FARGH"+3 → 2.99

   Analysis:

   "23TSV" →2.3
   "XYZ" →0
   23+0 →23
   "78.04FARGH" →78.04
   23<78.04→ -0.01 (TRUE)
   -0.01+3 →2.99

6. X>A>B → 0 (False)

   Analysis:

   10>3→ -0.01 (True)
   -0.01>5 → 0 (False)

2.4.3 Relational Equality Operator (Arithmetic or String)

Relational Equality operations are signified by the use of the equal sign (=), and can be considered either arithmetic or string in nature, depending upon the type of operands used. The results of expressions using this operator are either -0.01 (True equivalence) or 0 (False equivalence).

Care should be taken when using a floating point number (created by a $M function) with the equality operator. In such cases, the following rules govern.

1. If the 1st argument is a floating point number, it is interpreted as a string value.
2. If the 2nd argument is a floating point number and the 1st argument value is a fixed decimal numeric value, a MIXED error is generated.
3. If the 2nd argument is a floating point number and the 1st argument's value is a string value, a string conversion of the 2nd argument occurs.

The definition of equality when used with strings or fixed decimal numbers is given below.

Legend:

Symbol	Operation
=	Numeric or String Equivalence

Forms:

nv = nvel → True or False	A direct evaluation of the nvel is made and the operation is performed. A numeric result is produced, -0.01 (True) or 0 (False).
nv = svel → True or False	The svel is converted to a numeric value and a numeric comparison is made. A numeric result is produced -0.01 (True) or 0 (False).
sv = nvel → True or False	The sv is converted to a numeric value and a numeric comparison is performed. A numeric result is produced -0.01 (True) or 0 (False).
sv = svel → True or False	The svel is compared with the sv on a character-by-character basis. A numeric result is produced -0.01 (True) or 0 (False).

NOTE

The > or < operator cannot be used to make non-numeric string comparisons.

Examples:

Where: A=3; B=5; C=12; D=6

1. A+B=C/D →0 (False)

   Analysis:

   ```
   A+B →8
   8=12 →0 (False)
   0/6 →0
   ```

2. A+B=(C/D) →0 (False)

   Analysis:

   ```
   A+B →8
   C/D →2
   8=2 →0 (False)
   ```

3. "5DOGS"*5=25→ -0.01 (True)

   Analysis:

   ```
   "5DOGS" →5
   5*5→ 25
   25=25→ -0.01 (True)
   ```

4. -3+2="ABCD234" →0 (False)

   Analysis:

   ```
   -3+2→ -1
   "ABCD234" →0
   -1=0 →0 (False)
   ```

5. "JACOBS"="JACOB" →0 (False)

   Analysis:

   The string JACOBS is not equal to the string JACOB.

2.4.4 Relational String Operators

The relational string operators provide facilities for determining the characteristics of string data. Results of expressions using these operators are either -0.01 to represent a True relation or 0 to represent a False relation.

Symbol	Operation
[	String Contains - The string specified by left operand is examined for the occurrence of the string specified by the right operand. If a match is found the result is True (-0.01); otherwise the result is False (0).
]	String Follows - The string specified by the left operand is compared character-for-character with the string specified by the right operand to establish relative position according to the MUMPS collating sequence. (Refer to

Symbol	Operation
	Appendix B.) If the string specified by the left operand "follows" that specified by the right operand, the result is True (-0.01); otherwise the result is False (0).
?	Pattern Verification - The string specified by the left operand is examined for the occurrence of the character patterns specified by the Pattern Specification Codes (psc) contained in the right operand. If a matching condition exists the result is True (-0.01), otherwise the result is False (0). Pattern Specification Codes may be preceded by a single decimal integer (n) in the range 0 - 9 to specify the number of occurrences of a particular character type. If 0 is specified, the associated character type is ignored. If no number is specified an indefinite number of characters of the specified type are accepted.

Code	Meaning
A	Verify upper case alphabetics
B	Verify lower case alphabetics
C	Verify upper and lower case alphabetics
D	Verify numerics
M	Verify numerics and upper case alphabetics
N	Verify numerics and lower case alphabetics
O	Verify numerics and upper and lower case alphabetics
P	Verify punctuation
Q	Verify punctuation and upper case alphabetics
R	Verify punctuation and lower case alphabetics
S	Verify punctuation and upper and lower case alphabetics
T	Verify numerics and punctuation
U	Verify numerics, punctuation and upper case alphabetics
V	Verify numerics, punctuation and lower case alphabetics
W	Verify any character

All characters which are not strictly alphabetic or numeric are considered to be punctuation. Literals may also be used to verify the occurrence of specific characters in a string.

Forms:

nv	[ ]	nvel → True or False	The nv is converted into its string equivalent. The nvel is evaluated and the result converted into its string equivalent. The comparison is made, and a numeric result produced: -0.01 (True) or 0 (False).
nv	[ ]	svel → True or False	The nv is converted into its string equivalent, the comparison is made and a numeric result produced: -0.01 (True) or 0 (False).
sv	[ ]	nvel → True or False	The nvel is evaluated and the result is converted to its string equivalent, the comparison made, and a numeric result produced: -0.01 (True) or 0 (False).

sv	[or]	svel → True or False	The comparison is made and a numeric result produced: -0.01 (True) or 0 (False).

CAUTION

If the svel is not a variable or literal (i.e., it results from the evaluation of a function or subexpression), there is a possibility that the internal string accumulator used by the expression evaluator, may overflow thus terminating program operations with a MXSTR error.

sv?n psc_1 n psc_2 ... → True or False	The string is examined in accordance with the pattern specification code(s) to the right of the operator and a numeric result is produced: -0.01 (True) or 0 (False).
nv?n psc_1 n psc_2 ... → True or False	The nv is converted to its string equivalent, the string is examined in accordance with the pattern specification code(s), and a numeric result is produced: 0.01 (True) or 0 (False).

Examples:

1. Where: A = 3; B = 500

   A+B[50 → True

   Analysis:

```
A+B → 503
503 → 503                  convert numeric to string
 50 → 50                   convert numeric to string
503[50 → -0.01 (True)      string 503 contains 50
```

2. Where: A = "ADAMS   JQ"
   B = "ADAMS   JA"
   A]B → -0.01 (True)

   Analysis:

   The left string is compared on a character for character basis to the right string. The result is True since "Q" follows "A" in the collating sequence. (Refer to Appendix B.)

   Where: A = "JONES   J"
   B = "JONES   J"
   A]B → False

   Analysis:

   The comparison is made as above but a False result is produced since the strings are identical. To absolutely establish equality, a relational equality operation must be performed. Thus A=B → True.

3. Where: DAT = "04/23/72"
   DAT?2D"/"2D"/"2D → -0.01 (True)

   Analysis:

   The string contained in DAT is examined for a pattern consisting of 2 numeric characters followed by a slash (/) followed by 2 numeric characters followed by another slash (/) followed by 2 numeric characters. Note the use of literals to verify specific characters.

2.4.5 String Concatenation Operator

The string concatenation operator (@) permits the joining together (concatenation) of expression elements to form strings. The string vaue represented by the right operand is appended to the string value represented by the left operand.

Forms:

nv@nvel → sv — The nv is converted to its string equivalent. The nevel is evaluated and the result is converted to its string equivalent. The operation is performed and a string result produced.

nv@svel → sv — The nv is converted to its string equivalent and the svel is evaluated. The operation is performed and a string result produced.

sv@nvel → sv — The nvel is evaluated and the result is converted to its string equivalent. The operation is performed and a string result produced.

sv@svel → sv — The svel is evaluated, the operation performed, and a string result produced.

Examples:

1. "CAT"@"SUP" → CATSUP

2. Where: MO = 4
   DA = 22
   YR = 72

   MO@"/"@DA@"/"YR → 4/22/72

3. Where: B = 6
   C = 2

   "THE RESULT IS "@(B>C) → THE RESULT IS -0.01

2.4.6 Boolean Operators

The operators described below permit the construction of Boolean expressions using the AND (conjunction), OR (disjunction) and NOT (logical complement).

Operator	Operation
&	AND - forms the Boolean AND (logical product) of the operands
!	OR - forms the Boolean inclusive OR (logical sum) of the operands
'(apostrophe)	NOT - forms the logical complement of the operand (unary operation)

Boolean Truth Table

Where: True = -0.01
False = 0
N = any number including True and False
M = any nonzero number

AND	OR	NOT
True & True=True	True! True=True	'True=False
True & False=False	True! False=True	'False=True
False & True=False	False! True=True	'M=False
False & False=False	False! False=False	
N &True =N	N! True =True	
True & N =N	True! N =True	
N & False =False	N! False =N	
False & N =False	False! N =N	

The AND and OR operations are performed on a bit-by-bit basis on two 32-bit quantities which allows either simple evaluation of True and False quantities or complex masking operations (by knowledgeable system programmers). The NOT operator is a logical rather than a bit-by-bit complement. As shown above, the complement of a False (0) value is True (-0.01) but, by convention, the complement of any non-zero quantity, including True, is always False (0). A true bit-by-bit complement of a value can be accomplished using unary minus, thus:

-N-0.01

All operands used with Boolean operators are assumed to be numeric. Operands which are string values are evaluated and converted to a numeric value in accordance with the rules for mode conversion (see Section 2.3.3). The results of Boolean operations are always numeric.

Binary Forms:

nv	& / !	nvel → nv	The operation is performed and a numeric result produced.
nv	& / !	svel → nv	The svel is evaluated and the result converted to a numeric value. The operation is performed and a numeric result produced.
sv	& / !	nvel → nv	The sv is converted to a numeric value; the operation performed, and a numeric result produced.
sv	& / !	svel → nv	The sv is converted to a numeric value. The svel is evaluated and the result is converted to a numeric value. The operation is performed and a numeric result produced.

Unary Forms:

NOT is a unary operator and may prefix any element in an expression.

'nvel →nv	The nvel is evaluated and, if the result is either True or False, its sense is reversed. Any other value of the nvel produces False.
'svel →nv	The svel is evaluated and its result is converted to a numeric value. If the value is either True or False, its sense is reversed. Any other value produces False.

Examples:

Where: A = 4; B = 2; X = 8; C = 3

1. (A>B)&(A<X) → -0.01 (True)

   Analysis:

   4>2 → -0.01 (True)
   4<8 → -0.01 (True)
   True & True →-0.01 (True)

2. '(A+C)!('X-'C)!("12CATS">"10") → -0.01 (True)

   Analysis:

   4+3 → 7
   '7 → 0
   '8 → 0
   '3 → 0
   0-0 → 0
   0!0 →False
   "12CATS" →12
   12>10 →-0.01 (True)
   0!-0.01→-0.01 (True)

2.4.7 Trailing Data Mode Operators

A trailing operator may be appended to the last element in any expression to ensure the data mode of the final results of the expression.

Operator	Operator
@	Convert the data mode of the expression result to a string value.
+	Convert the data mode of the expression result to a numeric value.

Forms:

sv+ →nv	The sv is converted to its numeric equivalent.
nv+ →nv	The operation is ignored since the operand is already numeric.
nv@ →sv	The nv is converted to its string equivalent.

sv@ → sv — The operation is ignored since the operand is already a string.

Examples:

Where: A = 100, B = 2, C = "00"

1. A+B@ → "102"
2. A>B@ → "-0.01"
3. A/B@C+ → 5000

CHAPTER 3

COMMANDS

INTRODUCTION

A command is the principal algorithmic component of the MUMPS language and consists of one or more elements the first of which is a mnemonic that characterizes the action or procedure to be performed.

Examples:

GOTO,READ,SET,OVERLAY

Any remaining elements in a command are arguments and their delimiters, and special symbols. Arguments specify a logical entity, such as a variable or expression to or upon which the action of the command is directed.

3.1 RULES FOR COMMAND SYNTAX

1. Commands which are to be executed immediately (Direct Mode) do not use Step Numbers. The first character of the command is the first character on the line following the system's right angle bracket (>) prompting symbol.

2. Commands which are to be executed as part of a stored program (Indirect Mode) are preceded by a Step Number. A command is separated from a Step Number by a single space.

3. Each command may be abbreviated to its first letter. Furthermore, to do so saves partition space since only the first character is necessary but all succeeding characters up to the next space (␣) character are stored. Care should be used when abbreviating commands to avoid confusing certain commands which are executable only in Direct Mode with others which can only be executed in Indirect Mode. For example:

   E␣2.5

   In Direct Mode it means: ERASE step 2.5. In Indirect Mode it is read as: ELSE, and produces a syntax error, since 2.5 is not a valid command.

4. A command is separated from its argument or argument list by a single space.

5. Multiple arguments to a command are separated from each other by commas.

6. Multiple commands on a line must be separated from each other by a single space.

7. Certain commands permit the optional use of an argument or argument list. Note that the ELSE command is an exception, only one intervening space is allowed. If such a command is not the last command on a line, and is to be used with no argument list, it must be separated from the next command by two spaces.

8. Program comments may be appended to say command line. When used, they must be preceded by a semicolon (;). The semicolon may be separated from the preceding command argument list or Step Number by an optional space.

9. The indirection syntax operator, symbolized by either underscore (__) or back arrow (←), provides dynamic command argument definition. In form, a command's argument is replaced by the symbol __ or ← immediately followed by a variable name. The variable must contain a string that is a syntactically correct argument or argument list. The argument(s) can be followed by one or more commands and their arguments (excluding the QUIT command). During execution, the contents of the variable are interpreted accordingly.[1]

   Example: where: ARG = "15+3/6"

   1.20␣TYPE␣←ARG - The contents of ARG are evaluated as the argument and the result is 3.

10. An optional Boolean Valued Expression preceded by a colon (:bve) can be used to specify conditional execution of certain commands and command arguments.

    Examples: 2.03␣GOTO␣3:A>B - control is transferred to Part 3 if the contents of 'A' is greater than the contents of 'B'.

    10.21␣WRITE:A=B␣2 - If A=B, all the Steps in Part 2 are written out to the currently assigned I/O device.

11. The colon can also be used to specify alternate forms of certain commands.[2]

    Example: 6.30␣READ␣X:5 - is a 'timed' READ.

---

1. Refer to Section 3.3.13 for further information on the indirection syntax.

2. Refer either to Table 3-1 or to the specific command descriptions (Sections 3.3.1 through 3.3.27) to determine the applicability of this feature.

3.2 FORMAT CONTROL

The following special symbols are used with the PRINT, READ, and TYPE commands to effect format control:

Symbol	Description
#	Number sign is used as a format control character to initiate a Page Feed or a FORM Feed on an output device.
!	Exclamation point is used as a format control character to initiate a Carriage-RETURN-LINE-FEED (CRLF) sequence on an output device.
?	Question mark specifies horizontal tabulation (output only) on devices such as the terminals, line printer and paper-tape punch. The ? symbol is followed by a nve to specify the number of spaces from the absolute left margin. Form: ?nve.

When these symbols appear consecutively on a line, the intervening commas, normally required to separate command arguments, can be omitted.

3.3 DESCRIPTIONS OF MUMPS COMMANDS

MUMPS commands fall into six functional groups as shown in Table 3-1 below. The following paragraphs describe each command and its argument. Examples are provided for clarification. The commands are presented in alphabetical order for easy reference.

Table 3-1
Functional Relationship of MUMPS Commands

Assignment Commands	Assign and deassign values to symbolic representations.
Set	{:bve} ␣ var=expression,...
Kill	{:bve} { | ␣ variable,... / ␣ ␣ | }
Xkill	{:bve} ␣ lvar,...
Control Commands	Govern the sequence in which commands Steps, Parts, and related programs are executed.
Goto	{:bve} ␣ spn {:bve} ,...
Do	{:bve} ␣ | spn / svl {:bve} | ,...
If	| bve,... / ␣ ␣ | next command
For	␣ ivar= {nve_1 {:nve_2 :nve_3} ,...} {nve_1 :nve_2}␣ | WHILE / UNTIL | ␣ bve} ␣ next command
Else	␣ next command
Call	{:bve} ␣ pnam {:spn} ,...
Overlay	{:bve} ␣ pnam {:spn}
Start	{:bve} ␣ pnam {(nve)} {:spn} ,...
Quit	{:bve}
Hang	{:bve} ␣ nve {:bve} ,...
Halt	{:bve} ␣ ␣

Table 3-1 (Cont.)
Functional Relationship of MUMPS Commands

Input/Output Commands	Direct the input and output of data to and from the various devices in the hardware environment.
Type {:bve}	{ ␣ | {expression format variable ␣} ,... | }
Read {:bve} ␣	| lvar {:nve} literal format | ,...
Print {:bve} ␣	| nve literal format | ,...
Write {:bve}	{ ␣ | spn_1 ␣ {:spn_2} ,... | }
Assign {:bve} ␣	| nve_1 0:bve { | :sve{:nve_3{:nve_4}} :nve_2{:nve_3} | } | ,...
Unassign {:bve} ␣ nve,...	
Editing Commands	Permit the examination, modification, and storage of MUMPS programs.
Modify {:bve} ␣	| spn: sve spn: sve_1 sve_2 |
Erase {:bve}	{ | ␣spn ␣ ␣ {:spn} ,... | }
Load {:bve}	{ | ␣pnam ␣ ␣ | }
File {:bve}	{ | ␣pname ␣ ␣ ,... | }

Table 3-1 (Cont.)
Functional Relationship of MUMPS Commands

Debugging Commands	Facilitate the creation and maintenance of MUMPS programs.
Break { ␣ :bve / ␣ ␣ }	
Go {:bve}	
System I/O Command	Permits privileged modification of core and disk memory by MUMPS System Programs.
View {:bve} ␣ nve_1 {$:nve_2$} ,...	
Timesharing Commands	Permits database protection through a hierarchical interlock by applications program convention.
Lock {:bve} ␣ | gvar {:nve} / (gvar ,gvar,...) {:nve} | ,...	
Unlock {:bve} ␣ ␣	

ASSIGN

3.3.1 ASSIGN Command

Mode: Direct or Indirect

Syntax:

```
                               | nve1   ( | :sve {:nve3 {:nve4}} | ) |
Assign {:bve}␣ |        (           | :nve2 {:nve3}        |    |   ,...
                               | 0:bve  ) |                      | ) |
```

Description:

This command permits one or more I/O devices (DECtape, Magtape, Paper Tape, Line Printer, Sequential Disk Processor and terminals) to be reserved for the exclusive use of a program (Indirect Mode) or programmer (Direct Mode). The last device specified in the argument list is made 'current' by setting the partition's $I System Variable to that number (refer to Section 1.6.8). This means that subsequent I/O commands, such as READ, TYPE, WRITE, and PRINT, are directed to the 'current' device. Other devices specified in the argument list though not 'current' are 'owned' and are not available for use by other programs in different partitions.

Each device is assigned in the sequence specified by the argument list. If all assignments are successful, the next command on the line is executed. If an assignment is not successful (i.e., a device is "owned" by another job), this job is suspended and the remainder of the argument list is not processed. When the device becomes available, job suspension terminates and argument list processing is resumed.

NOTE

> Devices not currently "owned" should be assigned in numerically ascending order to avoid conflicts with other jobs competing for the same devices. Failure to follow this procedure can cause two or more of the competing jobs to hang.

Each argument (nve_1) must specify a legal device number (refer to the MUMPS-11 Programmer's Guide). Device 0 always refers to the Principal I/O Device (i.e., the terminal that initiated program operation). The use of illegal device number or numbers for nonexistent devices causes a NODEV error and immediate program termination.

The use of the optional arguments "nve_2", "nve_3", and "nve_4" depends on the device specified by nve_1. If nve_1 is a terminal (device 1-19 and 64-111), then nve_2 may be used to control the right margin of output to the terminal. It specifies the maximum number of characters to output on any given line. Margin control remains in effect until either an UNASSIGN command or another ASSIGN command (nve_2 = 0) is issued. If the user wishes to affect margin control for the Principal I/O Device, nve_1 must be the actual device number of that terminal - not device 0.

If nve_1 is a CPU-CPU device, the optional ":bve" may be used to change the state of the CPU driver. If the bve is True (non-zero result), the CPU driver enters the message state. If the bve is False (zero result), the CPU driver enters the terminal state. The default state is terminal state. The device remains in its current state until another ASSIGN directing a change of state is executed. When the device is UNASSIGNed, it is reset to the terminal state.

If nve_1 is a DECtape device, then nve_2 may be used to specify the location (byte) of the next character to be read or written. The specified value is entered in the $A System Variable. Legal values for :nve_2 for DECtape are in the range 0-294,911 (integer). When reading or writing sequential records, nve_2 need not be specified in subsequent ASSIGN commands since the system keeps track of the current tape position automatically.

If nve_1 is a magtape device, then sve may be used to modify the tape format for subsequent Magtape I/O. Each character in sve represent a switch according to the following table.

Switch Character	Effect
A	ASCII character set
D	DOS-11 compatible format
E	EBCDIC character set
F	Fixed length logical record
L	Standard Labeling (ANSI or IBM)
S	Stream data format
U	Unlabeled
V	Variable length logical records (ANSII "D", IBM "V" format)
digit	Density bits specification

Nve_3 may be used to specify a fixed length logical record size in bytes; a 0 in nve_3 is used for stream or variable length records. Nve_3 is required if the fixed length record option is being used. Nve_4 may be used to specify the physical block size in bytes of data blocks. Nve_4 can range from 140 through 512. When the Assign command is used to establish ownership of a magtape drive, a default tape format is assumed which can be immediately modified for the duration of ownership by the optional Assign arguments. Optional arguments of subsequent Assigns which merely route I/O to the drive are ignored.

Further information regarding the effect of the Assign command on Magtape operations can be found in the Magtape section of the MUMPS-11 Programmer's Guide.

If nve_1 is a Sequential Disk Processor, then nve_2 may be used to specify the location (byte address) of the next character to be read or written. Legal values for nve_2 for the Sequential Disk Processor are in the range 0-511. The specified value is entered in the $B and $H System Variables according to the formula:

ADR=$H*256+$B
where $H=0 or 1 (page)
and $B=n, $0 \leq n \leq 255$ (byte in page)

If nve_1 is a Sequential Disk Processor and nve_2 is specified, the optional argument 'nve_3' may be used to specify the disk block address. The value is entered in the $A System Variable according to the formula.

nve_3=TYP*2,097,152+(UNT*262,144)+BLK

where TYP (device type)=
0 for RK11
1 for RF11
2 for RP11
3 for RJP04

UNT(unit number)=m, $0 \leq m \leq 7$

BLK (block address on unit)=n, $0 \leq n \leq$
4799 for RK11
1023 for RF11
79,000 for RP11(RP03)
39,999 for RP11(RP02)
170,543 for RJP04

As with the DECtape operation, the system also keeps track of the position of the current record on the SDP device after the initial ASSIGN command. Thereafter, nve_2 and nve_3 need not be specified in subsequent ASSIGNs when performing sequential I/O.

The "0:bve" argument permits enabling and disabling of the CTRL C and BREAK control characters originating from the Principal I/O Device and from all currently "owned" terminals. When a user logs into the system using the Programmer Access Code (PAC), CTRL C and BREAK are enabled. When the PAC is not used, CTRL C and BREAK are disabled. If the :bve is True (i.e., non-zero result), CTRL C/BREAK are enabled. If the :bve is False (i.e., zero result), CTRL C and BREAK are disabled. The $J System Variable contains a status bit that is set whenever a CTRL C or BREAK is issued. This bit is reset whenever an A␣0:bve command is executed.

In Direct Mode, an ASSIGN makes the specified device "current" only for commands executed on the remainder of the line. When all commands are executed, $I is automatically reassigned to the Principal Device. The assigned device is, however, still "owned".

Examples:

1. 1.2 A 12	reserves device 12 and sets $I to 12 to make the device 'current'.
2. 3.05 A 2,3,55:5000	reserved devices 2, 3, and 55, (which is a DECtape), makes 55 'current' and sets $A to 5000 to position the DECtape for the next I/O command.
3. >A 0:1=2	disables terminal interrupts of currently owned terminals and makes the principal I/O device 'current'.
4. >F I=1:1:100 A 2 R TMP A 3 T TMP	reads 100 lines, one at a time, from device #2, and outputs them to device #3.
5. >S A="TEST" A 3 >T A TEST >	outputs the string 'TEST' to the principal I/O device, since all commands did not reside on the same line. Device 3 is still owned, however.
6. >A 46,58:512 W	reserves device 46 to View core, and reserves DECtape unit 3, positioning the tape at the head of the second tape block (512 bytes per block); makes the DECtape device "current" and writes the contents of the program buffer onto the DECtape.
7. 7.65 A 59:210:2400 T X,!	assigns the first (of four) Sequential Disk Processor and types the contents of X starting at byte 210 of block 2400 on RK11 unit 0, followed by a carriage-return, line feed.
8. 4.3 A 4:0 T "HELLO",! A 4:1 R REP:20	assume device 4 is a CPU device; send the text "HELLO" to the CPU in terminal state, switch to message state and wait 20 seconds for a reply.
9. 3.7 A 47:"AVL"	reserves Magtape drive 0 and specifies the ANSII standard "D" format.

10. 2.65 A 47:"EUF" :80:240

reserves Magtape drive 0 and specifies unlabeled EBCDIC 80-character fixed length records and 240 byte blocks (3 records per block).

BREAK

3.3.2 BREAK Command

Mode: Indirect

Syntax: Break {␣:bve / ␣␣}

Description:

This command is used to stop a program at a specified point to assist debugging during program development. The prime purpose of the command is to permit the examination of program variables at various states in a program's operation.

When performed, the command interrupts execution of the program, reassigns $I to 0 (Principal I/O Device), and prints out a question mark (?) followed by the word "BREAK" and the Step number containing the BREAK command. Control returns to the user in Direct Mode. At this point the program is still considered to be running, but in a suspended state. Any attempt to modify the program will cause an error. If modification is desired, type a CTRL C, make the modification, and restart the program from the beginning.

The GO command (Section 3.3.9) is used to continue program execution after a BREAK. However, the occurrence of any error, or typing CTRL C will cause the BREAK to be "lost", thereby preventing execution of a subsequent GO.

Examples:

1.	2.51 B:X<10	a BREAK will occur only if X is less than 10
2.	6.1 B	an unconditional BREAK will occur
3.	3.15 S A=B A 2 B␣;␣T B	the remainder of the line is not executed after the BREAK until a subsequent GO is issued.

CALL

3.3.3 CALL Command

Mode: Direct or Indirect

Syntax: Call {:bve} ␣ pnam {:spn} ,...

Description:

The CALL command initiates execution of a program residing in either the user's Program Directory or the System Library. Program execution begins where specified or at the lowest non-zero Part.

When the program that was called is finished executing (i.e., no more Steps to do or a QUIT statement encountered), the original calling program is read back into the partition and re-entered at the point immediately following the invoking CALL command.

NOTE

> The calling program must be FILEd (Section 3.3.7) or MUMPS will be unable to return to it after the CALLed program has terminated. This results in a NOPGM error message.

When a program is called, all the local variables in the partition are preserved and available to it. The local variables remain unchanged except for changes which the called program may make. Execution of the program begins at the first non-zero Part unless a Part or Step number was specified by the optional ':spn'. The CALL command takes an average of two disk access times, one to bring in the called program and one to return to the calling program. CALL effectively increases the size of a program that can run in a given partition, but trades on execution time to do it.

Each program named in the CALL command is loaded and executed one at a time, in the order of appearance in the CALL command.

Example:

This command line points out the features of the CALL command:

```
6.35 CALL A,SAM:2.50,ABE:5,%T
```

1. Calls program A and executes it.
2. Calls program SAM and executes it, starting at Step 2.50.
3. Calls program ABE and executes it, starting at Part 5.
4. Calls the Time Subroutine %T and executes it.

DO

3.3.4 DO Command

Mode: Direct or Indirect

Syntax: Do {:bve} ␣ spn {:bve}

Description:

The DO command initiates execution of the specified argument. When in Direct Mode, the DO command provides the only means for initiating execution of command strings currently held within the user's partition.

An argument may be:

1. a Step number
2. a Part number
3. an svl that contains either 1. or 2. above.

If control returns to the next DO command, the argument is in the mode in which the command was issued. In particular, if there is no other DO command argument and the DO command was issued in Direct Mode, the next command on the line is executed. Control ultimately returns to Direct Mode. If there is no other DO command argument, and the DO command was issued from Indirect Mode, control returns to the command following the DO. If no command follows the DO, control returns to the next (numerically greater) Step in that part (or back to the FOR command, if the DO was invoked in the range of a FOR). If there is no numerically greater Step in that Part, the program is terminated, and either control returns to Direct Mode, if the terminal user was logged in with a Programmer Access Code (PAC), or the terminal session is ended if the user was logged in simply to run the program.[1]

Examples:

1. `1.50 IF A=B!(C>D) DO 4`
2. `3.60 FOR X=1:5:45 DO 3.12,5,10.01`
3. `10.01 FOR K=2,3 DO 6:'(A=B),3.24`

Transfer out of the range of a DO using the GOTO command is legitimate and effectively alters the range of the DO to include all Steps and Parts specified by the GOTO.

1. See the QUIT command, Section 3.3.19, for a discussion of the logical levels of program execution.

ELSE

3.3.5 ELSE Command

Mode: Indirect

Syntax: Else␣next command

Description:

The command provides a means for testing the Boolean sense of the last IF command executed (Paragraph 3.3.13). When the sense of the last IF is False (O), commands following the ELSE on the same line will be executed. Otherwise, control will pass to the next program Step.

Note that the use of the ELSE command (like the IF command with no argument), is different from the classical use of this command in other high level languages. Instead, its action is completely dependent on the Boolean truth value established by the execution of the last IF command and in no way related to its position with respect to other IF commands.

Example:

```
3.26 I AGE>19 T "THANKS,! D 50 E T "NO ENTRY",! Q
3.28 E T "NO GOOD",! Q
3.29 T "DONE",!
50.10 I $D(↑AGE(AGE)) S ↑(AGE)=↑(AGE)+1 Q
```

In this example, if the condition in 3.26 is True, Part 50 is executed. Otherwise, the message "NO GOOD" is output. In Part 50, another IF condition is tested. Regardless of the outcome, control returns to the ELSE command in 3.26. If the Part 50 condition was False, the message "NO ENTRY" is output. If the condition is True, control passes to the ELSE in 3.28 which, in turn, passes control to 3.29 causing the message "DONE" to be output.

NOTE

An attempt to use ELSE from Direct Mode is interpreted as ERASE since only the first letter of a command is interpreted.

ERASE

3.3.6 ERASE Command

Mode: Direct

Syntax: Erase { :bve} { |␣spn ␣␣ {:spn} ,...| }

Description:

This command will delete an individual Step or Part, a range of Steps or Parts, or an entire program in the user's partition. Arguments must either be legal Step or Part numbers or be number value expressions (nve) which result in legal spn. The optional second spn is used to specify a range (inclusive). An ERASE with no arguments (terminated simply by EOM or two spaces if other commands exit on the line) deletes the entire program.

Examples:

1. >E 2.01 — erase Step 2.01
2. >ERASE 2.01:4.10 — erase Step 2.01 through Step 4.10
3. >E 4 — erase Part 4
4. >E — erase entire program
5. >E L A D 1 — erase entire program, load program A and start it at Part 1.

FILE

3.3.7 FILE Command

Mode: Direct or Indirect

Syntax: File {:bve}␣{pnam},...

Description:

The FILE command stores (files) the program steps currently residing in the user's partition on the disk and enters the program name (pnam) in the program directory associated with his UCI. If the FILE command does not have an argument, the current program name is assumed.

After a program has been filed, it still remains within the program buffer. The user can continue to run it, modify it, and refile it. Every time a FILE command is issued under the same program name, the program steps currently present in the program buffer completely replace the previous program filed on disk.

Example:

```
>F TST
```

Filed programs are deleted by filing a dummy program of zero length with the same name as the program to be deleted. This is accomplished by erasing the contents of the program buffer and subsequently filing that empty buffer under the program name to be deleted.

Examples:

1. `>E  F A9L` Deletes program A9L
2. `>E  F SAM,ABE,A` Deletes programs SAM, ABE and A

FOR

3.3.8 FOR Command

Mode: Direct or Indirect

Syntax: For␣lvar= { nve_1 { $:nve_2:nve_3$ } ,... } { $nve_1:nve_2$␣ WHILE/UNTIL ␣bve } ␣next command

where: lvar = Index Variable

where: nve_1 = Initial value of lvar
nve_2 = Value by which lvar is incremented
nve_3 = Limit value of lvar

Description:

The FOR command produces efficient looping (iteration) by repeating commands residing on the same line for a specific set of variable values. In operation, the local variable (lvar) is set to the value specified by the first argument (nve_1) and the commands on the remainder of the line are executed. The process is repeated for each new value (if any) of the first argument then the second, third, etc., until the lvar has been set to all values in the argument list.

The WHILE and UNITL clauses can be used to test the status of logical conditions external to the FOR loop. Only one of these clauses can be used at a time and must be the last argument.

Iteration is terminated in one of several ways:

1. The argument list becomes exhausted

2. A QUIT or GOTO command is encountered (Sections 3.3.20 and 3.3.10)

Upon termination, the index variable contains the last value assigned prior to termination.

The arguments used (nve_1, nve_2, and nve_3) may be assigned any value in the legal range of MUMPS numbers (see Section 1.4.1) including negative values. However, if the increment value (nve_2) is given a value of zero, an interminable looping condition will occur unless either the "WHILE/UNTIL" syntax is being used or a QUIT or GOTO is executed.

There are two distinct forms for FOR command arguments which can be used either separately or together, as required. The first is the list format which excludes the optional nve's ($nve_2:nve_3$). With this format, each argument represents one specific value to which lvar is assigned.

Thus:

FOR␣lvar=nve,nve,nve,...

The second form is the range format in which the optional nve's (nve_1 :nve_2) are used. Each argument may represent a range of values.

Thus:

FOR␣lvar=nve_1:nve_2:nve_3,nve_{1a}:nve_{2a}:nve_{3a},...

Both forms can also be intermixed:

FOR␣lvar=nve_1,nve_{1a},nve_{1b}:nve_2:nve_3,...

NOTE

> The indirection syntax operator (3.1) may not be used with arguments of a FOR command (e.g., 'F␣←') causes a SYNTX error.

Examples:

1. List Format

```
FOR X=1,4,10 TYPE "X",X
```

The loop will be repeated three times with X taking on the values 1,4, and 10.

2. Range Format

```
FOR X=1:1:10 TYPE "X",X
```

The loop will be repeated 10 times with X starting at 1 and increasing by 1 each time until it is equal to 10.

```
FOR X=1:1 WHILE Z>X TYPE "X",X
```

This loop will be repeated until X becomes equal to or larger than Z.

3. Range and List Formats

```
FOR I=5,8,33:6:57 TYPE I+(I/3),!
```

In this case, I will take the values 5, 8, 33, 39, 45, 51, and 57.

4. Special Cases

FOR I=A:1:Y DO 3	if A initially greater than Y part 3 is never done
FOR I=1:-1:-2 DO 1.05	step 1.05 is 'done' four times (for I=1, 0, -1 and -2)
FOR I=1:A:10 D 3.21	if A=0, an interminable loop on step 3.21 is initiated.

GO

3.3.9 GO Command

Mode: Direct

Syntax: Go {:bve}

Description:

The GO command is used to restart a MUMPS program which has been interrupted by the BREAK command (Section 3.3.2). This command can only be used after a BREAK has been executed and while it is still in effect. This means that GO cannot be successfully executed after a CTRL C has been typed or after the occurrence of any MUMPS error.

GOTO

3.3.10 GOTO Command

Mode: Indirect

Syntax: GOTO {:bve} ␣ spn {:bve} ,...

Description:

This command permits transfer of control from the current Step sequence to the specified Part or Step number. Once the change in control is effected, program execution progresses in the normal ascending Part/Step number sequence. GOTO can also be used to prematurely exit from a FOR command loop. However, GOTO cancels all previous FOR commands up to the last DO or CALL, and execution proceeds from that DO or CALL (refer to Section 3.3.4).

The argument can be either an actual Step or Part number or numeric valued expression (nve) which evaluates to a legal Step or Part number. Each argument in the list can be modified by an optional Boolean valued expression. It is reasonable to have multiple arguments only if they are modified using :bve's; otherwise the first argument would be the only one considered.

Example:

```
1.50 G 2.1:X<10,3.1:X>10,4.1
```

Control is transferred to Step 2.1 if X is less than 10; to 3.1 if X is greater than 10; and to 4.1 in all other cases.

HALT

3.3.11 HALT Command

Mode: Direct or Indirect

Syntax: Halt {:bve} ␣ ␣

Description:

This command terminates a MUMPS job and causes terminal sign-off.

NOTE

The difference between HALT and HANG is that HALT takes no arguments (:bve is not considered to be an argument).

Examples:

Direct Mode:

```
>H
```

Indirect Mode:

```
3.51 I A>B H
10.02 H:A>B
```

In both of these examples, HALT is executed if A is greater than B.

HANG

3.3.12 HANG Command

Mode: Direct or Indirect

Syntax: Hang {:bve} ␣ nve {:bve} ,...

Description:

This command suspends program execution for a specified time interval. The time interval (nve) is specified in seconds and must be a positive MUMPS number. The number is evaluated as an integer (i.e., the decimal point is ignored). If the nve equals zero, the remainder of the program's time slice (time sharing interval) is given up. When the specified time has elapsed, program execution resumes at the command following HANG. The maximum value which may be specified by nve is 65,535. If nve evaluates to a larger value, the maximum value is used and no error is generated.

This facility is especially useful in applications where the programmer periodically wants to check the status of a variable and take action when the variable has changed.

Example:

```
1.01 I $T<X H 300 G $L
1.21 C %DL
```

If the number of seconds since midnight ($T) is less than X, suspend program execution for 5 minutes and then check $T again; otherwise, call program %DL.

IF

3.3.13 IF Command

Mode: Direct or Indirect

Syntax: IF␣ | bve,...␣ (␣) | next command

Description:

This command is used to effect a change in a program's operation based on the validity of one or more Boolean Valued Expressions. Each bve in the argument list is evaluated. If all expressions are True (non-zero), command processing continues with the next command on the line. If any expression is False (zero), command processing for the remainder of the line is discontinued and the next Step is executed. IF may also be used without arguments, in which case the condition to be tested is the sense of last executed IF statement. The ELSE command is used to test the logical reverse of an IF (see Section 3.3.5).

Example:

```
2.08 IF A=B!(C=>D),NAM="JACK" DO 3

6.03 IF  GOTO 14.36
```

If A equals B, or C is greater than or equal to D, and NAME equals the string JACK, all the commands in Part 3 are executed. Assuming this is the case, (True) and there are no other intervening IF statements which result in a False condition, the execution of 6.03 will result in control passing to step 14.36.

When the Indirection Syntax is used, it must be the last argument. Further, the variable referenced by the indirection (i.e., ←variable) may contain commands[1] as well as additional arguments to the IF Command.

Example:

Where: a, b and c are arguments
x, y and z are commands

```
1.1 S␣D="a,b,c␣x␣y␣z"
1.2 IF␣←D
1.3 ......
```

As with IF without indirection, each argument is processed until a False result is obtained or a command is reached. Once a False argument is reached, the remainder of the line is skipped and processing continues on the next line. When an indirect reference is

1. Excluding the FOR Command

used, the commands which may follow on the same line are no longer dependent upon the logical result of the IF.

Example:

Where: a,b,c, and d are arguments, and
x and y are commands

```
3.1 SET␣D="c,d␣y"
3.2 IF␣a,b,←D␣x
```

If a and b are True, command x will always be executed, regardless of the truth value of arguments c and d, as long as y is not a GOTO or HALT command.

If nested indirection is used, the basic process remains the same. Suppose there are three levels of nesting (three indirect references). If all arguments up to the first indirect reference are True, the commands following on that line will be executed after the truth value of lower levels has been determined, and any lower level commands have been executed.

If all arguments up to the second indirect reference are True, commands following on that line will be executed after the truth value of lower levels has been determined, and any lower level commands have been executed. In general if all arguments on all levels are True, all commands on all levels will be executed, beginning at the deepest level.

EXCEPTION

Any GOTO or HALT Command will prevent execution of commands at all levels above it.

If one of the arguments contained in the second indirect reference happens to be False, the rest of the arguments on that line as well as the commands and arguments specified by the third indirect reference will not be processed. Rather, any commands following the second indirect reference will be executed followed by the first indirect reference.

Example:

Where: b, e, h, k and l are arguments
c, f, i, m and n are commands (other than GOTO)

```
6.1␣S␣A="b␣c"
6.2␣S␣D="e,←A␣f"
6.3␣S␣G="h,←D␣i"
6.4␣IF␣k,l,←G␣m␣n
```

If k and l are True, m and n will always be executed. If h is True, i will always be executed. If e is not True, the commands and arguments in A will not be reached and f will not be executed. If k, l, h, e and b are all True, c, f, i, m and n will be executed in that order.

Examples:

```
1.  1.1␣S␣P="A>B␣S␣A=B"
    1.2␣R␣"A=",A,!,"B=",B,!␣IF␣←P␣T␣B
```

In this example, the variable P is set to a string which is to be used as an argument to the IF in Step 1.2. Step 1.2 requests values for A and B; then the IF is evaluated. If A is greater than B, A is SET to the value of B. The value of B is typed regardless of the logical outcome.

2.
```
>2.1␣R␣"A=",A,!,"B=",B,!␣IF␣←P␣I␣␣T␣B
```

This example is a variation of Step 1.2 in the example above, in which a second IF has been added to permit the typing of B only if A is greater than B.

3. This example is taken from a MUMPS System Program. The variables RK, RF and RP (i.e., RK03, RF11 and RP11) have been previously set to the number of disk drives of each type in the system, or zero if there are none.

Step 1.1 is a string which is to be used in a subsequent indirect reference by means of the $STEP Function. The string could not be contained in a variable since the string itself contains a literal which must be delimited by quotation marks.

Step 2.2 requests the name of the System Disk placing the response in SD.

Step 2.3 contains an IF command with two arguments (a) and (b), and a GOTO (c).

```
1.1␣←SD␣E␣T␣"THERE ARE NO",SD,"DISKS IN THE SYSTEM"␣G␣2.2
.
.
.
2.2␣R␣"TYPE THE NAME OF THE SYSTEM DISK",!,SD
2.3␣IF␣(SD="RK")!(SD="RF")!(SD="RP"),←$S(1.1)␣GOTO␣4
                     (a)                  (b)       (c)
2.4␣T␣!,"RK, RF OR RP",!␣G␣2.2
```

Analysis:

1. SD contains the response obtained in Step 2.2. In 2.3 SD is tested to determine whether it contains the string RK, RF, or RP. If it is none of these, control passes to Step 2.4 and the indirect reference is never executed.

2. If the response is correct, the indirect reference is reached. It ensures that the specified disks exist. The first element in 1.1 is another indirect reference which permits testing of the contents of SD. This results in a Boolean evaluation of the contents of one of the variables: RK, RP or RF, to see if it is non-zero (i.e., some disks of the desired type exist). Since there are no commands in SD, commands following the second indirect reference are executed. The ELSE Command tests the outcome of the IF and causes the message beginning "THERE ARE..." to be output if False condition exists.

3. If a True condition results, control returns to Step 2.3 from the indirect commands in 1.1 and the 'G␣4' is executed.

KILL

3.3.14 KILL Command

Mode: Direct or Indirect

Syntax: Kill {:bve} { |␣variable,...| }
|␣ ␣|

Description:

The KILL command is used to delete both local and global variables. When used without arguments, all locally defined variables are deleted. Examples of this syntax are as follows:

```
10.50 K  S A=50

>K                    all local variables are deleted
```

When KILL is used on a subscripted variable, it is possible to delete any one of the array variables by its full name or all the variables by simply stating the array name. Simple variables need only be named.

Examples:

1. `3.01 K ABC(3)` — Local array element 3 is deleted from the array ABC.

2. `6.58 K ↑DEF(5),↑DEF(6),↑DEF(7)` — Global array elements 5, 6 and 7 are deleted from the array DEF.

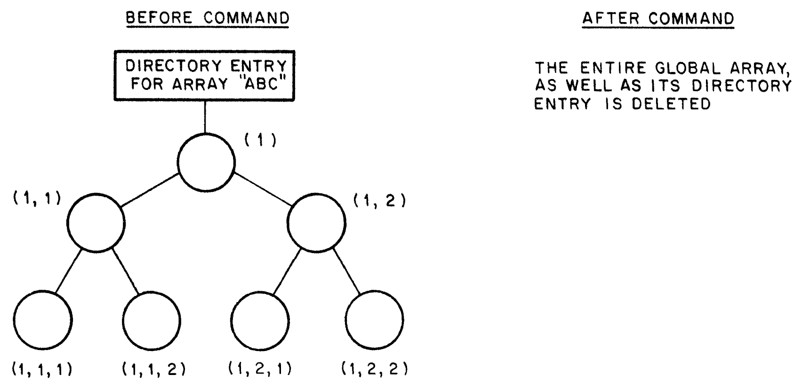
3. `1.99 K ABC` — Deletes all elements in the array ABC.

4. `>K X` — The local variable X is deleted.

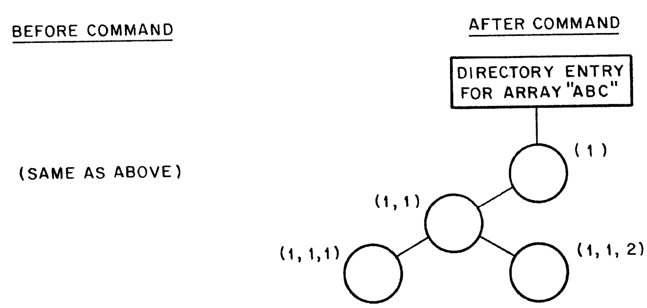
If both a simple variable and a local array are defined under the same name, a KILL referencing that name deletes the array as well as the simple variable.

The KILL command, when applied to global variables, can kill all the data in a global or "prune" the global array at any specified node.

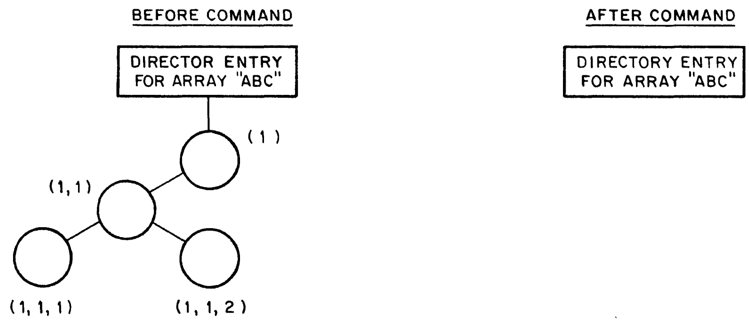
5. >K ^ABC



6. >K ^ABC(1,2)



7. >K ^ABC(1)


BEFORE COMMAND
DIRECTOR ENTRY FOR ARRAY "ABC"
(1)
(1,1)
(1,1,1)
(1,1,2)
AFTER COMMAND
DIRECTORY ENTRY FOR ARRAY "ABC"

LOAD

3.3.15 LOAD Command

Mode: Direct

Syntax: Load {:bve} { | ␣pnam / ␣ ␣ | }

Description:

This command loads a program from the disk into the user's partition. If a program name (pnam) is specified, the user's Program Directory is searched, his program buffer is erased, and the program is loaded. If no argument is given, loading occurs from the device specified in the previous ASSIGN command on the same line (i.e., the current value of the $I System Variable). In this case, the program buffer is not erased and the loaded program is merged with the contents of the program buffer. Steps in the loaded program take precedence over Steps in the program buffer having the same number. In either case, all local variables in the partition are preserved.

Examples:

1. >LOAD SAM — loads program SAM from user's program library

2. >A 2 L FILE SAM — loads program from device #2 (paper-tape reader), then files it under the name SAM.

LOCK

3.3.16 LOCK Command

Mode: Direct or Indirect

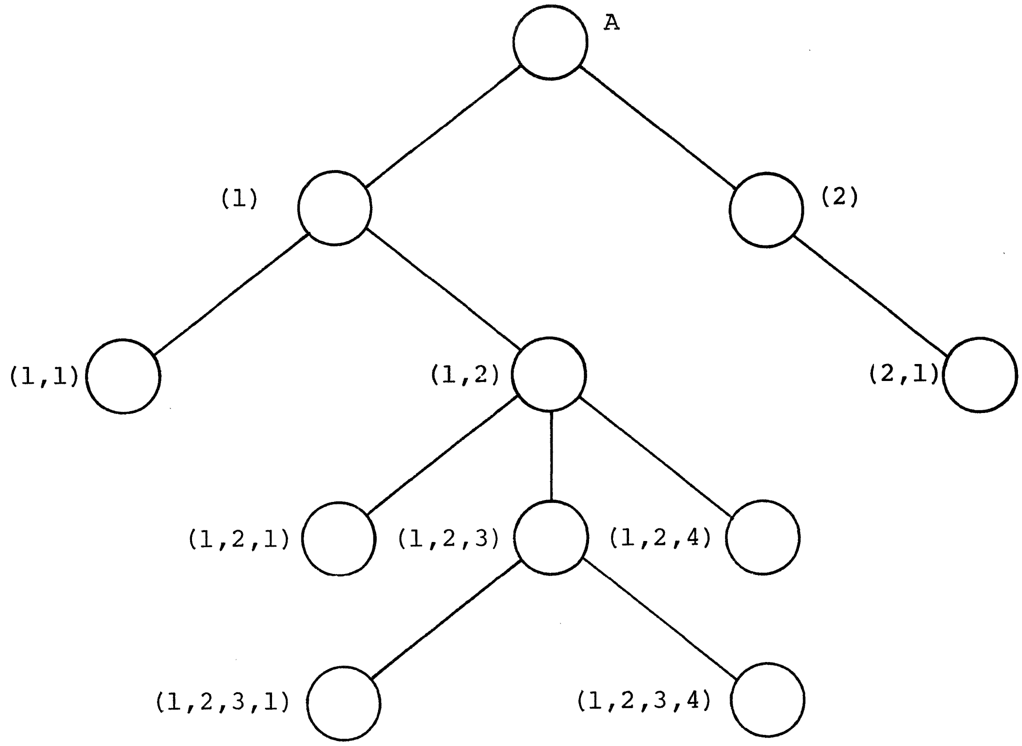
Syntax: LOCK {:bve} ␣ gvar | ␣ (gvar$_1$,gvar$_2$,... gvar$_n$) {:nve}

Description:

The LOCK command provides ownership of global variables on a node level. After execution of the LOCK command, the LOCKed node, all global variables in the tree structure directly below the LOCKed node or global variables above it in a direct path to the top of the global are unavailable for locking by other users. Use of the LOCK command is not mandatory; protection is provided only through application programming convention, and only when all users also use the LOCK command. The LOCK command does not require any use of the disc.

Example:

Given the global ↑A below,



After execution of LOCK↑A(1,2,3), the following nodes are unavailable for LOCKing while node↑A(1,2,3) is locked:

↑A(1,2,3,4)
↑A(1,2,3,1)
↑A(1,2)
↑A(1)
↑A

The following nodes of ↑A would be available for LOCKing by other users while node ↑A(1,2,3) is LOCKed:

↑A(2)
↑A(2,1)
↑A(1,1)
↑A(1,2,1)
↑A(1,2,4)

The execution of a LOCK command unlocks all global variables that were previously LOCKed by the user.

For example, after this series of commands are executed:

```
1.20   LOCK ↑A(1,2)
1.30   LOCK (↑A(2,1),↑A(1,1),↑A(1,2,4))
1.40   LOCK (↑A(1,1),↑A(1,2,3,4))
```

only global variables ↑A(1,1) and ↑A(1,2,3,4) would be LOCKed.

When using the multiple argument form of the LOCK command care must be taken to enclose multiple arguments in parentheses. If there are no parentheses the command will be interpreted as multiple LOCK directives rather than as one LOCK directive with multiple arguments.

For example, the following command:

```
1.10 LOCK ↑A(1,2,3) ,↑B(4) ,↑ABC(3,1)
```

would be the same as this sequence of commands:

```
1.10 LOCK ↑A(1,2,3)       locks ↑A(1,2,3)
1.15 LOCK ↑B(4)           unlocks ↑A(1,2,3), locks ↑B(4)
1.20 LOCK ↑ABC(3,1)       unlocks ↑B(4), locks ↑ABC(3,1)
```

and only global variable ↑ABC(3,1) will be locked after execution of the commands.

If a global variable is unavailable (LOCKed by someone else) when a LOCK request is made, the job will hang without any LOCKed nodes until the requested node(s) is free. If multiple arguments were used in the request, MUMPS will not LOCK any of the global variables until they are all available to be LOCKed.

The optional parameter (:nve) permits the MUMPS programmer to specify how long the system can wait to be able to LOCK the specified global variable(s) if they are unavailable when the request is initiated. The :nve must be a positive MUMPS integer, the decimal fraction is ignored. If MUMPS is unable to LOCK all the specified global variables before the specified time interval has elapsed, none of the globals will be LOCKed and control will be returned to the user.

The MUMPS programmer may check the result of a timed LOCK by inspecting bit 4 of the $JOB system variable. Bit 4 will be set to one if MUMPS was unable to perform the LOCK.

Example of a timed LOCK:

```
LOCK (↑A(1,2),↑B(3)):5
```

If MUMPS is unable to LOCK both ↑A(1,2) and ↑B(3) in a five second time frame, then neither global variable will be LOCKed, bit 4 of $JOB will be set to one, and control will be returned to the program. If the LOCK was successful then both ↑A(1,2) and ↑B(3) are LOCKed and bit 4 of the $JOB will be set to zero.

The following issues are important aspects of the LOCK command.

1. Use of LOCK is not required; protection is only through application programming convention.

2. LOCK does not use the disc. Furthermore, the LOCKed nodes do not have to correspond to existing nodes on the disc.

3. When performing a LOCK, all nodes for that user are first automatically UNLOCKed.

4. All arguments of the LOCK command must be full global references; naked expressions are not allowed.

5. The gvar expressions do not affect the naked global level.

6. A LOCK only pertains to the user's UCI. LOCKS under other UIC's are not affected.

MODIFY

3.3.17 MODIFY Command

Mode: Direct or Indirect

Syntax: Modify {:bve}␣ | spn:x sve_1x sve_2 |
| spn: sve |

Description:

The MODIFY command provides program editing capabilities by altering the contents of a Step. The command causes a search within the specified Step for sve_1 and if found, replaces it with sve_2. Argument delimiters, specified by x, may be any character. The only restriction is that the character used to delimit sve_1 and sve_2 should not be included in either expression.

NOTE

Step numbers cannot be changed with the MODIFY command.

If sve_1 is null (i.e., xxsve_2x), sve_2 will be inserted at the beginning of the Step command line. If sve_2 is null (i.e., xsve_1 xx), sve_1 will be deleted from the Step. The WRITE command is used to display the altered Step.

Example:

```
1.01 R "NAME",NAM,!             old line
>M 1.01:/"/"YOUR /              modify the line
>W 1.01                         write it out
1.01 R "YOUR NAME",NAM,"        new line
```

When MODIFY is used in Indirect Mode, the spn of the Step containing the MODIFY command must be less than the spn specified in the command. If it is not, a PROTect error is generated. Further, MODIFY cannot be used to increase the size of a program.

The second version of the MODIFY command is used for step creation. This version creates the step specified by spn and uses the value of sve for the step's contents. The spn must be previously undefined.

Example:

M 2.12:".S␣A=B"

creates the step

2.12 ␣S␣A=B

OVERLAY

3.3.18 OVERLAY Command

Mode: Indirect

Syntax: Overlay {:bve} ␣ pnam {:spn}

Description:

This command loads and starts programs residing in the user's Program Directory as well as the System Library. Program execution begins at the lowest non-zero Part unless a Step or Part number (spn) is specified. When spn is used, execution starts where specified.

NOTE

> If a non-existent :spn is specified, a NOPGM error is generated. The name of the program and the next higher spn available is printed out following the NOPGM message.

Local variables remain unchanged unless the overlaying program changes them.

When the currently controlling program control command is a DO and an OVERLAY is executed, that DO is effectively converted into a CALL for the remainder of its duration. That is, execution of a QUIT in the program which is OVERLAYed would result in a return to the DO in the program which contained the OVERLAY command.

OVERLAY is similar to the GOTO command except that the program flow is transferred to another program. The program flow does not return to the program containing the OVERLAY command unless the OVERLAY was executed when a DO was the currently controlling program control command.

The OVERLAY command takes an average of one disk access to load the specified program and is therefore faster than the CALL command which takes two disk accesses. This should be taken into account when designing a system of application programs.

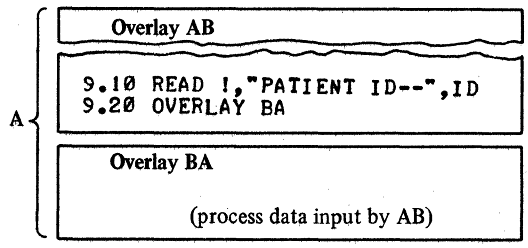
Example:

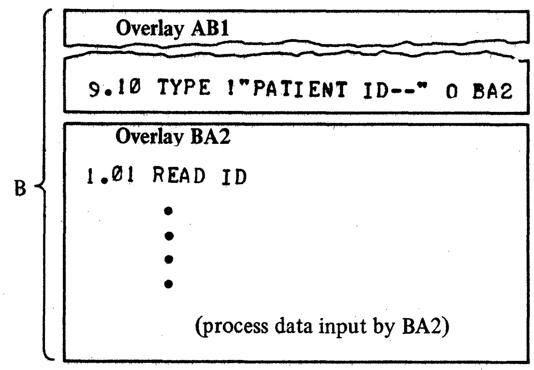
7.61 OVERLAY SAM:1.52	brings program SAM into memory and starts it at Step 1.52.

When using OVERLAY, it is more efficient to execute the command at a point where the current program segment becomes 'I/O bound'. This permits the time taken by the overlaying process to be 'submerged' by the I/O processing time.

Example:

Assume that a program consisting of two overlays called AB and BA requires input from a terminal at some point in its operation. This can be accomplished as shown in either A or B (below). However, the time taken to type out "PATIENT ID", as shown in B, is also used to submerge the time needed to effect the overlay of BS as well.





PRINT

3.3.19 PRINT Command

Mode: Direct or Indirect

Syntax: Print {:bve} | nve / literal / format | ,...

Description:

This command is used primarily to output device dependent control characters to the currently assigned I/O device ($I System Variable). Device dependent data is output using nve to represent a decimal integer value whose 7 low-order bits are accepted as ASCII. The fractional portion of the nve (if any) is ignored. Thus, the programmer can take advantage of the control functions of a particular device.

Example:

1.03 PRINT 7,13,12	will (on a teleprinter) ring the bell, return the carriage without a LINE FEED, and FORM Feed.
7.23 P 29,31	will, on a VT05, move the cursor to the upper left corner of the screen and clear the screen.

Arguments to the PRINT command may also be MUMPS-11 format control characters (#,?nve,!) or literals. For example, the command

6.50 PRINT #,7,?20,"YOU WIN"	causes a teleprinter to: perform a FORM Feed, ring the bell, tabulate 20 spaces from the left margin, and type: YOU WIN

Special nve arguments to the PRINT command allow the programmer to change system protection parameters in the $J System Variable and to effect control functions for magnetic tape I/O operations. System protection arguments are:

1. P␣1024 enables Library Program and Global update.
2. P␣2048 disables Library Program and Global update.[1]
3. P␣256 enables memory or disk write with VIEW command.
4. P␣512 disables memory or disk write with VIEW command.[1]

1. System default condition.

If an error occurs (SYNTX, PROT, etc.) after a P␣1024 or a P␣256 is issued, these parameters are reset to the system default condition (P␣2048 and P␣512). The MUMPS Programmer's Guide provides more details on the use of the $J System Variable.

Special arguments to control magnetic tapes are also described in the MUMPS-11 Programmer's Guide.

QUIT

3.3.20 QUIT Command

Mode: Direct or Indirect

Syntax: Quit {:bve}

Description:

The QUIT command terminates a logical process, including the execution of a Step, Part or program. The command is often used to prematurely terminate operations which are executed within the range of the DO, FOR, and CALL commands.

To understand the QUIT command, it is useful to think of a program's execution as occurring at different logical levels. The first or lowest level is simply operation in Indirect Mode itself. Higher levels are attained by the use of the DO, FOR, and CALL commands and their subsequent nesting. Each time one of those commands is encountered within its own range or that of another, the level is raised by one. The normal termination of these commands lowers the level by one. When QUIT is executed, the current level is also lowered by one and the associated DO, FOR, or CALL command is terminated. When the terminal user is logged-in to the system with a Programming Access Code (PAC), a QUIT at the lowest level switches control to Direct Mode. When logged-in simply to run a program (i.e., UCI:pnam↵), QUIT at the lowest level ends the session at the terminal.

Examples:

1.
```
1.01 FOR I=1:1:100 S A=A+I Q:A>X
1.02 ....
```

In this program if A becomes greater than X, the QUIT prematurely terminates the FOR loop and control passes to Step 1.02. Otherwise, the FOR loop terminates normally after 100 iterations.

2.
```
8.10 FOR X=3:33:3300 D 9 I X+P=A Q
 :     └─ 2nd level     └─ 3rd level
 :
8.3
9.10 I PR=<1.2 Q
4th level ─┐    └─ return to 2nd level
9.12 CALL A Q
```

In this program, the level is raised by one when the FOR loop is entered. When the "D␣9" in 8.1 is executed, the level is raised to 3. When PR=<1.2, the QUIT returns to the 2nd level and the rest of the FOR loop is performed. If PR is not =<1,2, program A is called, raising the level to 4. When program A completes its operation or a QUIT is executed from within it, control returns to the QUIT command following the CALL, which returns control to the 2nd level. When X+P=A, QUIT restores level 1 and Step 8.3 is executed.

READ

3.3.21 READ Command

Mode: Direct or Indirect

Syntax: Read {:bve} ␣ | lvar {:nve} / literal / format | ,...

Description:

This command is used to input one or more lines of characters into specified local variables (lvar) from the currently ASSIGNed input device (value of $I System Variable). Literals and format control characters (Section 3.2) can also be output to the device, provided that it is capable of accepting output (a NODEV error results if it is not).

Each string input is assigned to the specified lvar. Note that all data input is string-valued so the MUMPS programmer wishing numeric data must provide the necessary checks on input strings. (For example, see Section 2.4.4 on pattern verification.)

The optional argument (:nve) permits timed reading by specifying the number of seconds for which the command is to be effective. Each argument in the command can use this feature. It is particularly beneficial when an applications program must deal with terminals which are infrequently attended or unattended.

The :nve must be a positive MUMPS integer; the decimal fraction is ignored. If no input is detected before the specified interval has elapsed, a null string is returned in the lvar, bit 4 of the $J System Variable is set, and the next command on the line is executed. If input occurs before the interval expires, the interval is repeated until one of the following conditions exists:

1. No input has ocurred since the last interval (a null string is returned).

2. A Carriage RETURN or ALT MODE is received.

In the case of (a.), all accumulated characters up to time-out are discarded and a null string is returned; with (b.) however, all characters in the input line are returned.

Examples:

1. `1.32 READ !,"NAME?",NAM(I),!,"AGE=",AGE(I)`

   In this example, the command requests two consecutive lines of input from the terminal. The lvar AGE(I) is assumed to be a one or two digit numeric character string, and the program must convert this if it is desired to store it as numeric data. Automatic mode conversion will be employed when the lvar is used subsequently in the program, however, this does not affect the data mode of the data in AGE(I) unless it is directly altered as in: S␣AGE(I)=AGE(I)+. (Refer to Section 2.3.3 for more information.)

2. `1.36 R "ANYONE THERE?",!,RES:20 I $J&.16 H`

In this example, the message "ANYONE THERE?" followed by a Carriage RETURN/LINE FEED sequence is output. If there is no response within 20 seconds or the operator took more than 20 seconds to type a character of input, the program will halt.

SET

3.3.22 SET Command

Mode: Direct or Indirect

Syntax: Set {:bve} ␣ variable=expression,...

Description:

The SET command assigns the result of an expression to a specified variable. The variable can be simple, subscripted, or global. The variable is followed by an equal sign (=) which in turn is followed by any expression that conforms to the rules for forming expressions (see Chapter 2). The expression is evaluated and the variable is set to the result.

The list of variables and associated expressions is evaluated and assigned from left to right. If a variable used in an expression is set by a previous argument, the value used is that most recently assigned.

Example:

```
1.10 S A=2
1.20 S A=3,B=A*2          B is SET to 6
```

Automatic mode conversion is employed during expression evaluation. The ultimate mode of an expression - string or numeric (including Boolean values) - is determined by the type of the last operator in the expression. It may be a trailing operator. Legal trailing operators include: concatenation (@) to force a string valued result and addition (+) to force a numeric result. (See Section 2.4.7).

WARNING

Special care should be exercised to avoid omitting the equal sign (=) since this situation is not detected as an error. Instead, the command is interpreted to be a START command in which the variable name to the left of the missing equal sign is taken as the name of the program to be STARTed. If a program of that name exists, it is loaded and started; otherwise a NOPGM error results.

Example:

```
1.32 S A=B,DAT D1
              ^
              |________________ missing '=' sign
```

This Step is interpreted as:

SET A = 3, then START a program called DAT at Part 1

Example:

2.5 S $E=20.5	sets the $E System Variable so that all MUMPS errors (except GARB errors) will trap to Step 20.5 for analysis by the application.
20.5 S ERR=$E,$E=30	saves the contents of $E (a value in the range 0 through -0.36 denoting a specific MUMPS error) and resets $E to trap to spn 30 in case any error occurs prior to completion of application error processing.

START

3.3.23 START Command

Mode: Direct or Indirect

Syntax: Start {:bve} ␣pnam {(nve)} {:spn} ,...

Description:

This command permits a currently executing program to load and start one or more programs that run concurrently in separate partitions. The optional nve specifies the partition size in integer multiples of 128 words. The optional :spn specifies the Step or Part number at which execution is to begin; otherwise, execution begins at the lowest non-zero Part.

NOTE

> If a non-existent :spn is specified, a NOPGM error is generated. The name of the program and the next higher spn available is printed out following the NOPGM message.

Each STARTed program must ASSIGN all required I/O devices. Furthermore, STARTed programs share the Principal I/O Device of the starting program. Before a STARTed program can use the Principal I/O Device, however, the starting program or any other STARTed program must UNASSIGN (U␣0) the device.

Error messages which result from a STARTed program are output to the starting program's Principal I/O Device regardless of the current ownership of that device. However, if a STARTed program is to give up its partition, the Principal I/O Device must be available.

Example:

`3.24 START %DV,%LX:5,ACL:2`	loads and starts %DV, %LX beginning at Part 5 and ACL beginning at Part 2, each in a separate partition.
`1.5 S ↑PI=3.14 S RAD(8):10.25`	sets a global variable PI equal to 3.14 and starts a program RAD at Step 10.25 in a 1K partition. Refer to the MUMPS Operator's Guide for details on partition sizes and availability.

TYPE

3.3.24 TYPE Command

Mode: Direct or Indirect

Syntax: Type {:bve} { ␣ | { expression / format / variable / ␣ } ,... | }

Description:

The TYPE command outputs data to the currently ASSIGNed device ($I System Variable). Arguments can be expressions or the format control characters (#,?nve or !) described in Section 3.2. If no arguments are specified, the current values of all local and System Variables are output.

Examples:

1.36 TYPE "VALUE=",A,!	results in 'VALUE=contents of A' followed by a Carriage RETURN,LINE FEED sequence.
>TYPE	types out the contents of all local and system variables.
2.50 TYPE #!,"A+B*C=",A+B*C	types FORM Feed, Carriage RETURN, LINE FEED sequence, 'A+B*C=results of a+b*c.'

UNASSIGN

3.3.25 UNASSIGN Command

Mode: Direct or Indirect

Syntax: Unassign {:bve} ␣ nve,...

Description:

The UNASSIGN command releases the specified I/O device(s) and associated buffers from the ownership of the current job for use by other programs (i.e., it reverses the effect of the ASSIGN command). At least one argument must be specified or a syntax error is generated. Arguments which reference nonexistent devices or devices not previously ASSIGNed are ignored. The nve is interpreted as an integer; decimal fractions are ignored.

A program's Principal I/O Device (device on which terminal user logged-in) may also be UNASSIGNed to permit its use by other programs. The operating system automatically reassigns it when an error is detected or Direct Mode is entered.

Example:

>U 1,3,63 Unassigns devices 1,3 and 63.

UNLOCK

3.3.26 UNLOCK Command

Mode: Direct or Indirect

Syntax: UNLOCK {:bve} ␣ ␣

Description:

The UNLOCK command releases the global variable(s) from ownership of the current job for use by other programs (i.e., it reverses the effect of the LOCK command). All previously LOCKed global variable(s) will be UNLOCKed.

An UNLOCK is automatically performed when a job is HALTed.

VIEW

3.3.27 VIEW Command

Mode: Direct or Indirect

Syntax: View $\{:bve\}$ ␣ nve_1 $\{:nve_1\}$,...

Description:

This is a special purpose command permitting both reading and writing of disk storage blocks in the system's data base, as well as the writing of memory locations. The command aids in the creation of MUMPS application and system programs where the direct modification of disk or memory is required. It is assumed that the user of VIEW is familiar with the system's file structure and the memory-resident system tables described in the MUMPS-11 Programmer's Guide, particularly the system table (SYSTAB). Further, the use of VIEW is restricted by several levels of protection, since its use by unqualified individuals could seriously degrade system operation.

The function performed by VIEW depends upon the presence of the optional nve_2. When nve_2 is not specified (and device No. 63 is assigned), VIEW operation is directed to disk. The address of a disk block to be accessed and the logical disk number is specified by nve_1. If nve_1 is positive, the specified disk block is read; if negative, the block is written. Only the integer part of the nve's are used by VIEW; fractions are ignored.

NOTE

> When using VIEW to write to the disk, no other jobs should be running, including the "Garbage Collector".[1] The MUMPS-11 Operator's Guide describes procedures for establishing this condition.

When accessing disk, the following expression must be used for forming nve_1:

nve_1 = TYP*2,097,152+(UNT*262,144)+BLK

where TYP(device type)=
- 0 for RK11
- 1 for RF11
- 2 for RP11
- 3 for RJP04

UNT(unit number)=m, $0 \le m \le 7$

1. The Garbage Collector routine is described in the MUMPS-11 Programmer's Guide.

BLK(block address on unit)= n,0≤n≤
- 4799 for RK11
- 1023 for RF11
- 79,999 for RP11 (RP03)
- 39,999 for RP11 (RP02)
- 170,543 for RJP04

When using VIEW to read or write disk blocks, input from and output to the disk is directed to a special buffer in memory called the VIEW Device Buffer. Each transfer by VIEW causes an entire disk block (256 words) to be read or written from this buffer. The VIEW Device Buffer is accessed by the user via VIEW (to write memory) and the $VIEW Function (to read memory).[1] The address of the VIEW Device Buffer is obtained from the System Table entry labelled "UTLBUF". The address of the System Table is contained in location 44_{10}. Using $VIEW, the buffer address can be obtained as follows:

Where: OFF = OFFSET TO 'UTLBUF' in System Table.

```
1.2Ø S ADR=$V($V(44)+OFF)
              \_/
               1
             \___/
               2
          \_________/
               3
```

1. Get address of System Table.
2. Add OFFSET TO 'UTLBUF' to obtain the 'UTLBUF' address.
3. Get the address of the 'VIEW' Device Buffer.

When nve_2 is specified (and device No. 63 or No. 46 is assigned), VIEW operation is directed to memory. The address of the memory location is specified by nve_1, and its contents by nve_2. Since VIEW operates on word (as opposed to byte) addressing, if nve_1 is odd, it is interpreted internally as an even number by subtracting 1. Both nve_1 and nve_2 must always be positive when addressing memory.

NOTE

> The VIEW command allows access to 28K words of memory. For systems with more than 28K words of memory, references to address locations (nve_1) 40960[2]-57342 are interpreted as address locations 0-16382 beginning at the base of the current partition.

There are three levels of protection that control the use of VIEW:

1. The user or program must "own" the use of VIEW by having ASSIGNed either device No. 63 for operations directed to either memory or disk, or device No. 46 for operations directed to memory only (Paragraph 3.3.1).

---

1. Described in Section 4.2.16.

2. This would vary as to how the system is built.

2. To read disk blocks either:

   a. The user must be logged in with the System's UCI described in the MUMPS-11 Programmer's Guide, or

   b. the program must be a Library Program (i.e., % symbol must be the first character in the program's name).

3. To write a memory or disk:

   a. Conditions 1 and 2 (above) must be met; and

   b. The Print command, P␣256, must be issued.

Examples:

1. This program zeroes out the VIEW Device Buffer:

```
1.01 A 46 S VBF=$V($V(44)+OFF)
1.02 F I=0:2:510 VIEW VBF+I:0
```

2. The following is part of a program that could be used to copy one unit of an RK03/RK05 Disk Pack to another. N and M are the physical device numbers (RK0, RK1, etc.). Each device has 4800 data blocks.

```
1.01 R "INPUT RK UNIT:",N,!,"OUTPUT RK UNIT:",M,!
1.05 A 63 P 256 F I=1:1:4800 V N*262144+I,-(M*262144+I)
1.10 U 63 P 512 T "DONE",!
```

WRITE

3.3.28 WRITE Command

Mode: Direct or Indirect

Syntax: Write {:bve} { ␣ | spn₁ ␣ {:spn₂} ,... }

Description:

This command is used to output MUMPS programs or individual Steps and Parts residing in the Program Buffer of the user's partition to the currently ASSIGNed I/O device ($I System Variable). WRITE essentially performs the opposite function of the LOAD command.[1] If no arguments are specified, the entire program (all Steps) is output. The optional Boolean expression (:bve) establishes conditional execution.

Spn_1 specifies individual Step or Part numbers, while $:spn_2$ specifies a range of Steps or Parts between spn_1 and spn_2 inclusive. Both Parts and Steps can be intermixed in the same command or its arguments.

Examples:

`>A 3 W`	Write out all Steps on the Line Printer (device #3).
`1.36 W 4.2,1.13,6:7`	Output Steps 4.20, 1.13 and Parts 6 and 7.
`10.04 W 7.14:7.30,1.55:2.03`	Output Steps 7.14 through 7.30 and 1.55 through 2.03.

1. LOAD also inputs programs from the user's Program Directory on disk. FILE must be used to save programs on disk.

XKILL

3.2.29 XKILL Command

Mode: Direct or Indirect

Syntax: Xkill {:bve} ␣lvar,...

Description:

The XKILL (eXclusive KILL) command deletes all local variables and their associated arrays, except those specified in the argument list. This command is an extension to the more general KILL command. Note that subscripted variables are illegal arguments and cause a SYNTX error.

Example:

3.26 X A,B,C G 11.10	Kill all local variables except A, B and C then GOTO step 11.10.

CHAPTER 4

FUNCTIONS

4.1 INTRODUCTION

A function is a component of an expression that invokes an algorithm the result of which is an expression element. Each MUMPS function is identified by a unique mnemonic, the first character of which is always the dollar sign ($). There are two types of functions: numeric and string. Numeric functions return numeric values (nv), while string functions return string values (sv). The value returned is not named and can never be explicitly referenced. The returned value internally replaces the function designation and its arguments within the expression.

4.1.1 Nesting of Functions

MUMPS functions may be nested to the same extent that functions which produce numeric results may be nested within any other function. Functions which produce string valued results may NOT be nested. Furthermore, where the argument to any function is required to be a string value, it must be in the form of a string variable or literal (svl).

4.1.2 Syntax Rules for MUMPS Functions

1. Functions names may be abbreviated to the first character after the dollar sign ($).

2. Arguments are enclosed in parentheses and immediately follow the function name.

3. Multiple arguments are separated by commas.

Table 4-1
Summary of Functions

Type	Name	Action
Numeric		
	$Create (svl)	Creates unique number from 3-character string.
	$Define (| lvar / gvar |)	Checks data type of a variable.
	$Find ($svl_1$,$svl_2$ {,nve})	Finds the position of a given character within a string.
	$High (| lvar / gvar (subscript) |)	Obtains the next element in an array.
	$Integer (nve)	Truncates the fractional part of a decimal number.
	$Length (svl)	Calculates length of a string.
	$M ($marg_1$ { | + - * / > < = >= => <= =< <> >< | $marg_2$... | + - * / > < = >= => <= =< <> >< | $marg_n$ })	Allows floating point calculations.
	$Next (nve)	Obtains next step after nve.
	$Query (gvar)	Finds next (physical) global node.

Table 4-1 (Cont.)
Summary of Functions

Type	Name	Action
String	\$Root (nve)	Finds square root.
	\$View (nve)	Returns the contents of core location.
	\$Altercase (svl)	Converts upper case ASCII to lower and vice versa.
	\$Extract (svl,nve_1 {,nve_2})	Extracts characters from specified positions in a string.
	\$Piece (svl_1,svl_2,nve_1 {,nve_2})	Extracts fields within a string.
	\$Step (nve)	Obtains contents of a step.
	\$Text (nve)	Converts numbers to ASCII.

4.2 FUNCTION DESCRIPTIONS

The following paragraphs define the purpose and use of MUMPS functions. The symbols used to define the syntax of each function are the same as those used in Chapter 3. Definitions of these symbols can be found under the Document Conventions section of this manual. Function descriptions are presented in alphabetic order for ease of reference.

$ALTERCASE

4.2.1 $ALTERCASE Function

Type: String

Syntax: $Altercase (svl)

Description:

The $A function is used to convert alphabetic characters from lower case to upper case and vice-versa. When converting lower case to upper case, lower case character codes in the range 97 through 122 (141_8 - 172_8) are mapped to their upper case equivalents in the range 65 through 90 (101_8 - 132_8). When converting upper case to lower case, upper case character codes are converted to equivalent lower case codes; mapping is the reverse of that specified above. Conversion is performed on a character-by-character basis. The programmer may not nest $A functions in a command string.

Example:

Assuming: NAM(1) = "uncle", NAM(2)="hYpOCraTes",NAM(3)="thomas"

```
1.10 F I=1:1:3 S NAM(I)=$A(NAM(I)) T NAM(I)
```

The above program converts the strings contained in three variables to their alternate case. Thus:

```
UNCLE
HyPocRAtES
THOMAS
```

$CREATE

4.2.2 $CREATE Function

Type: Numeric

Syntax: $Create (svl)

Description:

This function creates a unique positive 21-bit MUMPS number, in the range 0.00-20,971.51, from the first three characters of a specified string. Each character is converted from 8 to 7 bits to permit storage within the 21-bits. Conversion is performed using the following formula:

N=((C(1)*2(14)+(C(2)*2(7))+C(3))/100

Where: N=resulting number
C_1 = decimal character code for 1st character
C_2 = decimal character code for 2nd character
C_3 = decimal character code for 3rd character

The relationship of the characters to the resulting number is shown below:

BIT 31 — 22	21 — 14	13 — 7	6 — 0
0	1st CHAR (C_1)	2nd CHAR (C_2)	3rd CHAR (C_3)

11-1447

If fewer than three characters are available, the characters are left justified within the resulting numbers. The programmer can use the $Text function (Section 4.2.15) to convert the created number back to ASCII.

Example:

$C can be used to create subscripts from strings allowing data to be stored in subscript form. The following command line might be used to create subscripts for a program to maintain a telephone book. Assume three levels of subscripting based on the first nine characters of a last name (S_1 contains the first three characters, S_2 the second group of three and S_3 the last three characters and NUM=telephone number).

```
>SET ↑TEL($C(S(1)),$C(S(2)),$C(S(3)))=NUM
```

$DEFINE

4.2.3 $DEFINE Function

Type: Numeric

Syntax: $Define (lvar | gvar)

Description:

The $DEFINE function checks the data type of either local or global variables. The argument to the function is the name of the variable to be checked. There are eight possible data type values returned:

Data Type	Definition
0	undefined variable
1	single numeric datum
2	string datum
3	double numeric datum
4	pointer to structure at lower level or local array name
5	pointer or local array name and single numeric valued datum
6	pointer or local array name and string valued datum
7	pointer or local array name and double numeric datum
8	4-word floating point numeric (resultant from $M)

Examples:

1. If local variable B contains a numeric quantity less than 327.68, its data type is 1.

```
>T $D(B(2))
1
>
```

2. If global variable ↑ ABC(X,Y) contains the string "JOHN DOE" and has no lower level associated with it, its data type is 2.

```
>S C=$D(↑ABC(X,Y))
>T C
2
```

$EXTRACT

4.2.4 $EXTRACT Function

Type: String

Syntax: $Extract (svl,nve$_1$ {,nve$_2$})

Description:

The $EXTRACT function extracts all the characters from the specified string variable or literal (svl) that are between character positions specified by nve$_1$ and nve$_2$ inclusive. If nve$_1$ is greater than nve$_2$, $EXTRACT returns a 'null string'. If nve$_2$ is equal to nve$_1$ or if nve$_2$ is omitted, $EXTRACT returns the character specified by nve$_1$. Values of nve$_1$ which are less than 1 are interpreted as 1. If the length of the string is such that $E runs out of characters before satisfying nve$_2$, then the function returns all characters between nve$_1$ character and the end of the string. Only the integer part of nve$_1$ and nve$_2$ are considered.

NOTE

1. If the string argument (svl) is a global variable, no other arguments may be global variables.
2. Nesting $E functions in a command string is illegal.

Example:

Assume that the string variable NAM="JOHN DOE" is to be changed to the form: last-name, comma, first-name. The following statements will do it, using the concatenation operator (@) and the $FIND function (Section 4.2.5):

```
>S NAM="JOHN DOE"
>
>1.36 S LST=$E(NAM,$F(NAM," ",1),$L(NAM))
>1.38 S FIR=$E(NAM,1,$F(NAM," ",0)-2)
>1.40 S NAM=LST@","@FIR
>D 1
>T NAM
DOE,JOHN
```

$FIND

4.2.5 $FIND Function

Type: Numeric

Syntax: $Find ($svl_1$,$svl_2$ {,nve})

Description:

The $FIND function returns a number representing the character position of the character following sve_2 within svl_1. The search for svl_2 within svl_1 begins at the first character unless the optional nve is given, in which case the search begins at the nveth character in svl_1. If nve is negative or svl_2 is not found, then $FIND returns zero (0).

NOTE

> Only one of the arguments can be a global variable.

Example:

```
>S STR="ABCDEFGHIL"

>T $F(STR,"A",1)
2                              returns 2

>T $F(STR,"A")
2                              returns 2

>T $F(STR,"A",3)
Ø                              returns 0, since "A" does not occur
                               after third character in the string

>T $F(STR,"GHI")
1Ø                             returns 10

>T $F(STR,"HIJ")
Ø                              returns 0. String does not contain
                               string, HIJ.
```

$HIGH

4.2.6 $HIGH Function

Type: Numeric

Syntax:

```
$High ( | gvar             | )
        | lvar (subscript) |
```

Description:

The $HIGH function is used to locate the next numerically greater subscripted variable in either a local or a global array. $HIGH compares the value of the subscript in the argument to the values of all other subscripted variables in the array (at the same subscripting level). When the variable having the next higher subscript is found, ($HIGH returns the value of that subscript. If there is no higher subscript,) $HIGH returns-0.01. A negative subscript value is used in the argument to determine the existence of a variable with a subscript of zero. If $H detects a subscript that is higher by an increment of 0.01, it terminates the search and returns that value since 0.01 is the smallest allowable increment between two subscripts. For this reason, the use of contiguous subscript values having increments of 0.01 can provide improvements in program execution speed when many $HIGH's must be performed.

Examples:

1. Given local array: A(1),A(2.5),A(3.68)

```
$H(A(1))         returns 2.5
$H(A(2.5))       returns 3.68
$H(A(1.5))       returns 2.5
$H(A(3.68))      returns -0.01
```

2. Given global array:↑B(1),↑B(1,1),↑B(1,2),↑B(1,1,1),↑B(1,1,3), ↑B(1,1,3,0)

```
$H(↑B(1))          returns -0.01
$H(↑B(1,1))        returns 2
$H(↑B(1,2))        returns -0.01
$H(↑B(1,1,1))      returns 3
$H(↑B(1,1,2))      returns 3
$H(↑B(1,1,3))      returns -0.01
$H(↑B(1,1,3,-1))   returns 0
```

$INTEGER

4.2.7 $INTEGER Function

Type: Numeric

Syntax: $Integer (nve)

Description:

The $INTEGER function returns the integer portion of the specified numeric valued expression (nve). The fractional part of the nve is truncated.

Example:

This program checks for odd and even numbers by using $I to discard any remainder resulting from division by 2. If the numbers are equal after multiplying the result by the divisor, the number is even. $I is also used to discard any fractions that are input.

```
>1.10 READ "TYPE A NUMBER -",A,! S A=$I(A)
>1.20 IF $I(A/2)*2=A T "EVEN",! G 1.1
>1.30 TYPE "ODD",! G 1.1


>D 1
TYPE A NUMBER -1
ODD
TYPE A NUMBER -56
EVEN
TYPE A NUMBER -241
ODD
TYPE A NUMBER -2346.02
EVEN
TYPE A NUMBER -
```

$LENGTH

4.2.8 $LENGTH Function

Type: Numeric

Syntax: $Length (svl)

Description:

The $LENGTH function returns the number (quantity) of characters contained in the specified string variable or literal (svl). The length of a string may range from 0 to 132 characters.

Example:

The following steps use $L to format an output line.

```
1.32 READ !,"NAME=",NAM," ADDRESS=",ADR,!                 G 1
1.34 TYPE "NAME: ",NAM,?($L(NAM)+22),"ADDRESS: ",ADR,! G 1

>D 1

NAME=ELSIE PFLUGG ADDRESS=34 GUELPH COURT
NAME: ELSIE PFLUGG                 ADDRESS: 34 GUELPH COURT          RT
```

$M

4.2.9 $M Function

Type: Numeric

Syntax:

```
$M(marg1 { + | marg2 ... + | margn } )
           -                -
           *                *
           /                /
           >                >
           <                <
           =                =
           >=               >=
           =>               =>
           <=               <=
           =<               =<
           <>               <>
           ><               ><
```

Description:

The $M function allows standard arithmetic and relational arithmetic operations to be performed on numbers outside the normal range of MUMPS numbers. $M expressions produce four-word, double precision floating-point results in the absolute value range $.14X10^{-38} \leq n \leq 1.7X10^{38}$ with an accuracy of 17 significant digits. $M expressions produce either a floating-point or fixed-point result depending on the last operation performed. When the last operator is an arithmetic operator, the expression result is a floating-point number which is stored as a data type 8 datum (see $D function description). When the last operator is a relational arithmetic operator, the result is a fixed-point (MUMPS) number, either -0.01 for True relations or 0.00 for False relations.

When a floating-point result is converted to a string, the floating-point data is in the form:

0.nn ...nD mm for positive numbers greater than 1
0.nn ...nD -mm for positive numbers greater than 0 and less than 1
-0.nn ...nD mm for negative numbers less than -1
-0.nn ...nD -mm for negative numbers less than 0 and greater than -1

Expression operands (marg) must be within the subset of standard operands shown below:

1. A constant

2. A simple variable that contains a character string representation of either a MUMPS number or a valid floating-point number. Except for the floating-point output

formats shown above, mixed alphanumeric strings (e.g., "123ABC") are interpreted as zero value.

3. A simple local variable or a global variable that contains the results of a previous $M operation.

4. A subexpression.

Illegal use of $M expressions may produce MODER or $MERR error messages.

Examples:

The following are examples of legal $M expression operands:

```
STR where:     STR ="-6.123D 5"       12
               STR = "5"              19.7346
               STR =".15672"          478655.1
               STR ="67651.98"        -14.07
M where:       M =0.12D-14            X + 4.43*(3.27+Z)
               M =-0.373468D04          where: X = 0.12D-03
                                               Z = "0.17365421D-07"
```

Wherever a string interpretation of a numeric quantity is indicated, a floating-point datum is permitted. However, conversion of a floating-point datum to a fixed-point number is not allowed. Thus, although floating-point numbers can be used with string operators, a floating-point number cannot be used with arithmetic operators outside of a $M function, and care should be taken when using the equality operator with a floating-point numeric.

The following examples illustrate string conversion of a floating-point number.

```
>S␣A = $M(2*4)@␣T␣A,!,$D(A)
 0.8D␣01
 2

>S␣A = $M(2*4),B = A@"***"␣T␣B
 0.8D␣01***

>S␣A = $M(2*4)␣T␣A?"0."D"D␣01"
 -0.01
```

The following examples demonstrate the use of $M with the TYPE command:

```
>S A="2",B="4",C="1A",D=".1D 2"
>T $M(A),"   ",$M(A*4),"   ",$M(A*B),"   ",$M(C),"   ",$M(D)
0.2D 01   0.8D 01   0.8D 01   0.D 00   0.1D 02
>

>S E=$M(A+(B*2)-5) T E
0.5D 01
>

>S G=$M(A-B/10000) T G
-0.2D-03
>

>S G=$M(A-B*10000) T $M(G*1000000)
-0.2D 11
>

>S J=2 T $M(J)

MODER>0 ●
```

```
>S ↑H(1,1)=$M(D*A+(B/2)/6) T ↑H(1,1)
0.3666666666666667D 01
>

>T $M(2),"   ",$M(123456.1234),"   ",$M(.25D 2)
0.2D 01   0.1234561234D 06   0.25D 02
>

>S H=$M(B>A),I=$M(A+5=B) I H T H,"  B>A",! I 'I T I," IT'S FALSE"
-0.01  B>A
0 IT'S FALSE
>
```

$NEXT

4.2.10 $NEXT Function

Type: Numeric

Syntax: $Next (nve)

Description:

The $NEXT function returns the Step number of the first Step following the Step specified by nve. If there are no Steps following the value of nve, $NEXT returns zero (0).

Example:

If a program has the following steps:

```
 1.01
 1.32
 4.91
10.13
then:
$N(1.01)            returns 1.32
$N(1.32)            returns 4.91
$N(2.35)            returns 4.91
$N(4.91)            returns 10.13
$N(10.13)           returns 0
```

$PIECE

4.2.11 $PIECE Function

Type: String

Syntax: $Piece ($svl_1$,$svl_2$ nve_1 {,nve_2})

Description:

The $PIECE function examines the string specified by svl_1, which is assumed to be divided into "fields" delimited by the first character of svl_2. $PIECE returns the string value contained in the fields specified by the two arguments nve_1 and nve_2, inclusive.

If nve_2 is not specified:

1. a null string is returned where $nve_1 \leq 0$
2. the (nve_1th) field is returned (without delimiters) where $nve_1 > 0$.

If nve_2 is specified:

1. and $nve_1 \leq 0$, nve_1 is set to 1.
2. a null string is returned where $nve_2 < nve_1$.
3. the (nve_1th) field is returned (without delimiters) where $nve_2 = nve_1$.

If $PIECE runs out of fields before reaching nve_2, it returns any characters between the delimiter (svl_2) and the end of the string.

NOTE

1. If sve_1 is a global variable, no other argument can be global variables.
2. Nesting $P functions is illegal.

Examples:

Given:

```
STR="34,6.09,JOHN DOE,BOSTON,JUN,22"
DEL=","
X=7
Y=3
```

1.	$PIECE (STR,DEL,3)	returns 'JOHN DOE'
2.	$P (STR,",",Y,X-Y)	returns 'JOHN DOE, BOSTON
3.	$PIECE (STR,"O",4)	returns 'ST'
4.	$P (STR,","8)	returns a null string since there is no 8th field in sve(1)
5.	$P (STR, " ",1)	returns '34,6.09,JOHN'

$QUERY

4.2.12 $QUERY Function

Type: Numeric

Syntax: $Q (gvar)

Description:

This function allows global nodes at a given level to be sequentially processed in the physical order in which they appear. This is particularly useful when used with the naked global syntax. $Q first searches for the node indicated by gvar, and then returns the next physical subscript within that same block (or continuation block). $Q returns -0.01 when the subscript indicated by gvar is the last physical subscript at that level. If a nonexistent global node is indicated by gvar, $Q returns the value -0.02. If the last subscript indicated in gvar is negative, $Q returns the first (physical) subscript at that level.

Example:

This example lists all the nodes of global ↑M at the 5th level in the physical order in which they appear:

```
>1.1 S A=-1
>1.2 S A=$Q(↑M(9,9,8,4,A)) I A>-.01 T A,"**",↑(A),! G $L
```

Note the use of the $L System Variable.

```
>D 1.1
100**100
1**TOM
2**3452
5**243
6**ALFRED
7**789
8**7UU
10**WH.34
3**2ALFRED NEMBHH
4**435 KLIPCH ST
9**JACK O'C

>
```

$ROOT

4.2.13 $ROOT Function

Type: Numeric

Syntax: $Root (nve)

Description:

The $ROOT function returns the square root (numeric value) of nve, to two decimal places of accuracy. The nve must be positive, otherwise a MINUS error is generated.

Examples:

1.
```
>T $R(64)
8
>
```

2.
```
>T $R(2)
1.41
>
```

$STEP

4.2.14 $STEP Function

Type: String

Syntax: $Step (nve)

Description:

The $STEP function returns a string value which is the contents of the Step specified by nve. The string begins with the first character following the space after the Step number, and ends with the last character in the line. This function is useful when programs need to store certain data in non-executable Parts. Nesting $S functions is illegal.

Example:

The program segment below shows what might be part of a command evaluator. The program asks the user for an option number. If a ? is typed, the program types a 'menu' of the options available. The menu is stored under its own part number and each entry is contained within a step as a literal. This part is never executed. The FOR loop indexes on the step numbers containing the 'menu', and $Step extracts each literal, in turn, to be output.

```
>1.10 READ "OPTION?      ",X QUIT:X=""
>1.20 IF X["?" F I=10.1:.1:10.5 TYPE !,$S(I)

>10.10 1 ALPHA SEARCH
>10.20 2 NUMERIC SEARCH
>10.30 3 PAYROLL NO. SORT
>10.40 4 FILE MERGE
>10.50 5 OUTPUT TRANSACTION FILE
```

```
>D 1
OPTION?    ?

1 ALPHA SEARCH
2 NUMERIC SEARCH
3 PAYROLL NO. SORT
4 FILE MERGE
5 OUTPUT TRANSACTION FILE
>
```

$TEXT

4.2.15 $TEXT Function

Type: String

Syntax: $Text (nve)

Description:

The $TEXT function translates the numeric argument nve to return up to four ASCII characters, one per byte. Each byte is masked to 7 bits, starting with the high-order byte. If it is null, it is simply ignored.

$TEXT is primarily for use by system programmers who are familiar with the internal data formats of the MUMPS system. $TEXT is often used in conjunction with the VIEW command to convert the contents of a location known to contain ASCII data. Nesting $T functions is illegal.

Examples:

1. This command line types out the characters contained in the 1st word of the UCI Table.[1]

```
>1.2 A 63 T $T($V($V($V(44)+8))/100)
```

2. Step 1.2 A reconverts a subscript, created in Step 1.1 by $C, back to a string.

```
>1.10 S B=$C("TST")
>1.20 T $T(B/16384)@$T((B/128)&2.55)@$T(B&2.55)

>D 1
TST
>
```

1. Described in the MUMPS Programmer's Guide.

$VIEW

4.2.16 $VIEW Function

Type: Numeric

Syntax: $View (nve)

Description:

The $VIEW function returns an integer that is the decimal value of the contents of the memory location specified by the nve. The function operates on a word (as opposed to byte) address basis. Therefore, even if the nve is an odd number, it is interpreted internally as an even number by subtracting one. (See also the VIEW command, Section 3.3.25.) Only the integer portion of the nve is accepted, decimal fractions are ignored.

The use of $VIEW is restricted to users who are logged in under the system's User Class Identifier code (UCI number 1) and to programs that reside under the control of UCI number 1 (either System Utility programs or Library Programs).

NOTE

1. Protection features do not apply when referencing locations in the PDP-11's External Page (locations 57344_{10} through 65535_{10}).

2. The $VIEW function allows access to 28K words of memory. For systems with more than 28K words of memory, any references for address locations (nve) 40960 - 57342 will be interpreted as address locations 0 - 16382 beginning at the base of the current partition

Examples:

1. $V (0) Examine memory location 0 as a word

2. $V (16.62) Examine memory location 16_{10} as a word (note decimal fraction is ignored)

3. $V (3) Examine memory location 2 as a word

APPENDIX A

GLOSSARY OF TERMS

Array — An array, which can consist of either local or global variables, is a group of subscripted variables that have a common identifier.

Binary Operator — A binary operator is an operator that requires two operands (expression elements).

Boolean Valued Expression — A Boolean Valued Expression (bve) is an expression, which, when evaluated, produces either a True (-0.01) or False (0) result.

Command — A command is the principal algorithmic component of the MUMPS Language. MUMPS commands consist of a set of keywords that characterize actions. (e.g., GOTO, SET, HALT, RUN, etc.).

Concatenation — Concatenation is the process of linking together two or more string data elements to form a single string. Concatenation is a string expression operation that is designated by the commercial "at" sign (@).

Constant — A constant is a quantity within the range of legal MUMPS numbers (±21474836.47) explicitly stated in an argument to a command or as an operand in an expression.

Data Base — Data base is that body of disk-stored information residing in global arrays.

Direct Mode — Direct Mode is that mode of system operation which enables the programmer to:

1. enter commands and/or functions for immediate execution

2. create or modify steps of a user's program

Directory	A directory is a disk resident table which can contain the names and disk starting addresses of either programs or global files. Each User Class Identifier in a MUMPS-11 system is associated with two directories; a program directory, and a global directory.
Double Numeric Quantity	This term refers to MUMPS numbers whose absolute values lie in the range ±327.68 through 21474836.47 which are stored by the operating system in two consecutive words. (See also Single Numeric Quantity.)
Expression	An expression is any legal combination of operands (elements) and operators. Legal expression elements include: literals, constants, variables, subexpressions, and function references. An expression may consist of a single element an element/operator combination or a series of element/operator combinations.
Expression Element	An expression element is the operand component of a MUMPS expression. An expression element may be a constant, a simple variable, a literal, a local subscripted variable, a global variable, a function reference, or a subexpression.
Floating Point Numeric	A 4-word floating point number in the range $\pm 0.14*10^{38}$ to $\pm 1.7*10^{38}$. The MUMPS $M function allows floating point numbers to be used with the operators + - * / <> =. A Floating Point number may be stored only as a local variable which is not the name of an associated array (i.e., pointer variables are excluded) or as a global variable.
Function	A function is a MUMPS expression component that invokes an algorithm, the result of which is an expression element (operand). Each MUMPS function is assigned a unique mnemonic, the first character of which is the dollar sign ($) symbol.
Global	A global is a tree-structured data file stored in the common data base on the disk. Globals comprise an external system of symbolically referenced arrays.
Global Variable	A global variable is a subscripted variable that forms an element (or node) of a global array.

Identifier — An identifier is a name consisting of one to three alphanumeric characters. The first character must be either an alphabetic character or the percent (%) symbol. Identifiers are used as names for variables, programs, library (or system) programs, and globals. The percent symbol is reserved for naming Library Programs and Globals, though any local variable can use percent as the first character of its name.

IF Switch — The IF Switch is a logical switch that resides in the Program Vector area in each user's partition. This switch is set to the logical result of the last executed IF statement, either True (-0.01) or False (0). Note that an IF without arguments or an ELSE only tests the logical value of the IF Switch and does not change it.

Indirect Mode — Indirect Mode is that mode of system operation in which the steps of a stored program are executed. In this mode of operation, commands cannot be entered from the terminal and programs cannot be created or modified.

Indirect Reference — An indirect reference is a feature of the language that permits a string variable to represent a command's argument or argument list. In operation, the string value of the variable is taken as the argument or argument list. The indirection symbol, back arrow (←) or underscore (_), must precede the variable reference.

Job — A job is any user activity which requires the use of a partition. For example, logging in or STARTing a program are Jobs.

Library Program — This term refers to those programs that are listed in the Program Directory of the System UCI (UCI #1) and have a percent symbol (%) as the first character of their names. Programs residing in the system in this way can be run by any user regardless of UCI.

Literal — A literal is an element of the language that permits the explicit representation of character strings in expressions and in command and function arguments by delimiting them with quotation marks (""). Literals may not contain:

quotation marks	CTRL O	Line Feed
Carriage RETURN	CTRL C	Form Feed
ALTMODE	CTRL U	Vertical Tab
RUBOUT (DEL)	NUL	

Local Variable	A local variable is a variable that resides in the partition of the program that created it (as opposed to a global variable).
Naked Reference	The naked reference is a feature that provides an abbreviated method for accessing global variables to reduce disk access time. This permits subsequent references to a global to be made simply by specifying an up-arrow (↑) followed by one or more subscripts. The variable name is assumed from the last global reference in which a name was explicitly stated. The first subscript in the naked reference replaces last subscript in the previous reference (either naked or complete). Using the naked reference reduces disk access time since the search for the specified node begins at the subscripting level attained by the last global reference rather than at the global directory level.
Node	A node is a global array element addressed by a subscript.
Numbers	Numbers in MUMPS are signed fixed-point quantities in the range ±21474836.47. Decimal fractions greater than two places are truncated to two places.
Numeric Valued Expression	A numeric valued expression (nve) is an expression which, when evaluated, produces a numeric result.
Operator	An operator is a component of a MUMPS expression that invokes an algorithm to perform either arithmetic, string, or Boolean manipulations. (See binary operator and unary operator).
Part Number	A part number is the integer portion of a step number and is used to refer collectively to all steps having a common integer base.
Partition	A partition is the memory area within which a job resides. A partition is allocated to a job either at terminal log-in time or upon execution of the START command. A partition contains both program and local variable storage areas as well as program state information necessary for timesharing operation.

Pattern Verification — Pattern verification is a feature of the language which permits evaluation of text strings for the occurrence of desired combinations of alphabetic, numeric, and punctuation characters. Pattern verification is specified by the "?" operator followed by Pattern Specification Codes (psc).

Principal I/O Device — This term refers to the keyboard terminal that initiated the job. This is the device to which control returns when an error message is to be output or when an ASSIGN␣O command is issued.

Program Name — A program name is an identifier that is associated with a particular program. System Library program names must use the percent symbol (%) as the first character.

Programmer Access Code — The Programmer Access Code (PAC) is a three-character code, created at System Generation time, that allows the terminal user to enter Direct Mode.

Queue — A queue is an ordered list in which the first item to be entered is the first item to be removed (first-in-first-out sequence).

Run Queue — The Run Queue is a System Queue which contains the number of the job currently executing in its time slice. This queue is effectively a one entry queue.

Secondary Storage — This term refers to all I/O devices which are not used to contain the global data base (non-disk), (i.e., paper tape, magtape, or DECtape).

Single Numeric Quantity — This term refers to MUMPS numbers in the range ±327.67 which are stored by the operating system in one 16-bit word. (See also Double Numeric Quantity).

Sparse Array — A sparse array refers to the method of storage allocation used for local and global arrays in which space is allocated only as variables are explicitly defined (unlike other languages which require dimension or size statements for preallocation of storage).

Step Number — A step number is a number used to identify each line of a MUMPS program. A step number must be in the range 0.01 - 327.67, excluding all numbers in this range that are integers.

String — A string is a contiguous combination of any of the ASCII characters. (132 characters maximum)

String Concatenation	See Concatenation.
String Valued Expression	A string valued expression (sve) is an expression which produces a string result upon evaluation.
Subexpression	A subexpression is an expression element that consists of any legitimate expression enclosed in parentheses.
Subscripts	A subscript is a numeric valued expression or expression element which is appended to a local or global variable name to uniquely identify specific elements of an array. Subscripts are enclosed in parentheses. Multiple subscripts must be separated by commas.
Subscripted Variable	A subscripted variable is a variable to which a subscript is affixed (see subscript and variable). Both global and local variables are forms of subscripted variables.
System Program	A System Program is a program either supplied by DEC or created by the MUMPS user which is used to assist the MUMPS system owner in the operational maintenance of the system. System Programs normally reside under the protection of the System UCI (UCI #1).
System Queues	This term refers to the set of queues used by the MUMPS Operating System to control the allocation of system resources (see Run Queue and Wait Queue).
System UCI	The System User Class Identifier (UCI) code is that UCI code assigned to the first entry in the system's UCI table. The Program and Global Directories associated with the System UCI are used to contain both System and Library programs and globals.
System Variable	A System Variable is a variable that is permanently defined within the operating system. These variables provide system and control information to all programs. The first character of a System Variable is always a dollar sign ($). System Variables are maintained and modified by the operating system and/or system manager only.
Time Slice	This term refers to the period of time allocated by the operating system to process a particular partition's program. This term is synonymous with 'timesharing interval'.

Unary Operator	A unary operator is an operator that requires a single operand (expression element).
User Class Identifier (UCI)	A UCI is a three-character code used at terminal log-in time to permit access to the group of programs and global files with which it is associated. When used with the Programmer Access Code, the UCI allows these programs to be modified and new programs to be created.
Variable	A variable is the symbolic representation of a logical storage location. Specific types include local, global, simple and subscripted variables. Variables are symbolically referenced by means of identifiers.
Wait Queues	The Wait Queues are a group of System Queues which contain the numbers of the jobs awaiting service by the operating system.

APPENDIX B

MUMPS CHARACTER SET

The following table shows, with the corresponding octal and decimal equivalents, the 128-character set of 7-bit ASCII code used by MUMPS for data, command, and control purposes. In addition, the order of the character set as shown establishes the MUMPS collating sequence used by the system's Expression Evaluator when establishing string relationships.

For command and control purposes, MUMPS uses the 64-character graphic subset. The system also uses the control codes shown in brackets ([ ]). These codes should not be used as input data. The NUL, code 000, is used internally as a logical end-of-message and cannot be used. Characters shown in braces ({ }) are part of the 1963 ASCII Character Set and may appear in the character set of some terminals.

All characters may be used for data input and output except for these mentioned above. The system does not perform any character conversion. It is the programmer's responsibility to perform all upper/lower-case letter conversions or mappings which are required for the particular application.

CHARACTER SET

Octal Code	Decimal Code	Character	Octal Code	Decimal Code	Character
000	000	NUL	[025	021	NAK (CTRL U)*]
001	001	SOH (Backspace)†	026	022	SYN
002	002	STX (Forward space)†	027	023	ETB
003	003	ETX (CTRL C)* (Write EOF)†	030	024	CAN
004	004	EOT (Write block)†	031	025	EM
005	005	ENQ (Rewind)†	032	026	SUB
006	006	ACK (Read block)†	[033	027	ESC (ALT MODE)*]
007	007	BELL	034	028	FS
010	008	BS* (Read label)†	035	029	GS
011	009	HT	036	030	RS
012	010	LF	037	031	US
013	011	VT	040	032	Space
014	012	FF	041	033	!
015	013	CR	042	034	"
016	014	SO	043	035	#
[017	015	SI(CTRL 0)*]	044	036	$
020	016	DLE	045	037	%
021	017	DC1	046	038	&
022	018	DC2	047	039	'
023	019	DC3	050	040	(
024	020	DC4	051	041	)

*Asterisk denotes the control function for MUMPS terminals, if different from specified or other use.

†Dagger denotes the control function for magtape devices.

CHARACTER SET (Cont)

Octal Code	Decimal Code	Character	Octal Code	Decimal Code	Character
052	042	*	125	085	U
053	043	+	126	086	V
054	044	,	127	087	W
055	045	—	130	088	X
056	046	.	131	089	Y
057	047	/	132	090	Z
060	048	0	133	091	[
061	049	1	134	092	\
062	050	2	135	093	]
063	051	3	136	094	Λ {↑}
064	052	4	137	095	— {←}
065	053	5	140	096	`
066	054	6	141	097	a
067	055	7	142	098	b
070	056	8	143	099	c
071	057	9	144	100	d
072	058	:	145	101	e
073	059	;	146	102	f
074	060	<	147	103	g
075	061	=	150	104	h
076	062	>	151	105	i
077	063	?	152	106	j
100	064	@	153	107	k
101	065	A	154	108	l
102	066	B	155	109	m
103	067	C	156	110	n
104	068	D	157	111	o
105	069	E	160	112	p
106	070	F	161	113	q
107	071	G	162	114	r
110	072	H	163	115	s
111	073	I	164	116	t
112	074	J	165	117	u
113	075	K	166	118	v
114	076	L	167	119	w
115	077	M	170	120	x
116	078	N	171	121	y
117	079	O	172	122	z
120	080	P	173	123	{
121	081	Q	174	124	¦
122	082	R	175	125	} (ALT MODE)*
123	083	S	176	126	~ (ALT MODE)*
124	084	T	177	127	DEL (RUBOUT)†

*Asterisk denotes the control function for MUMPS terminals, if different from specified or other use.
†Dagger denotes the control function for magtape devices.

APPENDIX C

EXPLANATION OF MUMPS MESSAGES

When execution of a MUMPS program is terminated by either an error, a CTRL C, or by pressing the BREAK key, the program executive outputs a short message to indicate the reason for termination. This message is followed by the number of the Step being executed and the program name unless the error occurred while in Direct Mode. The error message format is:

?message>spn␣pnam

MUMPS messages are categorized as follows:

1. MUMPS Programming Error Messages - result from errors associated with programming problems (either incorrect language syntax or semantic misunderstandings).

2. Voluntary Program Termination Message - there is only one message of this type.

3. Debugging Aid Message - indicates that a BREAK command has been encountered in the program.

4. Operating System Error Messages - result from various troubles which are detected by the operating system and which are beyond the control of the MUMPS application programmer.

MUMPS errors are considered terminal unless the user's program Sets the $E System Variable for application program control of error processing. The programmer may Set $E to a Step or Part number (S␣$E=spn) to which control will go if an error occurs (except GARB0 - GARB4 errors which are reported only on the console terminal, and do not terminate a running job). When $E is set to an spn and an error occurs, the system transfers control to the spn and resets $E to an index in the range 0 through -0.37 which indicates the type of error encountered (e.g., 0 = INRPT, -0.01 = MXNUM - see below). The number of the Step that contains the error is entered in the $W System Variable. The system also cancels all currently active DO, FOR, and CALL commands. It is the user's responsibility to reset $E to an spn if he wishes to control further error processing; otherwise, error processing reverts to system control.

If an error occurs and $E is not set by the programmer, the action taken by the system depends on the mode in which the user signed on at log-in. If the programming access code (PAC) was used, control is returned to Direct Mode after the error message is output. Otherwise, the job is aborted after typing the error and 'EXIT' messages and the terminal is automatically logged-out.

Each of the messages is explained on the pages which follow:

C.1 MUMPS PROGRAMMING ERROR MESSAGES

Message	$E Index	Meaning
CMMND	-0.15	Indicates illegal use of a command: 1. Command is undefined in the language; 2. An argument has been omitted where required.
DIVER	-0.19	Indicates an attempt to perform division by zero.
DKSER	-0.04	If not a system software error (Section C.4) this user software error indicates an attempt to: 1. use VIEW command to access a block number larger than size of the referenced disk, or a nonexistent disk; or 2. use the disk (e.g., creating global variables, issuing the FILE, LOAD, etc., commands) under a UCI that has no associated directories.
FRACT	-0.08	Indicates that a fractional number was encountered when the process being executed was expecting a integer number. Also involved when a Step number has no fractional part.
FUNCT	-0.07	Indicates that the function is undefined in the language.
LBOV	-0.14	Indicates an attempt to input or output a line greater than 132 characters.
$MERR	-0.36	Indicates that an error occurred in $M processing. 1. exponent overflow 2. exponent underflow 3. division by 0 4. illegal trap instruction (system error)
MINIM	-0.03	Indicates that a number has more than two digits following the decimal point.
MINUS	-0.12	Indicates that a negative or zero number was encountered when a positive number was expected. For example, MUMPS causes a MINUS error if the user references a subscripted variable with a negative subscript: Only positive subscripts are allowed, except when using the $HIGH function (Section 4.2.6).

Message	$E Index	Meaning
MODER	-0.23	1. An nve was encountered where an svl was expected or vice versa. 2. Argument to $TEXT is not numeric. 3. Argument to $VIEW is not numeric.
MXNUM	-0.01	Indicates that the value of a number has exceeded the integer bounds set by the MUMPS system. The maximum value for a number is ±21474836.47.
MXSTR	-0.02	Indicates that the string has exceeded maximum length allowed (132 characters).
NAKED	-0.29	Indicates that the present user attempted to reference a global variable using "naked" syntax: 1. prior to any full syntax reference; or 2. after another user KILLed the global variable.
NODEV	-0.13	Indicates an attempt to ASSIGN a nonexistent device or the use of an illegal device number.
NOPGM	-0.28	Reference is made to a program name that does not exist in the program directory for this UCI and is not in the directory of Library (%) Programs.
NOTSY	-0.34	Indicates that the referenced device or function is not in the system (it may not have been linked at system generation).
NXMEM	-0.05	Non-Existent Memory was referenced in VIEW command or $VIEW function.
PGMOV	-0.24	Indicates that there is insufficient space available in the partition. Caused by: 1. too many program steps in the program being created via the terminal or in the program being loaded; (LOAD, CALL and OVERLAY commands) 2. too many local variables; 3. expression or subscript nesting too deep.

Message	$E Index	Meaning
PROT	-0.06	Indicates that an attempt was made to use either the VIEW Command or the $VIEW Function from a non-Library (%) Program or when not logged in under the System UCI. Also indicates that the MODIFY command issued from Indirect Mode specified an spn smaller than the current spn.
SBSCR	-0.09	Indicates illegal subscript usage: - subscript out of range; - negative subscript.
SPNER	-0.17	Indicates that an illegal or nonexistent Step or Part number was used.
STKOV	-0.10	Indicates that the available stack space is used up. Generally indicates nesting is too deep in DO or CALL statements.
STKUN	-0.11	Indicates execution of the Overlay command from Direct Mode (stack underflow).
SYMOV	-0.16	Symbol Table Overflow occurred on an attempt to create or change a local variable.
SYNTX	-0.27	Indicates that the current Step being executed has an error in syntax. Syntax errors include illegal punctuation, illegal use of operators, illegal use of parentheses, as well as errors encountered in editing a Step. Syntax errors comprise a great majority of errors made in the MUMPS system and usually the user will be able to determine the exact cause of the error by merely looking at the Step concerned.
UNDEF	-0.21	Indicates a reference to an undefined local or global variable.

C.2 VOLUNTARY PROGRAM TERMINATION

Message	$E Index	Meaning
INRPT	0	Signifies interruption of program execution caused by typing CTRL C or pressing the BREAK key.

C.3 DEBUGGING AID MESSAGE

Message	$E Index	Meaning
?n BREAK	None	Indicates that program control has reached a BREAK command at Step n. BREAK commands are used to interrupt execution of the program for debugging purposes. The GO command may be typed to resume operation.

C.4 MUMPS OPERATING SYSTEM ERROR MESSAGES

Message	$E Index	Meaning
GARB0	None	Disk error while reading a data block.
GARB1	None	Disk error while writing a data block.
GARB2	None	Disk error while reading a bit map.
GARB3	None	Disk error while writing a bit map.
GARB4	None	Disk error, an attempt to deallocate a bit map or data block not yet allocated.

NOTE

The above errors are disk errors detected by the system's Garbage Collector routine. The message is output to the console terminal. GARB1 and GARB3 result in suspension of all disk I/O until system restart. Notify system manager.

Message	$E Index	Meaning
DBDGD	-0.31	Indicates a data base degradation. The system attempted to read a block that was not actually allocated. Notify system manager.
DKDER	-0.33	Indicates that a disk I/O error occurred on an attempt to write a global data buffer. The error is not given until the write is actually attempted.
DKFUL	-0.26	Indicates that there is no more room on the disk for global or program storage. Caused by SET and FILE commands. Notify system manager.
DKHER	-0.20	Indicates disk hardware error. Notify system manager.
DKSER	-0.04	In addition to conditions listed under Section C.1, this may indicate that disk block pointers in the global data base reference nonexistent or invalid disk blocks. Notify system manager.

Message	$E Index	Meaning
DSKDG	-0.18	Indicates disk degradation. Attempt was made to allocate bit map for data storage. The system corrects the bit map subsequent to this error. Notify system manager.
DTERR	-0.30	Indicates DECtape hardware or operator error. Common causes are: 1. not set to ON LINE 2. not set to WRITE ENABLE 3. unit number not selected
LPERR	-0.38	Indicates a line printer hardware error. Common causes are: 1. device off line 2. out of paper 3. yoke open 4. power off
MTERR	-0.37	Indicates magtape hardware or operator error as determined by the current contents of the $A System Variable. The system generates this error only if the user SET the $E System Variable.
PARER	None	A parity error occurred on an 11/70 when referencing an address in the partition. The job is HALTed, and that partition cannot be reused.
PLDER	-0.35	The system cannot retrieve the program being LOADed, CALLed, or STARTed. The FILE command did not complete writing the program. The user must re-FILE the back-up copy of the program.

SWAP	-0.32	Indicates
		1. that the previous swap-out overflowed the user partition stack. The error is not reported until the next swap-in.
		2. imminent system stack overflow. May be caused by faulty programming techniques, for example:
		`1.10 F I=1:1:1000 D 2`
		`2.10 D 1`

Message	$E Index	Meaning
SYSDG	-0.25	Indicates that the table in main memory which represents the bit maps on a physical disk unit (Disk Storage Allocation Table) does not correspond to the block allocation specified by the disk's bit maps. The Disk Block Tally Utility Program allows recovery from this error. Notify system manager.
SYSER	-0.22	System stack underflow on swapout. Notify system manager.

APPENDIX D

SYMBOL USAGE

The following special symbols are used by MUMPS in addition to the logical operators described in Chapter 2.

Symbol	Definition
#	Number sign is used as a format control character to initiate a Page Feed or a FORM FEED on an output device.
!	Exclamation point is used as a format control character to initiate a Carriage RETURN/LINE FEED sequence on an output device.
?	Question mark is multiply defined: 1. as an output format control character for terminals, line printer and paper-tape punch, it is followed by an nve to indicate the number of spaces to tabulate in from the absolute left margin(e.g., ?5=5 spaces from the left margin); 2. as an expression operator, it is followed by a Pattern Specification Code (psc). 3. it is the first character printed when a BREAK command or error interrupts a program's execution.
,	Comma is used as the term separator in an argument list.
␣	Space is multiply defined: 1. A command followed immediately by two spaces indicates the command has no arguments; 2. One space separates a command from its arguments, or the last argument of a preceding command from the next command on the line.
:	Colon is multiply defined: 1. a delimiter for field separation in the argument of FOR, MODIFY, and ASSIGN commands. 2. used to indicate the presence of an optional expression appended to a command or the argument of a command (where allowed).

Symbol	Definition
	3. used to indicate the presence of an optional bve appended to a command (;bve may not be appended to FOR, ELSE or IF commands). If the bve is true, the command is executed. If the bve is false, control is passed to the next command on the line or the next line (whichever is applicable). The "next command on the line" is identified by skipping to the second space following the bve. If a bve is appended to a command no argument of that command may contain a space (i.e., a string literal enclosed in quotes).
;	Semicolon is used as a delimiter to indicate that the remainder of a line is a comment.
>	Right caret is the prompting character used by MUMPS-11 when operating in Direct Mode to signal to the user that the system is ready to accept a command; that is, commands and functions may be entered for immediate execution, or program steps may be entered for program execution.
$	Dollar sign is multiply utilized. 1. precedes the first character of a System Variable. 2. precedes the first character of a function name.
%	Percent sign is used as the first character of a library program or library global name.
""	Quotation marks are used to delimit literals.
← or _	Back arrow or underscore is used to specify the indirection operation for command argument replacement.
↑ or ^	Up-arrow or up caret precedes a global variable reference.

APPENDIX E

CONVERSION TABLES

2^x IN DECIMAL

x	2^x	x	2^x	x	2^x
0.001	1.00069 33874 62581	0.01	1.00695 55500 56719	0.1	1.07177 34625 36293
0.002	1.00138 72557 11335	0.02	1.01395 94797 90029	0.2	1.14869 83549 97035
0.003	1.00208 16050 79633	0.03	1.02101 21257 07193	0.3	1.23114 44133 44916
0.004	1.00277 64359 01078	0.04	1.02811 38266 56067	0.4	1.31950 79107 72894
0.005	1.00347 17485 09503	0.05	1.03526 49238 41377	0.5	1.41421 35623 73095
0.006	1.00416 75432 38973	0.06	1.04246 57608 41121	0.6	1.51571 65665 10398
0.007	1.00486 38204 23785	0.07	1.04971 66836 23067	0.7	1.62450 47927 12471
0.008	1.00556 05803 98468	0.08	1.05701 80405 61380	0.8	1.74110 11265 92248
0.009	1.00625 78234 97782	0.09	1.06437 01824 53360	0.9	1.86606 59830 73615

$10^{\pm n}$ IN OCTAL

10^n	n	10^{-n}	10^n	n	10^{-n}
1	0	1.000 000 000 000 000 000 00	112 402 762 000	10	0.000 000 000 006 676 337 66
12	1	0.063 146 314 631 463 146 31	1 351 035 564 000	11	0.000 000 000 000 537 657 77
144	2	0.005 075 341 217 270 243 66	16 432 451 210 000	12	0.000 000 000 000 043 136 32
1 750	3	0.000 406 111 564 570 651 77	221 411 634 520 000	13	0.000 000 000 000 003 411 35
23 420	4	0.000 032 155 613 530 704 15	2 657 142 036 440 000	14	0.000 000 000 000 000 264 11
303 240	5	0.000 002 476 132 610 706 64	34 327 724 461 500 000	15	0.000 000 000 000 000 022 01
3 641 100	6	0.000 000 206 157 364 055 37	434 157 115 760 200 000	16	0.000 000 000 000 000 001 63
46 113 200	7	0.000 000 015 327 745 152 75	5 432 127 413 542 400 000	17	0.000 000 000 000 000 000 14
575 360 400	8	0.000 000 001 257 143 561 06	67 405 553 164 731 000 000	18	0.000 000 000 000 000 000 01
7 346 545 000	9	0.000 000 000 104 560 276 41			

$n \log_{10} 2$, $n \log_2 10$ IN DECIMAL

n	$n \log_{10} 2$	$n \log_2 10$	n	$n \log_{10} 2$	$n \log_2 10$
1	0.30102 99957	3.32192 80949	6	1.80617 99740	19.93156 85693
2	0.60205 99913	6.64385 61898	7	2.10720 99696	23.25349 66642
3	0.90308 99870	9.96578 42847	8	2.40823 99653	26.57542 47591
4	1.20411 99827	13.28771 23795	9	2.70926 99610	29.89735 28540
5	1.50514 99783	16.60964 04744	10	3.01029 99566	33.21928 09489

ADDITION AND MULTIPLICATION TABLES

Binary Scale

Addition:

$0 + 0 = 0$
$0 + 1 = 1 + 0 = 1$
$1 + 1 = 10$

Multiplication:

$0 \times 0 = 0$
$0 \times 1 = 1 \times 0 = 0$
$1 \times 1 = 1$

Octal Scale

Addition

0	01	02	03	04	05	06	07
1	02	03	04	05	06	07	10
2	03	04	05	06	07	10	11
3	04	05	06	07	10	11	12
4	05	06	07	10	11	12	13
5	06	07	10	11	12	13	14
6	07	10	11	12	13	14	15
7	10	11	12	13	14	15	16

Multiplication

1	02	03	04	05	06	07
2	04	06	10	12	14	16
3	06	11	14	17	22	25
4	10	14	20	24	30	34
5	12	17	24	31	36	43
6	14	22	30	36	44	52
7	16	25	34	43	52	61

MATHEMATICAL CONSTANTS IN OCTAL SCALE

$\pi = 3.11037\ 552421_8$	$e = 2.55760\ 521305_8$	$\gamma = 0.44742\ 147707_8$
$\pi^{-1} = 0.24276\ 301556_8$	$e^{-1} = 0.27426\ 530661_8$	$\ln \gamma = -0.43127\ 233602_8$
$\sqrt{\pi} = 1.61337\ 611067_8$	$\sqrt{e} = 1.51411\ 230704_8$	$\log_2 \gamma = -0.62573\ 030645_8$
$\ln \pi = 1.11206\ 404435_8$	$\log_{10} e = 0.33626\ 754251_8$	$\sqrt{2} = 1.32404\ 746320_8$
$\log_2 \pi = 1.51544\ 163223_8$	$\log_2 e = 1.34252\ 166245_8$	$\ln 2 = 0.54271\ 027760_8$
$\sqrt{10} = 3.12305\ 407267_8$	$\log_2 10 = 3.24464\ 741136_8$	$\ln 10 = 2.23273\ 067355_8$

POWERS OF TWO

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125
1 099 511 627 776	40	0.000 000 000 000 909 494 701 772 928 237 915 039 062 5
2 199 023 255 552	41	0.000 000 000 000 454 747 350 886 464 118 957 519 531 25
4 398 046 511 104	42	0.000 000 000 000 227 373 675 443 232 059 478 759 765 625
8 796 093 022 208	43	0.000 000 000 000 113 686 837 721 616 029 739 379 882 812 5
17 592 186 044 416	44	0.000 000 000 000 056 843 418 860 808 014 869 689 941 406 25
35 184 372 088 832	45	0.000 000 000 000 028 421 709 430 404 007 434 844 970 703 125
70 368 744 177 664	46	0.000 000 000 000 014 210 854 715 202 003 717 422 485 351 562 5
140 737 488 355 328	47	0.000 000 000 000 007 105 427 357 601 001 858 711 242 675 781 25
281 474 976 710 656	48	0.000 000 000 000 003 552 713 678 800 500 929 355 621 337 890 625
562 949 953 421 312	49	0.000 000 000 000 001 776 356 839 400 250 464 677 810 668 945 312 5
1 125 899 906 842 624	50	0.000 000 000 000 000 888 178 419 700 125 232 338 905 334 472 656 25
2 251 799 813 685 248	51	0.000 000 000 000 000 444 089 209 850 062 616 169 452 667 236 328 125
4 503 599 627 370 496	52	0.000 000 000 000 000 222 044 604 925 031 308 084 726 333 618 164 062 5
9 007 199 254 740 992	53	0.000 000 000 000 000 111 022 302 462 515 654 042 363 166 809 082 031 25
18 014 398 509 481 984	54	0.000 000 000 000 000 055 511 151 231 257 827 021 181 583 404 541 015 625
36 028 797 018 963 968	55	0.000 000 000 000 000 027 755 575 615 628 913 510 590 791 702 270 507 812 5
72 057 594 037 927 936	56	0.000 000 000 000 000 013 877 787 807 814 456 755 295 395 851 135 253 906 25
144 115 188 075 855 872	57	0.000 000 000 000 000 006 938 893 903 907 228 377 647 697 925 567 626 953 125
288 230 376 151 711 744	58	0.000 000 000 000 000 003 469 446 951 953 614 188 823 848 962 783 813 476 562 5
576 460 752 303 423 488	59	0.000 000 000 000 000 001 734 723 475 976 807 094 411 924 481 391 906 738 281 25
1 152 921 504 606 846 976	60	0.000 000 000 000 000 000 867 361 737 988 403 547 205 962 240 695 953 369 140 625
2 305 843 009 213 693 952	61	0.000 000 000 000 000 000 433 680 868 994 201 773 602 981 120 347 976 684 570 312 5
4 611 686 018 427 387 904	62	0.000 000 000 000 000 000 216 840 434 497 100 886 801 490 560 173 988 342 285 156 25
9 223 372 036 854 775 808	63	0.000 000 000 000 000 000 108 420 217 248 550 443 400 745 280 086 994 171 142 578 125
18 446 744 073 709 551 616	64	0.000 000 000 000 000 000 054 210 108 624 275 221 700 372 640 043 497 085 571 289 062 5
36 893 488 147 419 103 232	65	0.000 000 000 000 000 000 027 105 054 312 137 610 850 186 320 021 748 542 785 644 531 25
73 786 976 294 838 206 464	66	0.000 000 000 000 000 000 013 552 527 156 068 805 425 093 160 010 874 271 392 822 265 625
147 573 952 589 676 412 928	67	0.000 000 000 000 000 000 006 776 263 578 034 402 712 546 580 005 437 135 696 411 132 812 5
295 147 905 179 352 825 856	68	0.000 000 000 000 000 000 003 388 131 789 017 201 356 273 290 002 718 567 848 205 566 406 25
590 295 810 358 705 651 712	69	0.000 000 000 000 000 000 001 694 065 894 508 600 678 136 645 001 359 283 924 102 783 203 125
1 180 591 620 717 411 303 424	70	0.000 000 000 000 000 000 000 847 032 947 254 300 339 068 322 500 679 641 962 051 391 601 562 5
2 361 183 241 434 822 606 848	71	0.000 000 000 000 000 000 000 423 516 473 627 150 169 534 161 250 339 820 981 025 695 800 781 25
4 722 366 482 869 645 213 696	72	0.000 000 000 000 000 000 000 211 758 236 813 575 084 767 080 625 169 910 490 512 847 900 390 625

OCTAL-DECIMAL CONVERSION

OCTAL-DECIMAL INTEGER CONVERSION TABLE

0000 to 0777 (Octal) | 0000 to 0511 (Decimal)

Octal	Decimal
10000	4096
20000	8192
30000	12288
40000	16384
50000	20480
60000	24576
70000	28672

	0	1	2	3	4	5	6	7
0000	0000	0001	0002	0003	0004	0005	0006	0007
0010	0008	0009	0010	0011	0012	0013	0014	0015
0020	0016	0017	0018	0019	0020	0021	0022	0023
0030	0024	0025	0026	0027	0028	0029	0030	0031
0040	0032	0033	0034	0035	0036	0037	0038	0039
0050	0040	0041	0042	0043	0044	0045	0046	0047
0060	0048	0049	0050	0051	0052	0053	0054	0055
0070	0056	0057	0058	0059	0060	0061	0062	0063
0100	0064	0065	0066	0067	0068	0069	0070	0071
0110	0072	0073	0074	0075	0076	0077	0078	0079
0120	0080	0081	0082	0083	0084	0085	0086	0087
0130	0088	0089	0090	0091	0092	0093	0094	0095
0140	0096	0097	0098	0099	0100	0101	0102	0103
0150	0104	0105	0106	0107	0108	0109	0110	0111
0160	0112	0113	0114	0115	0116	0117	0118	0119
0170	0120	0121	0122	0123	0124	0125	0126	0127
0200	0128	0129	0130	0131	0132	0133	0134	0135
0210	0136	0137	0138	0139	0140	0141	0142	0143
0220	0144	0145	0146	0147	0148	0149	0150	0151
0230	0152	0153	0154	0155	0156	0157	0158	0159
0240	0160	0161	0162	0163	0164	0165	0166	0167
0250	0168	0169	0170	0171	0172	0173	0174	0175
0260	0176	0177	0178	0179	0180	0181	0182	0183
0270	0184	0185	0186	0187	0188	0189	0190	0191
0300	0192	0193	0194	0195	0196	0197	0198	0199
0310	0200	0201	0202	0203	0204	0205	0206	0207
0320	0208	0209	0210	0211	0212	0213	0214	0215
0330	0216	0217	0218	0219	0220	0221	0222	0223
0340	0224	0225	0226	0227	0228	0229	0230	0231
0350	0232	0233	0234	0235	0236	0237	0238	0239
0360	0240	0241	0242	0243	0244	0245	0246	0247
0370	0248	0249	0250	0251	0252	0253	0254	0255

	0	1	2	3	4	5	6	7
0400	0256	0257	0258	0259	0260	0261	0262	0263
0410	0264	0265	0266	0267	0268	0269	0270	0271
0420	0272	0273	0274	0275	0276	0277	0278	0279
0430	0280	0281	0282	0283	0284	0285	0286	0287
0440	0288	0289	0290	0291	0292	0293	0294	0295
0450	0296	0297	0298	0299	0300	0301	0302	0303
0460	0304	0305	0306	0307	0308	0309	0310	0311
0470	0312	0313	0314	0315	0316	0317	0318	0319
0500	0320	0321	0322	0323	0324	0325	0326	0327
0510	0328	0329	0330	0331	0332	0333	0334	0335
0520	0336	0337	0338	0339	0340	0341	0342	0343
0530	0344	0345	0346	0347	0348	0349	0350	0351
0540	0352	0353	0354	0355	0356	0357	0358	0359
0550	0360	0361	0362	0363	0364	0365	0366	0367
0560	0368	0369	0370	0371	0372	0373	0374	0375
0570	0376	0377	0378	0379	0380	0381	0382	0383
0600	0384	0385	0386	0387	0388	0389	0390	0391
0610	0392	0393	0394	0395	0396	0397	0398	0399
0620	0400	0401	0402	0403	0404	0405	0406	0407
0630	0408	0409	0410	0411	0412	0413	0414	0415
0640	0416	0417	0418	0419	0420	0421	0422	0423
0650	0424	0425	0426	0427	0428	0429	0430	0431
0660	0432	0433	0434	0435	0436	0437	0438	0439
0670	0440	0441	0442	0443	0444	0445	0446	0447
0700	0448	0449	0450	0451	0452	0453	0454	0455
0710	0456	0457	0458	0459	0460	0461	0462	0463
0720	0464	0465	0466	0467	0468	0469	0470	0471
0730	0472	0473	0474	0475	0476	0477	0478	0479
0740	0480	0481	0482	0483	0484	0485	0486	0487
0750	0488	0489	0490	0491	0492	0493	0494	0495
0760	0496	0497	0498	0499	0500	0501	0502	0503
0770	0504	0505	0506	0507	0508	0509	0510	0511

1000 to 1777 (Octal) | 0512 to 1023 (Decimal)

	0	1	2	3	4	5	6	7
1000	0512	0513	0514	0515	0516	0517	0518	0519
1010	0520	0521	0522	0523	0524	0525	0526	0527
1020	0528	0529	0530	0531	0532	0533	0534	0535
1030	0536	0537	0538	0539	0540	0541	0542	0543
1040	0544	0545	0546	0547	0548	0549	0550	0551
1050	0552	0553	0554	0555	0556	0557	0558	0559
1060	0560	0561	0562	0563	0564	0565	0566	0567
1070	0568	0569	0570	0571	0572	0573	0574	0575
1100	0576	0577	0578	0579	0580	0581	0582	0583
1110	0584	0585	0586	0587	0588	0589	0590	0591
1120	0592	0593	0594	0595	0596	0597	0598	0599
1130	0600	0601	0602	0603	0604	0605	0606	0607
1140	0608	0609	0610	0611	0612	0613	0614	0615
1150	0616	0617	0618	0619	0620	0621	0622	0623
1160	0624	0625	0626	0627	0628	0629	0630	0631
1170	0632	0633	0634	0635	0636	0637	0638	0639
1200	0640	0641	0642	0643	0644	0645	0646	0647
1210	0648	0649	0650	0651	0652	0653	0654	0655
1220	0656	0657	0658	0659	0660	0661	0662	0663
1230	0664	0665	0666	0667	0668	0669	0670	0671
1240	0672	0673	0674	0675	0676	0677	0678	0679
1250	0680	0681	0682	0683	0684	0685	0686	0687
1260	0688	0689	0690	0691	0692	0693	0694	0695
1270	0696	0697	0698	0699	0700	0701	0702	0703
1300	0704	0705	0706	0707	0708	0709	0710	0711
1310	0712	0713	0714	0715	0716	0717	0718	0719
1320	0720	0721	0722	0723	0724	0725	0726	0727
1330	0728	0729	0730	0731	0732	0733	0734	0735
1340	0736	0737	0738	0739	0740	0741	0742	0743
1350	0744	0745	0746	0747	0748	0749	0750	0751
1360	0752	0753	0754	0755	0756	0757	0758	0759
1370	0760	0761	0762	0763	0764	0765	0766	0767

	0	1	2	3	4	5	6	7
1400	0768	0769	0770	0771	0772	0773	0774	0775
1410	0776	0777	0778	0779	0780	0781	0782	0783
1420	0784	0785	0786	0787	0788	0789	0790	0791
1430	0792	0793	0794	0795	0796	0797	0798	0799
1440	0800	0801	0802	0803	0804	0805	0806	0807
1450	0808	0809	0810	0811	0812	0813	0814	0815
1460	0816	0817	0818	0819	0820	0821	0822	0823
1470	0824	0825	0826	0827	0828	0829	0830	0831
1500	0832	0833	0834	0835	0836	0837	0838	0839
1510	0840	0841	0842	0843	0844	0845	0846	0847
1520	0848	0849	0850	0851	0852	0853	0854	0855
1530	0856	0857	0858	0859	0860	0861	0862	0863
1540	0864	0865	0866	0867	0868	0869	0870	0871
1550	0872	0873	0874	0875	0876	0877	0878	0879
1560	0880	0881	0882	0883	0884	0885	0886	0887
1570	0888	0889	0890	0891	0892	0893	0894	0895
1600	0896	0897	0898	0899	0900	0901	0902	0903
1610	0904	0905	0906	0907	0908	0909	0910	0911
1620	0912	0913	0914	0915	0916	0917	0918	0919
1630	0920	0921	0922	0923	0924	0925	0926	0927
1640	0928	0929	0930	0931	0932	0933	0934	0935
1650	0936	0937	0938	0939	0940	0941	0942	0943
1660	0944	0945	0946	0947	0948	0949	0950	0951
1670	0952	0953	0954	0955	0956	0957	0958	0959
1700	0960	0961	0962	0963	0964	0965	0966	0967
1710	0968	0969	0970	0971	0972	0973	0974	0975
1720	0976	0977	0978	0979	0980	0981	0982	0983
1730	0984	0985	0986	0987	0988	0989	0990	0991
1740	0992	0993	0994	0995	0996	0997	0998	0999
1750	1000	1001	1002	1003	1004	1005	1006	1007
1760	1008	1009	1010	1011	1012	1013	1014	1015
1770	1016	1017	1018	1019	1020	1021	1022	1023

OCTAL-DECIMAL INTEGER CONVERSION TABLE (continued)

	0	1	2	3	4	5	6	7
2000	1024	1025	1026	1027	1028	1029	1030	1031
2010	1032	1033	1034	1035	1036	1037	1038	1039
2020	1040	1041	1042	1043	1044	1045	1046	1047
2030	1048	1049	1050	1051	1052	1053	1054	1055
2040	1056	1057	1058	1059	1060	1061	1062	1063
2050	1064	1065	1066	1067	1068	1069	1070	1071
2060	1072	1073	1074	1075	1076	1077	1078	1079
2070	1080	1081	1082	1083	1084	1085	1086	1087
2100	1088	1089	1090	1091	1092	1093	1094	1095
2110	1096	1097	1098	1099	1100	1101	1102	1103
2120	1104	1105	1106	1107	1108	1109	1110	1111
2130	1112	1113	1114	1115	1116	1117	1118	1119
2140	1120	1121	1122	1123	1124	1125	1126	1127
2150	1128	1129	1130	1131	1132	1133	1134	1135
2160	1136	1137	1138	1139	1140	1141	1142	1143
2170	1144	1145	1146	1147	1148	1149	1150	1151
2200	1152	1153	1154	1155	1156	1157	1158	1159
2210	1160	1161	1162	1163	1164	1165	1166	1167
2220	1168	1169	1170	1171	1172	1173	1174	1175
2230	1176	1177	1178	1179	1180	1181	1182	1183
2240	1184	1185	1186	1187	1188	1189	1190	1191
2250	1192	1193	1194	1195	1196	1197	1198	1199
2260	1200	1201	1202	1203	1204	1205	1206	1207
2270	1208	1209	1210	1211	1212	1213	1214	1215
2300	1216	1217	1218	1219	1220	1221	1222	1223
2310	1224	1225	1226	1227	1228	1229	1230	1231
2320	1232	1233	1234	1235	1236	1237	1238	1239
2330	1240	1241	1242	1243	1244	1245	1246	1247
2340	1248	1249	1250	1251	1252	1253	1254	1255
2350	1256	1257	1258	1259	1260	1261	1262	1263
2360	1264	1265	1266	1267	1268	1269	1270	1271
2370	1272	1273	1274	1275	1276	1277	1278	1279

	0	1	2	3	4	5	6	7
2400	1280	1281	1282	1283	1284	1285	1286	1287
2410	1288	1289	1290	1291	1292	1293	1294	1295
2420	1296	1297	1298	1299	1300	1301	1302	1303
2430	1304	1305	1306	1307	1308	1309	1310	1311
2440	1312	1313	1314	1315	1316	1317	1318	1319
2450	1320	1321	1322	1323	1324	1325	1326	1327
2460	1328	1329	1330	1331	1332	1333	1334	1335
2470	1336	1337	1338	1339	1340	1341	1342	1343
2500	1344	1345	1346	1347	1348	1349	1350	1351
2510	1352	1353	1354	1355	1356	1357	1358	1359
2520	1360	1361	1362	1363	1364	1365	1366	1367
2530	1368	1369	1370	1371	1372	1373	1374	1375
2540	1376	1377	1378	1379	1380	1381	1382	1383
2550	1384	1385	1386	1387	1388	1389	1390	1391
2560	1392	1393	1394	1395	1396	1397	1398	1399
2570	1400	1401	1402	1403	1404	1405	1406	1407
2600	1408	1409	1410	1411	1412	1413	1414	1415
2610	1416	1417	1418	1419	1420	1421	1422	1423
2620	1424	1425	1426	1427	1428	1429	1430	1431
2630	1432	1433	1434	1435	1436	1437	1438	1439
2640	1440	1441	1442	1443	1444	1445	1446	1447
2650	1448	1449	1450	1451	1452	1453	1454	1455
2660	1456	1457	1458	1459	1460	1461	1462	1463
2670	1464	1465	1466	1467	1468	1469	1470	1471
2700	1472	1473	1474	1475	1476	1477	1478	1479
2710	1480	1481	1482	1483	1484	1485	1486	1487
2720	1488	1489	1490	1491	1492	1493	1494	1495
2730	1496	1497	1498	1499	1500	1501	1502	1503
2740	1504	1505	1506	1507	1508	1509	1510	1511
2750	1512	1513	1514	1515	1516	1517	1518	1519
2760	1520	1521	1522	1523	1524	1525	1526	1527
2770	1528	1529	1530	1531	1532	1533	1534	1535

2000 to 2777 (Octal)	1024 to 1535 (Decimal)

Octal	Decimal
10000	4096
20000	8192
30000	12288
40000	16384
50000	20480
60000	24576
70000	28672

	0	1	2	3	4	5	6	7
3000	1536	1537	1538	1539	1540	1541	1542	1543
3010	1544	1545	1546	1547	1548	1549	1550	1551
3020	1552	1553	1554	1555	1556	1557	1558	1559
3030	1560	1561	1562	1563	1564	1565	1566	1567
3040	1568	1569	1570	1571	1572	1573	1574	1575
3050	1576	1577	1578	1579	1580	1581	1582	1583
3060	1584	1585	1586	1587	1588	1589	1590	1591
3070	1592	1593	1594	1595	1596	1597	1598	1599
3100	1600	1601	1602	1603	1604	1605	1606	1607
3110	1608	1609	1610	1611	1612	1613	1614	1615
3120	1616	1617	1618	1619	1620	1621	1622	1623
3130	1624	1625	1626	1627	1628	1629	1630	1631
3140	1632	1633	1634	1635	1636	1637	1638	1639
3150	1640	1641	1642	1643	1644	1645	1646	1647
3160	1648	1649	1650	1651	1652	1653	1654	1655
3170	1656	1657	1658	1659	1660	1661	1662	1663
3200	1664	1665	1666	1667	1668	1669	1670	1671
3210	1672	1673	1674	1675	1676	1677	1678	1679
3220	1680	1681	1682	1683	1684	1685	1686	1687
3230	1688	1689	1690	1691	1692	1693	1694	1695
3240	1696	1697	1698	1699	1700	1701	1702	1703
3250	1704	1705	1706	1707	1708	1709	1710	1711
3260	1712	1713	1714	1715	1716	1717	1718	1719
3270	1720	1721	1722	1723	1724	1725	1726	1727
3300	1728	1729	1730	1731	1732	1733	1734	1735
3310	1736	1737	1738	1739	1740	1741	1742	1743
3320	1744	1745	1746	1747	1748	1749	1750	1751
3330	1752	1753	1754	1755	1756	1757	1758	1759
3340	1760	1761	1762	1763	1764	1765	1766	1767
3350	1768	1769	1770	1771	1772	1773	1774	1775
3360	1776	1777	1778	1779	1780	1781	1782	1783
3370	1784	1785	1786	1787	1788	1789	1790	1791

	0	1	2	3	4	5	6	7
3400	1792	1793	1794	1795	1796	1797	1798	1799
3410	1800	1801	1802	1803	1804	1805	1806	1807
3420	1808	1809	1810	1811	1812	1813	1814	1815
3430	1816	1817	1818	1819	1820	1821	1822	1823
3440	1824	1825	1826	1827	1828	1829	1830	1831
3450	1832	1833	1834	1835	1836	1837	1838	1839
3460	1840	1841	1842	1843	1844	1845	1846	1847
3470	1848	1849	1850	1851	1852	1853	1854	1855
3500	1856	1857	1858	1859	1860	1861	1862	1863
3510	1864	1865	1866	1867	1868	1869	1870	1871
3520	1872	1873	1874	1875	1876	1877	1878	1879
3530	1880	1881	1882	1883	1884	1885	1886	1887
3540	1888	1889	1890	1891	1892	1893	1894	1895
3550	1896	1897	1898	1899	1900	1901	1902	1903
3560	1904	1905	1906	1907	1908	1909	1910	1911
3570	1912	1913	1914	1915	1916	1917	1918	1919
3600	1920	1921	1922	1923	1924	1925	1926	1927
3610	1928	1929	1930	1931	1932	1933	1934	1935
3620	1936	1937	1938	1939	1940	1941	1942	1943
3630	1944	1945	1946	1947	1948	1949	1950	1951
3640	1952	1953	1954	1955	1956	1957	1958	1959
3650	1960	1961	1962	1963	1964	1965	1966	1967
3660	1968	1969	1970	1971	1972	1973	1974	1975
3670	1976	1977	1978	1979	1980	1981	1982	1983
3700	1984	1985	1986	1987	1988	1989	1990	1991
3710	1992	1993	1994	1995	1996	1997	1998	1999
3720	2000	2001	2002	2003	2004	2005	2006	2007
3730	2008	2009	2010	2011	2012	2013	2014	2015
3740	2016	2017	2018	2019	2020	2021	2022	2023
3750	2024	2025	2026	2027	2028	2029	2030	2031
3760	2032	2033	2034	2035	2036	2037	2038	2039
3770	2040	2041	2042	2043	2044	2045	2046	2047

3000 to 3777 (Octal)	1536 to 2047 (Decimal)

OCTAL-DECIMAL INTEGER CONVERSION TABLE (continued)

4000 to 4777 (Octal) | 2048 to 2559 (Decimal)

Octal	Decimal
10000 -	4096
20000 -	8192
30000 -	12288
40000 -	16384
50000 -	20480
60000 -	24576
70000 -	28672

	0	1	2	3	4	5	6	7
4000	2048	2049	2050	2051	2052	2053	2054	2055
4010	2056	2057	2058	2059	2060	2061	2062	2063
4020	2064	2065	2066	2067	2068	2069	2070	2071
4030	2072	2073	2074	2075	2076	2077	2078	2079
4040	2080	2081	2082	2083	2084	2085	2086	2087
4050	2088	2089	2090	2091	2092	2093	2094	2095
4060	2096	2097	2098	2099	2100	2101	2102	2103
4070	2104	2105	2106	2107	2108	2109	2110	2111
4100	2112	2113	2114	2115	2116	2117	2118	2119
4110	2120	2121	2122	2123	2124	2125	2126	2127
4120	2128	2129	2130	2131	2132	2133	2134	2135
4130	2136	2137	2138	2139	2140	2141	2142	2143
4140	2144	2145	2146	2147	2148	2149	2150	2151
4150	2152	2153	2154	2155	2156	2157	2158	2159
4160	2160	2161	2162	2163	2164	2165	2166	2167
4170	2168	2169	2170	2171	2172	2173	2174	2175
4200	2176	2177	2178	2179	2180	2181	2182	2183
4210	2184	2185	2186	2187	2188	2189	2190	2191
4220	2192	2193	2194	2195	2196	2197	2198	2199
4230	2200	2201	2202	2203	2204	2205	2206	2207
4240	2208	2209	2210	2211	2212	2213	2214	2215
4250	2216	2217	2218	2219	2220	2221	2222	2223
4260	2224	2225	2226	2227	2228	2229	2230	2231
4270	2232	2233	2234	2235	2236	2237	2238	2239
4300	2240	2241	2242	2243	2244	2245	2246	2247
4310	2248	2249	2250	2251	2252	2253	2254	2255
4320	2256	2257	2258	2259	2260	2261	2262	2263
4330	2264	2265	2266	2267	2268	2269	2270	2271
4340	2272	2273	2274	2275	2276	2277	2278	2279
4350	2280	2281	2282	2283	2284	2285	2286	2287
4360	2288	2289	2290	2291	2292	2293	2294	2295
4370	2296	2297	2298	2299	2300	2301	2302	2303

	0	1	2	3	4	5	6	7
4400	2304	2305	2306	2307	2308	2309	2310	2311
4410	2312	2313	2314	2315	2316	2317	2318	2319
4420	2320	2321	2322	2323	2324	2325	2326	2327
4430	2328	2329	2330	2331	2332	2333	2334	2335
4440	2336	2337	2338	2339	2340	2341	2342	2343
4450	2344	2345	2346	2347	2348	2349	2350	2351
4460	2352	2353	2354	2355	2356	2357	2358	2359
4470	2360	2361	2362	2363	2364	2365	2366	2367
4500	2368	2369	2370	2371	2372	2373	2374	2375
4510	2376	2377	2378	2379	2380	2381	2382	2383
4520	2384	2385	2386	2387	2388	2389	2390	2391
4530	2392	2393	2394	2395	2396	2397	2398	2399
4540	2400	2401	2402	2403	2404	2405	2406	2407
4550	2408	2409	2410	2411	2412	2413	2414	2415
4560	2416	2417	2418	2419	2420	2421	2422	2423
4570	2424	2425	2426	2427	2428	2429	2430	2431
4600	2432	2433	2434	2435	2436	2437	2438	2439
4610	2440	2441	2442	2443	2444	2445	2446	2447
4620	2448	2449	2450	2451	2452	2453	2454	2455
4630	2456	2457	2458	2459	2460	2461	2462	2463
4640	2464	2465	2466	2467	2468	2469	2470	2471
4650	2472	2473	2474	2475	2476	2477	2478	2479
4660	2480	2481	2482	2483	2484	2485	2486	2487
4670	2488	2489	2490	2491	2492	2493	2494	2495
4700	2496	2497	2498	2499	2500	2501	2502	2503
4710	2504	2505	2506	2507	2508	2509	2510	2511
4720	2512	2513	2514	2515	2516	2517	2518	2519
4730	2520	2521	2522	2523	2524	2525	2526	2527
4740	2528	2529	2530	2531	2532	2533	2534	2535
4750	2536	2537	2538	2539	2540	2541	2542	2543
4760	2544	2545	2546	2547	2548	2549	2550	2551
4770	2552	2553	2554	2555	2556	2557	2558	2559

5000 to 5777 (Octal) | 2560 to 3071 (Decimal)

	0	1	2	3	4	5	6	7
5000	2560	2561	2562	2563	2564	2565	2566	2567
5010	2568	2569	2570	2571	2572	2573	2574	2575
5020	2576	2577	2578	2579	2580	2581	2582	2583
5030	2584	2585	2586	2587	2588	2589	2590	2591
5040	2592	2593	2594	2595	2596	2597	2598	2599
5050	2600	2601	2602	2603	2604	2605	2606	2607
5060	2608	2609	2610	2611	2612	2613	2614	2615
5070	2616	2617	2618	2619	2620	2621	2622	2623
5100	2624	2625	2626	2627	2628	2629	2630	2631
5110	2632	2633	2634	2635	2636	2637	2638	2639
5120	2640	2641	2642	2643	2644	2645	2646	2647
5130	2648	2649	2650	2651	2652	2653	2654	2655
5140	2656	2657	2658	2659	2660	2661	2662	2663
5150	2664	2665	2666	2667	2668	2669	2670	2671
5160	2672	2673	2674	2675	2676	2677	2678	2679
5170	2680	2681	2682	2683	2684	2685	2686	2687
5200	2688	2689	2690	2691	2692	2693	2694	2695
5210	2696	2697	2698	2699	2700	2701	2702	2703
5220	2704	2705	2706	2707	2708	2709	2710	2711
5230	2712	2713	2714	2715	2716	2717	2718	2719
5240	2720	2721	2722	2723	2724	2725	2726	2727
5250	2728	2729	2730	2731	2732	2733	2734	2735
5260	2736	2737	2738	2739	2740	2741	2742	2743
5270	2744	2745	2746	2747	2748	2749	2750	2751
5300	2752	2753	2754	2755	2756	2757	2758	2759
5310	2760	2761	2762	2763	2764	2765	2766	2767
5320	2768	2769	2770	2771	2772	2773	2774	2775
5330	2776	2777	2778	2779	2780	2781	2782	2783
5340	2784	2785	2786	2787	2788	2789	2790	2791
5350	2792	2793	2794	2795	2796	2797	2798	2799
5360	2800	2801	2802	2803	2804	2805	2806	2807
5370	2808	2809	2810	2811	2812	2813	2814	2815

	0	1	2	3	4	5	6	7
5400	2816	2817	2818	2819	2820	2821	2822	2823
5410	2824	2825	2826	2827	2828	2829	2830	2831
5420	2832	2833	2834	2835	2836	2837	2838	2839
5430	2840	2841	2842	2843	2844	2845	2846	2847
5440	2848	2849	2850	2851	2852	2853	2854	2855
5450	2856	2857	2858	2859	2860	2861	2862	2863
5460	2864	2865	2866	2867	2868	2869	2870	2871
5470	2872	2873	2874	2875	2876	2877	2878	2879
5500	2880	2881	2882	2883	2884	2885	2886	2887
5510	2888	2889	2890	2891	2892	2893	2894	2895
5520	2896	2897	2898	2899	2900	2901	2902	2903
5530	2904	2905	2906	2907	2908	2909	2910	2911
5540	2912	2913	2914	2915	2916	2917	2918	2919
5550	2920	2921	2922	2923	2924	2925	2926	2927
5560	2928	2929	2930	2931	2932	2933	2934	2935
5570	2936	2937	2938	2939	2940	2941	2942	2943
5600	2944	2945	2946	2947	2948	2949	2950	2951
5610	2952	2953	2954	2955	2956	2957	2958	2959
5620	2960	2961	2962	2963	2964	2965	2966	2967
5630	2968	2969	2970	2971	2972	2973	2974	2975
5640	2976	2977	2978	2979	2980	2981	2982	2983
5650	2984	2985	2986	2987	2988	2989	2990	2991
5660	2992	2993	2994	2995	2996	2997	2998	2999
5670	3000	3001	3002	3003	3004	3005	3006	3007
5700	3008	3009	3010	3011	3012	3013	3014	3015
5710	3016	3017	3018	3019	3020	3021	3022	3023
5720	3024	3025	3026	3027	3028	3029	3030	3031
5730	3032	3033	3034	3035	3036	3037	3038	3039
5740	3040	3041	3042	3043	3044	3045	3046	3047
5750	3048	3049	3050	3051	3052	3053	3054	3055
5760	3056	3057	3058	3059	3060	3061	3062	3063
5770	3064	3065	3066	3067	3068	3069	3070	3071

OCTAL-DECIMAL INTEGER CONVERSION TABLE (continued)

	0	1	2	3	4	5	6	7
6000	3072	3073	3074	3075	3076	3077	3078	3079
6010	3080	3081	3082	3083	3084	3085	3086	3087
6020	3088	3089	3090	3091	3092	3093	3094	3095
6030	3096	3097	3098	3099	3100	3101	3102	3103
6040	3104	3105	3106	3107	3108	3109	3110	3111
6050	3112	3113	3114	3115	3116	3117	3118	3119
6060	3120	3121	3122	3123	3124	3125	3126	3127
6070	3128	3129	3130	3131	3132	3133	3134	3135
6100	3136	3137	3138	3139	3140	3141	3142	3143
6110	3144	3145	3146	3147	3148	3149	3150	3151
6120	3152	3153	3154	3155	3156	3157	3158	3159
6130	3160	3161	3162	3163	3164	3165	3166	3167
6140	3168	3169	3170	3171	3172	3173	3174	3175
6150	3176	3177	3178	3179	3180	3181	3182	3183
6160	3184	3185	3186	3187	3188	3189	3190	3191
6170	3192	3193	3194	3195	3196	3197	3198	3199
6200	3200	3201	3202	3203	3204	3205	3206	3207
6210	3208	3209	3210	3211	3212	3213	3214	3215
6220	3216	3217	3218	3219	3220	3221	3222	3223
6230	3224	3225	3226	3227	3228	3229	3230	3231
6240	3232	3233	3234	3235	3236	3237	3238	3239
6250	3240	3241	3242	3243	3244	3245	3246	3247
6260	3248	3249	3250	3251	3252	3253	3254	3255
6270	3256	3257	3258	3259	3260	3261	3262	3263
6300	3264	3265	3266	3267	3268	3269	3270	3271
6310	3272	3273	3274	3275	3276	3277	3278	3279
6320	3280	3281	3282	3283	3284	3285	3286	3287
6330	3288	3289	3290	3291	3292	3293	3294	3295
6340	3296	3297	3298	3299	3300	3301	3302	3303
6350	3304	3305	3306	3307	3308	3309	3310	3311
6360	3312	3313	3314	3315	3316	3317	3318	3319
6370	3320	3321	3322	3323	3324	3325	3326	3327

	0	1	2	3	4	5	6	7
6400	3328	3329	3330	3331	3332	3333	3334	3335
6410	3336	3337	3338	3339	3340	3341	3342	3343
6420	3344	3345	3346	3347	3348	3349	3350	3351
6430	3352	3353	3354	3355	3356	3357	3358	3359
6440	3360	3361	3362	3363	3364	3365	3366	3367
6450	3368	3369	3370	3371	3372	3373	3374	3375
6460	3376	3377	3378	3379	3380	3381	3382	3383
6470	3384	3385	3386	3387	3388	3389	3390	3391
6500	3392	3393	3394	3395	3396	3397	3398	3399
6510	3400	3401	3402	3403	3404	3405	3406	3407
6520	3408	3409	3410	3411	3412	3413	3414	3415
6530	3416	3417	3418	3419	3420	3421	3422	3423
6540	3424	3425	3426	3427	3428	3429	3430	3431
6550	3432	3433	3434	3435	3436	3437	3438	3439
6560	3440	3441	3442	3443	3444	3445	3446	3447
6570	3448	3449	3450	3451	3452	3453	3454	3455
6600	3456	3457	3458	3459	3460	3461	3462	3463
6610	3464	3465	3466	3467	3468	3469	3470	3471
6620	3472	3473	3474	3475	3476	3477	3478	3479
6630	3480	3481	3482	3483	3484	3485	3486	3487
6640	3488	3489	3490	3491	3492	3493	3494	3495
6650	3496	3497	3498	3499	3500	3501	3502	3503
6660	3504	3505	3506	3507	3508	3509	3510	3511
6670	3512	3513	3514	3515	3516	3517	3518	3519
6700	3520	3521	3522	3523	3524	3525	3526	3527
6710	3528	3529	3530	3531	3532	3533	3534	3535
6720	3536	3537	3538	3539	3540	3541	3542	3543
6730	3544	3545	3546	3547	3548	3549	3550	3551
6740	3552	3553	3554	3555	3556	3557	3558	3559
6750	3560	3561	3562	3563	3564	3565	3566	3567
6760	3568	3569	3570	3571	3572	3573	3574	3575
6770	3576	3577	3578	3579	3580	3581	3582	3583

6000 to 6777 (Octal) | 3072 to 3583 (Decimal)

Octal	Decimal
10000	4096
20000	8192
30000	12288
40000	16384
50000	20480
60000	24576
70000	28672

	0	1	2	3	4	5	6	7
7000	3584	3585	3586	3587	3588	3589	3590	3591
7010	3592	3593	3594	3595	3596	3597	3598	3599
7020	3600	3601	3602	3603	3604	3605	3606	3607
7030	3608	3609	3610	3611	3612	3613	3614	3615
7040	3616	3617	3618	3619	3620	3621	3622	3623
7050	3624	3625	3626	3627	3628	3629	3630	3631
7060	3632	3633	3634	3635	3636	3637	3638	3639
7070	3640	3641	3642	3643	3644	3645	3646	3647
7100	3648	3649	3650	3651	3652	3653	3654	3655
7110	3656	3657	3658	3659	3660	3661	3662	3663
7120	3664	3665	3666	3667	3668	3669	3670	3671
7130	3672	3673	3674	3675	3676	3677	3678	3679
7140	3680	3681	3682	3683	3684	3685	3686	3687
7150	3688	3689	3690	3691	3692	3693	3694	3695
7160	3696	3697	3698	3699	3700	3701	3702	3703
7170	3704	3705	3706	3707	3708	3709	3710	3711
7200	3712	3713	3714	3715	3716	3717	3718	3719
7210	3720	3721	3722	3723	3724	3725	3726	3727
7220	3728	3729	3730	3731	3732	3733	3734	3735
7230	3736	3737	3738	3739	3740	3741	3742	3743
7240	3744	3745	3746	3747	3748	3749	3750	3751
7250	3752	3753	3754	3755	3756	3757	3758	3759
7260	3760	3761	3762	3763	3764	3765	3766	3767
7270	3768	3769	3770	3771	3772	3773	3774	3775
7300	3776	3777	3778	3779	3780	3781	3782	3783
7310	3784	3785	3786	3787	3788	3789	3790	3791
7320	3792	3793	3794	3795	3796	3797	3798	3799
7330	3800	3801	3802	3803	3804	3805	3806	3807
7340	3808	3809	3810	3811	3812	3813	3814	3815
7350	3816	3817	3818	3819	3820	3821	3822	3823
7360	3824	3825	3826	3827	3828	3829	3830	3831
7370	3832	3833	3834	3835	3836	3837	3838	3839

	0	1	2	3	4	5	6	7
7400	3840	3841	3842	3843	3844	3845	3846	3847
7410	3848	3849	3850	3851	3852	3853	3854	3855
7420	3856	3857	3858	3859	3860	3861	3862	3863
7430	3864	3865	3866	3867	3868	3869	3870	3871
7440	3872	3873	3874	3875	3876	3877	3878	3879
7450	3880	3881	3882	3883	3884	3885	3886	3887
7460	3888	3889	3890	3891	3892	3893	3894	3895
7470	3896	3897	3898	3899	3900	3901	3902	3903
7500	3904	3905	3906	3907	3908	3909	3910	3911
7510	3912	3913	3914	3915	3916	3917	3918	3919
7520	3920	3921	3922	3923	3924	3925	3926	3927
7530	3928	3929	3930	3931	3932	3933	3934	3935
7540	3936	3937	3938	3939	3940	3941	3942	3943
7550	3944	3945	3946	3947	3948	3949	3950	3951
7560	3952	3953	3954	3955	3956	3957	3958	3959
7570	3960	3961	3962	3963	3964	3965	3966	3967
7600	3968	3969	3970	3971	3972	3973	3974	3975
7610	3976	3977	3978	3979	3980	3981	3982	3983
7620	3984	3985	3986	3987	3988	3989	3990	3991
7630	3992	3993	3994	3995	3996	3997	3998	3999
7640	4000	4001	4002	4003	4004	4005	4006	4007
7650	4008	4009	4010	4011	4012	4013	4014	4015
7660	4016	4017	4018	4019	4020	4021	1022	4023
7670	4024	4025	4026	4027	4028	4029	4030	4031
7700	4032	4033	4034	4035	4036	4037	4038	4039
7710	4040	4041	4042	4043	4044	4045	4046	4047
7720	4048	4049	4050	4051	4052	4053	4054	4055
7730	4056	4057	4058	4059	4060	4061	4062	4063
7740	4064	4065	4066	4067	4068	4069	4070	4071
7750	4072	4073	4074	4075	4076	4077	4078	4079
7760	4080	4081	4082	4083	4084	4085	4086	4087
7770	4088	4089	4090	4091	4092	4093	4094	4095

7000 to 7777 (Octal) | 3584 to 4095 (Decimal)

OCTAL-DECIMAL FRACTION CONVERSION TABLE

Octal	Decimal	Octal	Decimal	Octal	Decimal	Octal	Decimal
.000	.000000	.100	.125000	.200	.250000	.300	.375000
.001	.001953	.101	.126953	.201	.251953	.301	.376953
.002	.003906	.102	.128906	.202	.253906	.302	.378906
.003	.005859	.103	.130859	.203	.255859	.303	.380859
.004	.007812	.104	.132812	.204	.257812	.304	.382812
.005	.009765	.105	.134765	.205	.259765	.305	.384765
.006	.011718	.106	.136718	.206	.261718	.306	.386718
.007	.013671	.107	.138671	.207	.263671	.307	.388671
.010	.015625	.110	.140625	.210	.265625	.310	.390625
.011	.017578	.111	.142578	.211	.267578	.311	.392578
.012	.019531	.112	.144531	.212	.269531	.312	.394531
.013	.021484	.113	.146484	.213	.271484	.313	.396484
.014	.023437	.114	.148437	.214	.273437	.314	.398437
.015	.025390	.115	.150390	.215	.275390	.315	.400390
.016	.027343	.116	.152343	.216	.277343	.316	.402343
.017	.029296	.117	.154296	.217	.279296	.317	.404296
.020	.031250	.120	.156250	.220	.281250	.320	.406250
.021	.033203	.121	.158203	.221	.283203	.321	.408203
.022	.035156	.122	.160156	.222	.285156	.322	.410156
.023	.037109	.123	.162109	.223	.287109	.323	.412109
.024	.039062	.124	.164062	.224	.289062	.324	.414062
.025	.041015	.125	.166015	.225	.291015	.325	.416015
.026	.042968	.126	.167968	.226	.292968	.326	.417968
.027	.044921	.127	.169921	.227	.294921	.327	.419921
.030	.046875	.130	.171875	.230	.296875	.330	.421875
.031	.048828	.131	.173828	.231	.298828	.331	.423828
.032	.050781	.132	.175781	.232	.300781	.332	.425781
.033	.052734	.133	.177734	.233	.302734	.333	.427734
.034	.054687	.134	.179687	.234	.304687	.334	.429687
.035	.056640	.135	.181640	.235	.306640	.335	.431640
.036	.058593	.136	.183593	.236	.308593	.336	.433593
.037	.060546	.137	.185546	.237	.310546	.337	.435546
.040	.062500	.140	.187500	.240	.312500	.340	.437500
.041	.064453	.141	.189453	.241	.314453	.341	.439453
.042	.066406	.142	.191406	.242	.316406	.342	.441406
.043	.068359	.143	.193359	.243	.318359	.343	.443359
.044	.070312	.144	.195312	.244	.320312	.344	.445312
.045	.072265	.145	.197265	.245	.322265	.345	.447265
.046	.074218	.146	.199218	.246	.324218	.346	.449218
.047	.076171	.147	.201171	.247	.326171	.347	.451171
.050	.078125	.150	.203125	.250	.328125	.350	.453125
.051	.080078	.151	.205078	.251	.330078	.351	.455078
.052	.082031	.152	.207031	.252	.332031	.352	.457031
.053	.083984	.153	.208984	.253	.333984	.353	.458984
.054	.085937	.154	.210937	.254	.335937	.354	.460937
.055	.087890	.155	.212890	.255	.337890	.355	.462890
.056	.089843	.156	.214843	.256	.339843	.356	.464843
.057	.091796	.157	.216796	.257	.341796	.357	.466796
.060	.093750	.160	.218750	.260	.343750	.360	.468750
.061	.095703	.161	.220703	.261	.345703	.361	.470703
.062	.097656	.162	.222656	.262	.347656	.362	.472656
.063	.099609	.163	.224609	.263	.349609	.363	.474609
.064	.101562	.164	.226562	.264	.351562	.364	.476562
.065	.103515	.165	.228515	.265	.353515	.365	.478515
.066	.105468	.166	.230468	.266	.355468	.366	.480468
.067	.107421	.167	.232421	.267	.357421	.367	.482421
.070	.109375	.170	.234375	.270	.359375	.370	.484375
.071	.111328	.171	.236328	.271	.361328	.371	.486328
.072	.113281	.172	.238281	.272	.363281	.372	.488281
.073	.115234	.173	.240234	.273	.365234	.373	.490234
.074	.117187	.174	.242187	.274	.367187	.374	.492187
.075	.119140	.175	.244140	.275	.369140	.375	.494140
.076	.121093	.176	.246093	.276	.371093	.376	.496093
.077	.123046	.177	.248046	.277	.373046	.377	.498046

OCTAL-DECIMAL FRACTION CONVERSION TABLE (continued)

Octal	Decimal	Octal	Decimal	Octal	Decimal	Octal	Decimal
.000000	.000000	.000100	.000244	.000200	.000488	.000300	.000732
.000001	.000003	.000101	.000247	.000201	.000492	.000301	.000736
.000002	.000007	.000102	.000251	.000202	.000495	.000302	.000740
.000003	.000011	.000103	.000255	.000203	.000499	.000303	.000743
.000004	.000015	.000104	.000259	.000204	.000503	.000304	.000747
.000005	.000019	.000105	.000263	.000205	.000507	.000305	.000751
.000006	.000022	.000106	.000267	.000206	.000511	.000306	.000755
.000007	.000026	.000107	.000270	.000207	.000514	.000307	.000759
.000010	.000030	.000110	.000274	.000210	.000518	.000310	.000762
.000011	.000034	.000111	.000278	.000211	.000522	.000311	.000766
.000012	.000038	.000112	.000282	.000212	.000526	.000312	.000770
.000013	.000041	.000113	.000286	.000213	.000530	.000313	.000774
.000014	.000045	.000114	.000289	.000214	.000534	.000314	.000778
.000015	.000049	.000115	.000293	.000215	.000537	.000315	.000782
.000016	.000053	.000116	.000297	.000216	.000541	.000316	.000785
.000017	.000057	.000117	.000301	.000217	.000545	.000317	.000789
.000020	.000061	.000120	.000305	.000220	.000549	.000320	.000793
.000021	.000064	.000121	.000308	.000221	.000553	.000321	.000797
.000022	.000068	.000122	.000312	.000222	.000556	.000322	.000801
.000023	.000072	.000123	.000316	.000223	.000560	.000323	.000805
.000024	.000076	.000124	.000320	.000224	.000564	.000324	.000808
.000025	.000080	.000125	.000324	.000225	.000568	.000325	.000812
.000026	.000083	.000126	.000328	.000226	.000572	.000326	.000816
.000027	.000087	.000127	.000331	.000227	.000576	.000327	.000820
.000030	.000091	.000130	.000335	.000230	.000579	.000330	.000823
.000031	.000095	.000131	.000339	.000231	.000583	.000331	.000827
.000032	.000099	.000132	.000343	.000232	.000587	.000332	.000831
.000033	.000102	.000133	.000347	.000233	.000591	.000333	.000835
.000034	.000106	.000134	.000350	.000234	.000595	.000334	.000839
.000035	.000110	.000135	.000354	.000235	.000598	.000335	.000843
.000036	.000114	.000136	.000358	.000236	.000602	.000336	.000846
.000037	.000118	.000137	.000362	.000237	.000606	.000337	.000850
.000040	.000122	.000140	.000366	.000240	.000610	.000340	.000854
.000041	.000125	.000141	.000370	.000241	.000614	.000341	.000858
.000042	.000129	.000142	.000373	.000242	.000617	.000342	.000862
.000043	.000133	.000143	.000377	.000243	.000621	.000343	.000865
.000044	.000137	.000144	.000381	.000244	.000625	.000344	.000869
.000045	.000141	.000145	.000385	.000245	.000629	.000345	.000873
.000046	.000144	.000146	.000389	.000246	.000633	.000346	.000877
.000047	.000148	.000147	.000392	.000247	.000637	.000347	.000881
.000050	.000152	.000150	.000396	.000250	.000640	.000350	.000885
.000051	.000156	.000151	.000400	.000251	.000644	.000351	.000888
.000052	.000160	.000152	.000404	.000252	.000648	.000352	.000892
.000053	.000164	.000153	.000408	.000253	.000652	.000353	.000896
.000054	.000167	.000154	.000411	.000254	.000656	.000354	.000900
.000055	.000171	.000155	.000415	.000255	.000659	.000355	.000904
.000056	.000175	.000156	.000419	.000256	.000663	.000356	.000907
.000057	.000179	.000157	.000423	.000257	.000667	.000357	.000911
.000060	.000183	.000160	.000427	.000260	.000671	.000360	.000915
.000061	.000186	.000161	.000431	.000261	.000675	.000361	.000919
.000062	.000190	.000162	.000434	.000262	.000679	.000362	.000923
.000063	.000194	.000163	.000438	.000263	.000682	.000363	.000926
.000064	.000198	.000164	.000442	.000264	.000686	.000364	.000930
.000065	.000202	.000165	.000446	.000265	.000690	.000365	.000934
.000066	.000205	.000166	.000450	.000266	.000694	.000366	.000938
.000067	.000209	.000167	.000453	.000267	.000698	.000367	.000942
.000070	.000213	.000170	.000457	.000270	.000701	.000370	.000946
.000071	.000217	.000171	.000461	.000271	.000705	.000371	.000949
.000072	.000221	.000172	.000465	.000272	.000709	.000372	.000953
.000073	.000225	.000173	.000469	.000273	.000713	.000373	.000957
.000074	.000228	.000174	.000473	.000274	.000717	.000374	.000961
.000075	.000232	.000175	.000476	.000275	.000720	.000375	.000965
.000076	.000236	.000176	.000480	.000276	.000724	.000376	.000968
.000077	.000240	.000177	.000484	.000277	.000728	.000377	.000972

OCTAL-DECIMAL FRACTION CONVERSION TABLE (continued)

Octal	Decimal	Octal	Decimal	Octal	Decimal	Octal	Decimal
.000400	.000976	.000500	.001220	.000600	.001464	.000700	.001708
.000401	.000980	.000501	.001224	.000601	.001468	.000701	.001712
.000402	.000984	.000502	.001228	.000602	.001472	.000702	.001716
.000403	.000988	.000503	.001232	.000603	.001476	.000703	.001720
.000404	.000991	.000504	.001235	.000604	.001480	.000704	.001724
.000405	.000995	.000505	.001239	.000605	.001483	.000705	.001728
.000406	.000999	.000506	.001243	.000606	.001487	.000706	.001731
.000407	.001003	.000507	.001247	.000607	.001491	.000707	.001735
.000410	.001007	.000510	.001251	.000610	.001495	.000710	.001739
.000411	.001010	.000511	.001255	.000611	.001499	.000711	.001743
.000412	.001014	.000512	.001258	.000612	.001502	.000712	.001747
.000413	.001018	.000513	.001262	.000613	.001506	.000713	.001750
.000414	.001022	.000514	.001266	.000614	.001510	.000714	.001754
.000415	.001026	.000515	.001270	.000615	.001514	.000715	.001758
.000416	.001029	.000516	.001274	.000616	.001518	.000716	.001762
.000417	.001033	.000517	.001277	.000617	.001522	.000717	.001766
.000420	.001037	.000520	.001281	.000620	.001525	.000720	.001770
.000421	.001041	.000521	.001285	.000621	.001529	.000721	.001773
.000422	.001045	.000522	.001289	.000622	.001533	.000722	.001777
.000423	.001049	.000523	.001293	.000623	.001537	.000723	.001781
.000424	.001052	.000524	.001296	.000624	.001541	.000724	.001785
.000425	.001056	.000525	.001300	.000625	.001544	.000725	.001789
.000426	.001060	.000526	.001304	.000626	.001548	.000726	.001792
.000427	.001064	.000527	.001308	.000627	.001552	.000727	.001796
.000430	.001068	.000530	.001312	.000630	.001556	.000730	.001800
.000431	.001071	.000531	.001316	.000631	.001560	.000731	.001804
.000432	.001075	.000532	.001319	.000632	.001564	.000732	.001808
.000433	.001079	.000533	.001323	.000633	.001567	.000733	.001811
.000434	.001083	.000534	.001327	.000634	.001571	.000734	.001815
.000435	.001087	.000535	.001331	.000635	.001575	.000735	.001819
.000436	.001091	.000536	.001335	.000636	.001579	.000736	.001823
.000437	.001094	.000537	.001338	.000637	.001583	.000737	.001827
.000440	.001098	.000540	.001342	.000640	.001586	.000740	.001831
.000441	.001102	.000541	.001346	.000641	.001590	.000741	.001834
.000442	.001106	.000542	.001350	.000642	.001594	.000742	.001838
.000443	.001110	.000543	.001354	.000643	.001598	.000743	.001842
.000444	.001113	.000544	.001358	.000644	.001602	.000744	.001846
.000445	.001117	.000545	.001361	.000645	.001605	.000745	.001850
.000446	.001121	.000546	.001365	.000646	.001609	.000746	.001853
.000447	.001125	.000547	.001369	.000647	.001613	.000747	.001857
.000450	.001129	.000550	.001373	.000650	.001617	.000750	.001861
.000451	.001132	.000551	.001377	.000651	.001621	.000751	.001865
.000452	.001136	.000552	.001380	.000652	.001625	.000752	.001869
.000453	.001140	.000553	.001384	.000653	.001628	.000753	.001873
.000454	.001144	.000554	.001388	.000654	.001632	.000754	.001876
.000455	.001148	.000555	.001392	.000655	.001636	.000755	.001880
.000456	.001152	.000556	.001396	.000656	.001640	.000756	.001884
.000457	.001155	.000557	.001399	.000657	.001644	.000757	.001888
.000460	.001159	.000560	.001403	.000660	.001647	.000760	.001892
.000461	.001163	.000561	.001407	.000661	.001651	.000761	.001895
.000462	.001167	.000562	.001411	.000662	.001655	.000762	.001899
.000463	.001171	.000563	.001415	.000663	.001659	.000763	.001903
.000464	.001174	.000564	.001419	000664	.001663	.000764	.001907
.000465	.001178	.000565	.001422	.000665	.001667	.000765	.001911
.000466	.001182	.000566	.001426	.000666	.001670	.000766	.001914
.000467	.001186	.000567	.001430	.000667	.001674	.000767	.001918
.000470	.001190	.000570	.001434	.000670	.001678	.000770	.001922
.000471	.001194	.000571	.001438	.000671	.001682	.000771	.001926
.000472	.001197	.000572	.001441	.000672	.001686	.000772	.001930
.000473	.001201	.000573	.001445	.000673	.001689	.000773	.001934
.000474	.001205	.000574	.001449	.000674	.001693	.000774	.001937
.000475	.001209	.000575	.001453	.000675	.001697	.000775	.001941
.000476	.001213	.000576	.001457	.000676	.001701	.000776	.001945
.000477	.001216	.000577	.001461	.000677	.001705	.000777	.001949

INDEX

MUMPS-11
Language Reference
Manual
DEC-11-MMLMA-D-D

READER'S COMMENTS

NOTE: This form is for document comments only. Problems with software should be reported on a Software Problem Report (SPR) form.

Did you find errors in this manual? If so, specify by page.

____

____

____

____

____

____

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

____

____

____

____

____

____

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

____

____

____

____

____

____

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Non-programmer interested in computer concepts and capabilities

Name ______________________ Date ______________

Organization ______________________________

Street ______________________________

City ______________ State __________ Zip Code or Country __________

If you require a written reply, please check here. ☐

-- Fold Here --

-- Do Not Tear - Fold Here and Staple --

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Software Communications
P. O. Box F
Maynard, Massachusetts 01754

digital

digital equipment corporation

Printed in U.S.A.