pdp11

MUMPS-11 Programmer's Guide

Order No. DEC-11-MMPGA-E-D

digital

MUMPS-11
Programmer's Guide

Order No. DEC-11-MMPGA-E-D

digital equipment corporation · maynard, massachusetts

First Printing, October 1972
October 1973
November 1974
January 1976
April 1977

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.

3/78-14

CONTENTS

CONTENTS (Cont.)

CONTENTS (Cont.)

CONTENTS (Cont.)

CONTENTS (Cont.)

FIGURES

CONTENTS (Cont.)

FIGURES

TABLES

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 and Interpretive Data Systems, Inc., assisted in the Version 4A 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, interpretive language, multi-user operating system for the PDP-11 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 applications programmer is relieved of all the burdens associated with driving peripheral equipment or assembly language programming. He may concentrate his energies on the analytical aspects of his problem. His major problems are: developing proper logical hierarchy for his data base, and developing efficient logic for his data processing requirement.

The MUMPS language is provided with its own stand-alone operating system. In addition to supporting 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 can be made available either to all system users or to a class of system users.

PREFACE

The MUMPS-11 Programmer's Guide is a reference manual designed to give the programmer all the information required to create, execute, and save MUMPS Language programs in the MUMPS-11 system environment. Chapter 1 provides a system overview which includes information on the MUMPS System Software/Hardware environment and the functions performed by the various modules of the operating system.

Chapter 2 provides complete information on the use of the MUMPS terminal. Specific subject areas covered are, log-in/log-out procedures and how to create, modify, run, save, and delete programs.

Chapter 3 contains general information on the use of I/O devices in MUMPS programs and specific operating information on each.

Chapter 4 describes the MUMPS Library Utility Programs, their functional characteristics and how to run them. In addition, information is provided which describes how to create Library Programs and Globals.

Chapter 5 discusses various programming considerations and techniques for the application and system programmers. Included are data base design considerations and the use of special elements of the MUMPS-11 language.

Appendices A through E are general reference sections for the application programmer, including a glossary, the system's error messages, and useful tables. Appendices F and G describe the internal system structure for the system programmer. Appendix I describes the conventions and characteristics of an I/O device driver for those users who wish to add devices to the system.

The *MUMPS-11 Language Reference Manual* is a prerequisite to complete understanding of this manual. The *MUMPS-11 Operator's Guide* contains information of specific interest to system operators and managers. The *Introduction to MUMPS-11 Language* is a tutorial manual that describes the MUMPS-11 language and data base in a step-by-step manner.

The following symbols are used throughout this manual:

Symbol	Definition
▌	Revision bar. Indicates differences between this manual and the previous edition.
↵	Universal symbol for line terminator. Line terminators for terminals are either Carriage RETURN or ALT MODE.
␣	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.
CTRL	Used with special system control characters. Depress CTRL key while striking designated character.
<u>UNDERLINING</u>	All examples of user-typed terminal input are underlined.
↑U or ^U	Echo for CTRL/U keyboard command.
\	Backslash - echo for RUBOUT keyboard command.
BREAK	Operation of BREAK key.

CHAPTER 1
INTRODUCTION

1.1 SYSTEM OVERVIEW

The MUMPS-11 system is a multi-user, time-sharing system which runs on the PDP-11 computer. Utilizing the high-level, interactive, string-oriented MUMPS-11 interpretive language, the system permits up to 40 simultaneous users, operating on any of the system's 65 terminals, to interact with a common tree-structured data base. The system is specifically designed to manipulate strings of data and to increase or decrease the size of data storage areas on the disk.

Additional features include:

- Variable length data elements and logical records
- Random access of data using multiple keys
- Variety of terminal and peripheral devices
- System utilities for backup, validation and reporting.

The PDP-11 is used as the Central Processor. It has a 16-bit word length and can be expanded from 49,152 to 245,760 bytes of memory.

Both fixed-head and removable disk pack systems are used for on-line storage of user programs, the data base, and system utility programs. The maximum on-line storage that the system can provide is approximately 417 million words.

A variety of terminals and printers are also supported, including: ASR-33 and ASR-35 Teletypes[1], LA30, LA36 DECwriters, LA180 matrix printers and VT05, VT50, VT52, and VT55 video terminals. In addition, the system can utilize other Teletype-like devices which can be connected to DIGITAL KL11, DL11, DC11, DH11 or DZ11 controllers.

Other standard peripherals supported by MUMPS-11 include industry-compatible disk, magnetic tape, DECtape, paper tape reader/punch, and line printer.

---

[1]Teletype is a registered trademark of Teletype Corporation.

The MUMPS Language contains a large repertoire of capabilities. The basic orientation of MUMPS-11 is procedural, much like FORTRAN or COBOL. Its capabilities are primarily directed toward the processing of variable length string data. In addition, standard algebraic and Boolean operations are also available. Data is represented in either string or numeric form, and mixed mode operations are expressly permitted. The language also allows assembly language-like bit manipulation operations.

Language processing is in every sense interpretive. Each line of a MUMPS command undergoes identical processing each time it is executed (intermediate code is not generated). The language interpreter has two operating modes: program execution mode (Indirect Mode) and program creation mode (Direct Mode). In Direct Mode, programs can be created, modified, debugged, and stored. Indirect Mode operation permits the execution of these programs.

The operating system is highly modular and resides permanently in memory (Figure 1-1). The system uses between 22K and 40K bytes of memory, depending on the hardware configuration. During system generation, the remaining memory is subdivided into user partitions[1] which are used to contain user programs. A partition holds one active user's program, local data, and system overhead data. There is no fixed correspondence between terminals and partitions. Assignment is performed dynamically at log-in time. The recommended size for partitions is approximately 4096 bytes each, but they do not all have to be the same size. A user's terminal is assigned the next available partition.

Each active user requiring CPU time obtains a time slice in turn. A checkpoint form of timesharing is utilized whereby a program is allowed to execute until its time slice has expired, plus any additional time required to complete a current operation. Control then passes to the next job (in priority order) requiring service.

---

[1]Machines with no more than 28K words of memory may have a maximum of 18 partitions. Machines with more than 28K bytes of memory may have a maximum of 40 partitions.

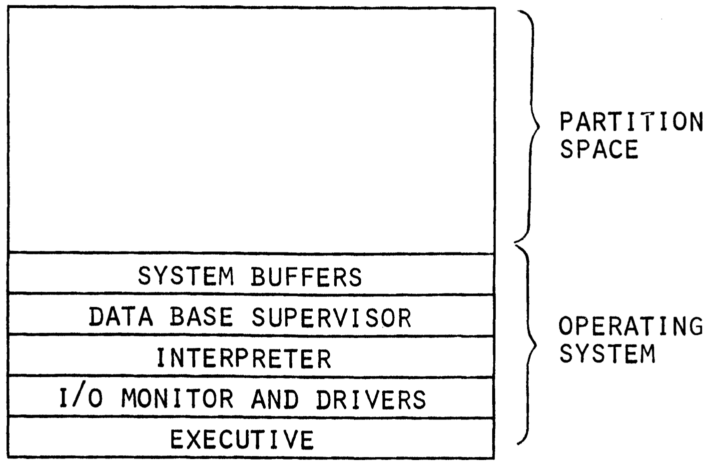


Figure 1-1 MUMPS-11 Memory Layout

The data management features of the system allow local data utilized by a program to be referenced symbolically. Storage space for this data is allocated as needed. Local data is that set of variables established within the domain of a particular partition and is defined only for programs within that partition. This form of storage is used for scratch or transient data. These local data arrays are treated as if they are intended to be sparse. That is, only subscripts for which data are defined are allocated space. A symbolic variable used in a program may be given either a numeric value or a variable length string value. When it has a string value, only that space actually required by the string is allocated.

This philosophy is extended to the management of data on the random access disk system. All elements stored in data files are referenced symbolically; the file name is similar to that of a symbolic local variable name in a program. Records in the data file are treated as array elements and are referenced by means of subscripts; subrecords are referenced by appending additional subscripts. Data files on the disk thus comprise an external system of arrays, which provide a common data base available to all programs within a given user class. The arrays which make up this external system are called global arrays. Each global array is identified by a unique name.

The structure of global arrays is hierarchical. Any element within an array tree may possess a numeric or string data value and (or) be a pointer to a lower level in the tree. Data may be stored at any level. There are no constraints upon the dimension or the size of an array. In addition, the number of subscripts in an array is dynamic, so that its content and structure may change during usage.

In addition to storing global data files, the disk is also used to contain MUMPS Language programs including the System Utility Programs and user-created programs.

The availability of programs and global data to users is controlled by the system's protection scheme. Up to sixteen classes of users can be defined within the system. Each user class has access only to those programs and globals residing in that class. Further, specially named library programs residing in UCI #1 (the System UCI) can be accessed by all users. Using an easily modifiable password at log-in time allows access to an associated user class. This password, called a User Class Identifier (UCI)[1], allows Indirect Mode Operation; i.e., programs can be run, but not changed, and global data can be read or written by these programs. An additional code called the Programmer Access Code (PAC)[1] can be used with any UCI code to permit Direct Mode operation. This allows programs and global files in a particular user class to be created and modified.

A set of MUMPS Language Utility Programs provides the user with the tools to maintain and service the system efficiently. The functions performed fall into four major categories: data base integrity, system and data base backup, system parameter changes, and utility subroutines. Data base integrity utilities enable the user to validate the structure of his data base(s). System and data base backup utilities enable the user to copy disk images and to save and restore individual programs and globals. System parameter change utilities allow the user to modify the system configuration as required. Utility subroutines are a set of programs which provide commonly used operations, such as printing the date and time.

1.2 SYSTEM HARDWARE

The MUMPS system is defined within the limits of a particular PDP-11 hardware system configuration; i.e., memory size requirements, necessary features, and types and numbers of peripheral devices. The

---

[1]The PAC and UCI codes are assigned by the system manager at System Generation time as described in the *MUMPS-11 Operator's Guide*.

system is distributed on magnetic tape (7-track or 9-track), and RK05 or RK06 disks. It requires at least one type of the four disk systems available. During system operation, disk, DECtape, magnetic tape, or paper tape is used for backup storage. Figure 1-2 shows the MUMPS minimum hardware requirements.

1.2.1 Minimum Hardware Requirements

The minimum equipment configuration necessary for MUMPS operation includes:

- PDP-11/10 Central Processing Unit
- 24K words of Memory
- Extended Arithmetic Unit (KE11A or KE11B) or Extended Instruction Set for 11/34, 11/40 (KE11E)
- Real Time Clock (KW11-P or KW11-L)
- Bulk Storage (ROM) Bootstrap Loader (MR11-DB)
- Communications Interface (DL11A or KL11A)
  Console Terminal (VT05, VT50, VT52, VT55, LA30, LA36, LA180, ASR33, or ASR35)
- Tapes[1]

  Magtape Control (TM11)
  Magtape Transport (TU10 or TS03) 7- or 9-track)

  OR

  Magtape Control (TJU16)
  Magtape Transport (TU16) 9-track

  OR

  DECtape Control (TC11)
  Dual DECtape Drive (TU56)

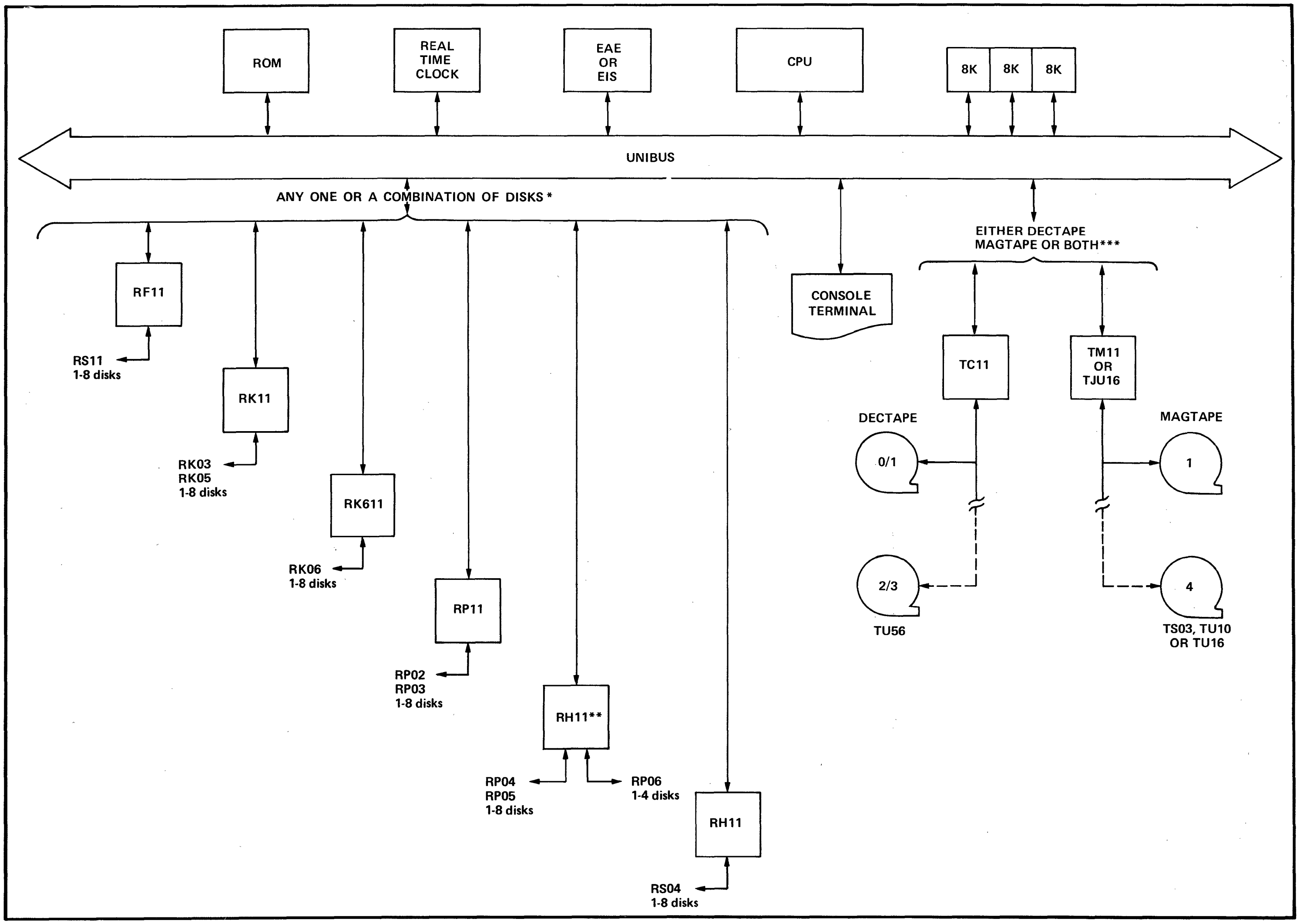
- Disks:

  DECpack Control (RK11)
  DECpack Drive (RK03/RK05J), 1.2 million words

  OR

  Disk Pack Control (RK611)
  Disk Pack Drive (RK06), 7 million words

[1]Tape is not necessary if your system includes 2 RK05 or RK06 drives.



* Due to system restrictions, only one of either the RK611, RF11, or RH11 (RS04) disks can be used. RP02, 03 disks and RP04, 05, 06 cannot be combined on the same system.

** An RP06 disk counts as 2 RP04 or RP05 disks in MUMPS.

*** If 2 RK05 or RK06 drives are used, no magtape or dectape is necessary.

Figure 1-2 MUMPS Minimum Hardware Requirements

OR

Disk Pack Control (RP11)
  Disk Pack Drive (RP02 or RP03), 10.24 million words
                                  or 20.48 million words

OR

Disk Pack Control (RH11)
  Disk Pack Drive (RP04, RP05) 44 million words
  Disk Pack Drive (RP06), 88 million words.

1.2.2 Optional Hardware

The following hardware, in addition to that previously specified, is also supported by MUMPS:

- Memory: Core, MOS, or Bipolar Memory up to 124K words.
- Floating Point Processor for 11/45, 11/55, or 11/70 (FP11-B or FP11-C).
- Memory Management Option (KT11 -C or -D).
- Communications Interface (DL11-E), connected to a Bell System type Dataset (max 1800 baud), 1 per terminal.
- Communications Controller (DMC-11), direct connection from CPU to CPU via a 20 milliamp current loop, or connected to one or more synchronous modems.
- Terminals (maximum 16 single-line controllers - DL11A or DL11C), ASR33 or ASR35 teleprinters, LA30, LA36 DECwriters, or VT05, VT50, VT52 and VT55 video terminals.
- Three multiplexers (DH11-AA or AC), or six multiplexers (DZ11), with a maximum of 16 terminals per DH multiplexer or a maximum of 8 terminals per DZ multiplexer (2 DZ11's are equivalent to 1 DH11; DZ's and DH's may be mixed on the same system)[1].
- High Speed Paper Tape Reader/Punch (PC11).
- Punched Card Reader (CR11) and Mark Sense Card Reader (CM11).
- Line Printer (LP11), LA180 matrix printer.
- DECtape Control (TC11).
- Two Dual DECtape Drives (TU56).

[1]See Table 3-1 for further information.

- Magtape Control (TM11) with up to 4 Magtape Drives (TU10 or TS03), 7- or 9-track, or Magtape Control (TJU16) with up to 4 Magtape Drives (TU16), 9-track.
- Disk Pack Control (RF11) with up to 8 Disk Pack Drives (RS11), 209.72 million words total.
- DECpack Control (RK11) with up to 8 DECdisk Drives (RK05J), 9.6 million words total.
- Disk Pack Control (RK611) with up to 8 Disk Pack Drives (RK06), 56 million words total.
- Disk Pack Control (RP11-C) with up to 8 Disk Pack Drives (RP02/RP03), 81.92 million or 163.84 million words total.
- Disk Pack Control (RH11) with up to 8 Disk Pack Drives (RP04, RP05) 351.84 million words total; or up to 4 Disk Pack Drives (RP06), 351.84 million words total (1 RP06 is equivalent to 2 RP04's or RP05's; RP04's, 05's, 06's, may be mixed in the same system).
- Disk Pack Control (RH11) with up to 8 Disk Pack Drives (RS04), 419.43 million words total.

NOTE

BA-11 Expander Boxes are required to provide power to peripheral devices and a DB-11 Bus Repeater is also needed if more than 18 loads are attached to the UNIBUS. This hardware presents no additional programming consideration but should be included as required to suit specific hardware configuration.

1.2.3 The System Device

The System Device is the disk memory device on which the MUMPS software resides. Specifically, the system resides on physical unit 0 of any one of the allowable disk systems (see Section 1.2.1). Thus, either RK0, RF0, RP0, or RJ0 can be used. The software occupies up to the first 50 blocks on the system device, while the remaining portion as well as all other disk units in the configuration are used to store MUMPS programs and data, including Sequential Disk Processor files.

1.3 SYSTEM SOFTWARE

The software which comprises MUMPS consists of the MUMPS Operating System and the MUMPS Library and System Utility Programs.

1.3.1 Operating System

The Operating System contains all software necessary to operate MUMPS in the hardware environment of the PDP-11. The software is entirely core resident and consists of the four subsystems described below:

- Executive - Supervises the timesharing/multiprogramming operations of the system.
- I/O Monitor - Supervises terminal and peripheral device I/O and interrupt processing.
- Language Interpreter - Implements and provides execution control of MUMPS Language programs.
- Data Base Supervisor - Performs all logical and physical control of the data base.

The paragraphs which follow describe the operational features of each.

1.3.1.1 Executive - The Executive implements the time-sharing aspects of the system and permits partitioned multiprogramming using dynamic assignment of user partitions. The Executive is used to pass control from one user to another in order to utilize the central processor as much as possible. As a result of memory partitioning, the Executive can switch from user to user in minimum time.

The Executive uses a set of priority-weighted queues to administer its scheduling algorithm including one or more Wait-Queues and a Run-Queue. Initially a job starts at the end of the highest priority Wait-Queue. Upon reaching the front of this queue, the job is placed in the Run-Queue and allowed to execute for the duration of its time slice. If the job is processor-bound at the end of its time slice, the Executive drops it from the Run-Queue and places it at the end of the next lower priority Wait-Queue. When the job reaches the front of this queue, the Executive doubles the job's time slice and places it in the Run-Queue. If the job remains processor-bound, it is placed in the lowest priority Wait-Queue upon expiration of its time slice. When it reaches the front of this queue, it is allocated a triple time slice and is placed in the Run-Queue. Thereafter the Executive circulates the job between the lowest Priority Wait-Queue and the Run-Queue. When the job becomes input/output bound, the Executive places the job in an

'I/O hung' state to await completion of the requested input/output task. Completion of the task causes the Executive to place the job at the end of the highest priority Wait-Queue.

The Executive runs jobs from lower priority Wait-Queues only when the higher priority Wait-Queues are empty. This technique produces the most favorable response time for the interactive parts of any job by servicing input/output bound tasks very quickly but taking longer to service CPU-bound tasks.

1.3.1.2 Input/Output Monitor - Once a job becomes input/output bound, the Executive places the job in the appropriate 'hung' state and the Input/Output Monitor initiates and processes the input/output activity through its interrupt handlers. The MUMPS Interpreter and the I/O Monitor communicate through buffers for terminal input/output character processing, but the I/O Monitor supervises the asynchronous filling and emptying of these buffers so as to overlap output with that program's processing whenever possible. The Input/Output Monitor creates a terminal-independent environment in which an application program can run with any terminal of the hardware system regardless of its specific speed and formatting characteristics. At terminal log-in, a partition initially 'owns' one terminal; it may subsequently acquire other terminals in the system or it may release the original terminal and continue as a background job.

The I/O Monitor also supervises the peripheral input/output devices of the system, including the magtape and DECtape drives, the paper tape reader-punch, line printer, etc.

1.3.1.3 Log-in Processing and the Language Interpreter - During terminal log-in, a user is assigned an available partition. User Class Identifier codes and the Programmer Access Code are checked for validity, resulting in either authorization or denial of access to associated programs and global files. Since terminal programming of application packages is allowed, stringent checks are performed by the Interpreter to safeguard the system's service operations from all programming activities. If the user intends to program, his partition is initialized and control is passed to the Interpreter for the subsequent programming session. If the user desires activation of a service program, the requested program is loaded from the disk into his partition and execution of that program commences. In either case, the user retains his partition until he logs-off the system.

All application programs are written in the MUMPS language. This language allows an application programmer to write a program and debug, edit, run, and modify it in a single interactive session at a terminal. This minimizes the programmer's time in solving a problem, the computer time needed in checking it out, and, most important, the elapsed time required to obtain a final running application program. The Interpreter itself is an integral part of the system. The Executive and I/O Monitor have been specifically tailored to work efficiently with the Interpreter.

The Interpreter examines and analyzes all MUMPS Language statements, executing, in turn, the desired operations. Each MUMPS Language statement undergoes identical processing each time it is examined (executed) by the Interpreter; intermediate code is not generated. Comprehensive error checking is also performed to assure proper language syntax.

In addition, the Interpreter also files and loads programs via the disk storage system. During program execution, it automatically overlays external program segments invoked by an active program. Proper linkages are set up to return to the invoking program when execution of the segments terminates.

A number of major advantages are obtained from the use of the Interpreter as the major components of the MUMPS system. First, programs written in an interpretive language do not require any compiling or assembling. Error comments during execution are printed at the programmer's terminal and allow quick recovery, modification of the program, and re-execution of it. All program debugging and modification operations are performed in the MUMPS language directly at the terminal. This makes modification convenient, particularly in a service environment where the trouble-shooting necessary to interface a program with an application area is a time-consuming process. The MUMPS environment allows a programming session to take the form of a conversational dialogue between the programmer and the terminal device.

1.3.1.4 Data Base Supervisor - The Data Base Supervisor consists of a group of routines which provide physical as well as logical control of the various disk systems which form the data base.

In MUMPS, all file information is referenced symbolically, in the context of hierarchical global variables and arrays. This replaces the classical manner of sequentially accessing record files on secondary memory devices. Instead, an attempt is made to logically map the content and structure of the tree-like symbol tables into the physical storage medium of the system. The Data Base Supervisor maps logical information at a specific level of an array into directories of fixed size blocks. These blocks are chained together in a linear fashion to contain all the data values stored at that specific level, as well as the pointer words which link it to chains of the next lower level. The data base devices used in the system are some combination of fixed head disks and/or moveable head disk packs. The organization of either an individual platter of the fixed head disk or a cylinder of the moveable head disk pack is two-dimensional, wherein any physical block has a track and segment coordinate.

Maps of addresses of unused disk blocks (storage allocation maps)[1] are maintained to facilitate the dynamic allocation of disk storage space to files. Whenever a continuation or a header block is to be allocated, a block in the map, whose segment address is a few segments away, is utilized. This method ensures that the time required to retrieve a particular datum is kept to a minimum. When data is updated, care is given to repacking, and sometimes reorganizing the individual data elements within a chain, to ensure maximum utilization of space for variable length data.

Once a block of data accommodating a given level of subscripting is referenced, its address is placed in the partition's overhead area and the block remains in memory until a reference to a different block is made. This permits considerable time savings when the 'naked' syntax is used in referencing globals[2].

When a level is reached, often no further disk access need be made to reference associated information. If any block in a disk is altered, it is written back on the disk.

When a part of a global structure is deleted, it is attached to a 'Garbage Chain'. The Garbage Collector routine removes blocks from the tree-like chain and refills the Storage Allocation Maps with the

---

[1]The Storage Allocation Maps are bit maps where there is a correspondence between map address and bit position within the map, and the disk address of the block.

[2]Appendix G provides a detailed description of the structure of global data.

addresses of the expunged blocks. This is done during periods of low CPU activity to avoid competition with active programs.

1.3.2 Utility Programs

The MUMPS Utility Programs are a package of programs written in the MUMPS Language which are supplied as a part of the MUMPS software package. These programs are provided to assist both the MUMPS System Programmer and the MUMPS System Manager in developing and maintaining the software and data for their particular application. The Utility Programs consist of two functionally distinct groups: System Utility Programs and Library Utility Programs. The System Utility Programs provide functions for use by the System Manager; they reside on the disk under the control of the system UCI (UCI #1) and are accessible only to those individuals possessing the System UCI Code. Library Utility Programs provide general services which are available to all system users, regardless of UCI[1]. These programs also reside under UCI #1 but employ a naming convention which distinguishes them from System Utilities. The main difference between the System Utilities and the Library Utilities is that only users logged-in under the System UCI code (UCI #1) may use the former, while all users, regardless of their UCI codes, may use the latter. Tables 1-1 and 1-2 briefly describe the MUMPS-11 Utility Programs. Further information about Library Utilities is provided in Chapter 4. The MUMPS-11 Operator's Guide describes the System Utilities.

---

[1]The Global Place Program (%GP) is an exception to this rule. Though functionally a system utility, it resides in the system as a library utility because of the nature of its operation. %GP is documented in Chapter 4 of the MUMPS-11 Operator's Guide.

Table 1-1

System Utility Program Summary

Name	Description
BCS	Broadcast Program Allows the operator to send messages to all or specified terminals.
CTK	System Caretaker Program Collects system error statistics.
DAT	Date Routine (or Date Set) Sets the $D system variable to the current date.
DBT	Disk Block Tally Program Calculates the number of disk blocks available for each disk (logical and physical), updates Disk Storage Allocation Table, tallies and reports errors.
DMP	Disk Block Dump Program Lists the contents of disk blocks or gives an analysis report of the system's crash block.
KTR	Caretaker Reporter Program Lists error statistics collected by the caretaker.
MSP	Modify System Parameters Program Alters UCI codes, terminal types, number and size of partitions, Programming Access Code, Magtape default mode.
RKC	RK Copy Used in RK05 distribution to physically copy one RK05 to another.
RSJ	Restore Job Allows jobs that are either in a wait queue or in an I/O hung state to be restored to the system.
RST	Restore Devices Releases devices either owned by a job or disabled by a system crash.
SDP	Sequential Disk Processor Space Allocation/Deallocation Program.
SIF	Status Information Program Provides system status information (calls the SS program) and system partition size assignments.
SS	System Status Program Provides information about the current users (UCI's) in the system, the status of their jobs, and utilization of system resources.
SSD	System Shutdown Gives instructions to the operator on how and when to "HALT" the system.

Continued on next page

Table 1-1 (Cont.)

System Utility Program Summary

Name	Description
STU	System Startup Initializes the system when disk bootstrap loading is performed.
SYSGEN	System Generator Program Tailors the basic MUMPS Operating System for specific hardware configuration.
TIM	Time Routine Sets the $T System Variable to the current time.
TP1 through TP8	System Test Package Provides basic test programs to help verify that a MUMPS system is operational.
%GP	Global Place Allows the system user to position global files on a specific unit and cylinder of a disk drive.

Table 1-2

Library Utility Program Summary

Name	Description
%D	Date subroutine Reports the current (system maintained) date on the specified I/O device.
%FD	Fast Program Directory Lister An abbreviated high-speed version of %PD, %FD lists only program names.
%GD	Global Directory Lists the names of all globals of the current UCI onto the designated output device.
%GL	Global Lister Lists the structure and content of a specified global file on the designated I/O device.
%GR	Global File Restore Restores all or specified Global Files onto the data base, entering their names in the Global Directory of the current UCI.
%GS	Global File Save Copies all or specified global files listed in the global directory or the current UCI onto the designated output device.
%GT	Global Trace Program Lists global nodes, their location, level, data type and contents for the current UCI.

Table 1-2 (Cont.)

Library Utility Program Summary

Name	Description
%GU	Global Utilization Program Analyzes a global, giving the number of nodes, total bytes, bytes/block and % utilization for each data type, for system overhead and for each free area in a global.
%GV	Global View Program Dumps a global disk block as seen by the system; prints subscript, pointer, data type and data for each node.
%IO	I/O Device Assignment Subroutine Assigns specified I/O device if available and informs the calling program of result.
%IU	In Use Message Program Displays the message "IN USE" on the currently assigned device.
%OD	Octal/Decimal Conversion Program Converts octal or decimal values to their decimal or octal equivalents.
%OP	User to Operator Communication Program Allows a terminal user to send messages to the console terminal.
%PD	Program Directory Lister Lists the contents of the Program Directory, the starting disk block number, and the length of each program of the current UCI on the designated I/O device.
%PL	Program Load Loads programs residing on paper tape, DECtape, or magtape, which were saved via %PS, onto the disk, and enters their names in the Program Directory of the current UCI.
%PS	Program Save Copies specified programs listed in the Program Directory of the current UCI onto the designated output device.
%T	Time Subroutine Reports the current (system maintained) time on the specified I/O device.

CHAPTER 2
USING THE TERMINAL

2.1 INTRODUCTION

MUMPS terminals are not only data input and output devices to be used with application programs, they are also the means by which MUMPS programs are created and executed.

This chapter describes how to use MUMPS 11 terminals for developing MUMPS application programs. In particular, the chapter describes how to: log-in and log-out; enter commands; correct typing errors; and store, load, and modify programs. The last section of this chapter describes MUMPS error processing.

Operating at a terminal, the programmer can:

- Execute MUMPS commands immediately;
- Input the steps of a program;
- Run programs and access global files listed in the directories of the current UCI;
- Run Library Utility Programs[1].

2.2 TERMINAL TYPES

There are many types of terminals used in MUMPS systems, but most have typewriter-like keyboards that do not vary significantly from one to another. The main differences between terminals are: choice of hard or soft copy, speed of operation, and the location of special control characters on the keyboard.

As described in Chapter 1, individuals who are permitted to run MUMPS programs are given an appropriate UCI code. With this code, they can run any program in the UCI's program directory. They can not, however, modify or create programs: a UCI alone does not allow access to Direct Mode. The system's Programmer Access Code (PAC) permits entry into Direct Mode through which programs can be created and modified and individual MUMPS commands can be executed.

---

[1]Library Utility Programs are described in Chapter 4.

2.3 PRELIMINARY OPERATIONS

Before attempting to log-in, the user should make sure that the power on his terminal is turned ON and the terminal is ON-LINE. The procedure varies from terminal to terminal, but for Teletypes and DEC-manufactured terminals it is as follows:

- Teletypes - Turn the LINE-OFF-LOCAL switch to LINE. The teleprinter's motor should start.
- LA30 DECwriter - Set the circuit breaker on the left of the back panel to ON. The motor should start and the READY light should be lit.
- LA36 DECwriter II - Operate the switches at the left side of the keyboard as follows:

  POWER ON/OFF to ON
  LINE/LOCAL to LINE
  BAUD RATE to match the applicable baud rate of the hardware controller.
- VT05 Video Terminal - Set the ON-OFF switch on the right side of the keyboard to ON; then set the LOC/REM switch to REM. The blinking cursor should appear on the screen after a few moments.
- VT50, VT52, and VT55 Terminals - Set the POWER ON/OFF switch located in the recessed portion of the right side of the cover to ON. Set switches S1 and S2 to match the applicable baud rate of the hardware controller as shown in Table 2-1.

Table 2-1
VT50 Baud Rate Switch Settings

Mode	Baud Rate Transmit	Receive	Switch S1*	Switch S2**
Local	9600	9600	1	G
	4800	4800	1	F
	2400	2400	1	E
	1200	1200	1	D
	600	600	1	C
	110	110	1	B
Full Duplex with Local Copy	9600	9600	2	G
	4800	4800	2	F
	2400	2400	2	E
	1200	1200	2	D
	600	600	2	C
	110	110	2	B
Full Duplex	9600	9600	3	G
	4800	4800	3	F
	2400	2400	3	E
	1200	1200	3	D
	600	600	3	C
	300	300	4	A
	150	150	5	A
	110	110	3	B
	75	75	6	A
Full Duplex (Split Speeds)	300	9600	4	G
	150	9600	5	G
	75	9600	6	G
	300	4800	4	F
	150	4800	5	F
	75	4800	6	F
	300	2400	4	E
	150	2400	5	E
	75	2400	6	E
	300	1200	4	D
	150	1200	5	D
	75	1200	6	D
	300	600	4	C
	150	600	5	C
	75	600	6	C

*Switch S1 Labels
1 = Local
2 = 1/2
3 = Full Duplex
4 = 300 Baud
5 = 150 Baud
6 = 75 Baud

**Switch S2 Labels
A = Bell System type 103 dataset
B = 110 Baud
C = 600 Baud
D = 1200 Baud
E = 2400 Baud
F = 4800 Baud
G = 9600 Baud

2.4 SPECIAL KEYBOARD CONTROL CHARACTERS

The following control characters have special significance to MUMPS terminals. CTRL characters are formed by depressing the CTRL key while striking the associated character key as with CTRL/C and CTRL/U.

- BREAK and CTRL/C - The BREAK button on the Teletype and the CTRL/C function on all terminals are used interchangeably to request use of a terminal or to attempt to interrupt the current operation. The use of BREAK and CTRL/C as interrupting characters is discussed under the ASSIGN command in the *MUMPS-11 Language Reference Manual* and in paragraph 2.5 of this manual.

- Carriage RETURN/ALT MODE/ESC - These characters are used interchangeably as line terminators for terminal input. They are represented internally by the NUL (ØØØ) ASCII code. Note that this usage of ESC is different from that for the VT52 (described in Section 3.7.1.4).

- RUBOUT - Deletes single characters in the current line starting with the rightmost character. Each time the key is depressed one character is deleted and a backslash (\) is echoed. (A RUBOUT on VT05 erases the last character typed.) This command is effective only if a line terminator has not been typed for the line in question.

  Example:

  ```
  1.25 T "FORESOCRE
  ```

  Given that the final 'E' has just been typed, the above misspelling can be corrected by typing seven RUBOUTs followed by the correct letter sequence.

  ```
  1.25 T "FORESOCRE\\\\\\\URSCORE
  ```

- CTRL/U - Deletes the entire line. Like RUBOUT, CTRL/U works only on lines which have not been terminated. Thus, typing CTRL/U rather than RUBOUT in the previous example would delete the whole line. MUMPS echoes ↑U followed by a Carriage RETURN/LINE FEED.

- CTRL/O - Suppresses terminal output printing. Typing CTRL/O again restores output printing if there is any more output to the terminal. The system automatically restores output printing when: returning to Direct Mode; the terminal is ASSIGNed; an error occurs; or a BREAK or CTRL/C is received and the terminal is unowned.

2.5 LOGGING-IN TO THE SYSTEM

Each user of a MUMPS terminal gains access to the system's programs using a special log-in sequence which involves one or two access codes (depending on the privileges allowed the user). These codes, provided by the MUMPS System Manager, are the User Class Identifier code or UCI, and the Programmer Access Code, or PAC. The MUMPS-11 System can have up to sixteen UCI's (classes of users). The UCI code must be entered by all terminal users. It allows access to the programs and globals listed in the Program and Global Directories assigned to that UCI. A user who is permitted simply to run programs needs to know only the UCI code and the names of those programs which he is permitted to use. Users who are allowed to create or modify programs and global files must know the system's PAC. This code permits system operation in Direct Mode whereby a programmer can execute MUMPS commands at the keyboard, as well as create, modify, and delete global data and programs associated with a specified UCI. The following procedures describe the steps to be used for system log-in:

a. Type either CTRL/C or BREAK. MUMPS will respond with

   MUMPS-11 Vnn nn

   where: vnn = version of system
   nn = terminal's device number

   On the next line, MUMPS types

   UCI:

b. The terminal user should respond in one of two ways. If a UCI and a program name are specified, that program will be loaded and started upon a successful log-in. If a UCI and the PAC are specified, MUMPS will enter Direct Mode. The response must be in the following form:

$$\text{uci:}\left|\begin{matrix}\text{prog}\\ \text{pac}\end{matrix}\right| \;\lrcorner$$

where: *uci* = *the user's UCI code*

: = *delimiter*

pac = *the Programmer Access Code (PAC)*

prog = *the name of a program to be run residing in the Program Directory of the UCI.*

If the user does not respond within 20 seconds, he is automatically logged-out and must again type CTRL/C or BREAK to reinitiate the log-in sequence.

c. After the codes have been typed, depress either Carriage RETURN or ALT MODE. If the codes are legitimate, MUMPS will either run the requested program, if a program name was specified, or type the right angle bracket (>) prompting symbol to signify its readiness to accept commands in Direct Mode, if the PAC was specified.

d. If log-in was not successful, an error message is printed and the terminal is logged-out.

The following are examples of system log-in sequences (user responses underlined):

a. Log-in to Run a Program (Indirect Mode operation only)

<u>*CTRL/C*</u>		
MUMPS-11 V3B	#6	
UCI: <u>NAM:ZT</u>		*Enter UCI and program name*
10:38AM		*Program runs to completion*
EXIT		*User is logged-out*

b. Log-in to Create, Modify and Run Programs (Direct or Indirect Mode Operation)

```
CTRL/C
MUMPS-11 V3B     #6
UCI: SAM:QYV          Enter UCI and PAC
>                     User is given control in Direct Mode
.
.                     User performs desired tasks
.
.
>HALT                 Log-out
```

2.6 LOGGING-OUT OF THE SYSTEM

When the user wishes to end his session at the terminal, one of the following procedures can be used.

a. If a program is running (i.e., Indirect Mode is in effect), type a CTRL C or a BREAK.

   If the user did not log-in with the PAC and program interrupt is enabled[1], the message "EXIT" is printed and the terminal is logged off automatically. If program interrupt is not enabled, the job may determine the action to be taken.[2] If the user had logged-in with the PAC, control is returned to Direct Mode and a '>' is printed.

b. If the user is operating in Direct Mode, log-out is accomplished simply by typing:

   >HALT

   immediately after the '>' character. The "EXIT" message is printed and the terminal is logged off.

2.7 ENTERING COMMANDS

Once a terminal user has logged-in to the MUMPS system using the Programming Access Code, almost any MUMPS Command or Function can be executed from the keyboard in Direct Mode. Exceptions are: OVERLAY, BREAK, ELSE, and GOTO.

---

[1]See ASSIGN Command in the *MUMPS-11 Language Reference Manual*.
[2]See paragraph 5.2 on the $J System Variable.

2.8 SUMMARY OF COMMAND AND FUNCTION SYNTAX RULES

The following is a list of the rules of syntax as presented in the *MUMPS Ldnguage Reference Manual.*

- 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.

- Commands which are to be executed as part of a stored program (Indirect Mode) are preceded by a Step Number. The first command on a line is separated from the Step Number by a single space.

- Each command may be abbreviated to its first letter (first letter after the '$' for a function). 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 be executed only in Indirect Mode.

  Example:

  ```
  E␣2.5
  ```

  In Direct Mode, it means: 'ERASE Step 2.5'. But: in Indirect Mode, it is read as: 'ELSE' and produces a SYNTX error, since 2.5 is not a valid command.

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

- Multiple arguments to a command or function are separated from each other by single commas.

- Multiple commands on the same line are separated by single spaces.

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

- Program comments may be added to any command line. When comments are used, they must begin with a semicolon (;). The semicolon must be separated from the preceding command argument list or Step Number by a space.

- The indirection syntax operator, symbolized by underscore (__) or back arrow (←), provides dynamic command argument definition. In form, the command argument is replaced by a variable name. During execution, the contents of that variable name are taken as the argument.

  Example:

  where: A=1, B=2 and C=3
  ARG = "B+C-A"

  ```
  >TYPE ←ARG
  4
  >
  ```

- An optional Boolean-valued expression preceded by a colon (:bve) can be used to specify conditional execution of certain commands and command arguments (see Appendix D).

  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 W:A=B 2
  ```

  If A=B, all the steps in part 2 are written out to the currently assigned I/O device.

  The colon is also used as a field delimiter in the arguments of FOR, MODIFY and ASSIGN commands.

  Example:

  ```
  1.09 F I=1:1:10 D X
  ```

  The colon can also be used to indicate the presence of an optional expression appended to some command arguments.

  Example:

  `6.30 READ X:5` *is a 'timed' Read.*

- Function arguments are enclosed in parentheses and immediately follow the function name.

- Functions which produce string valued results may *NOT* be nested. Further, where the argument to any function is required to be a string, it must be in the form of a string variable or literal (svl).

2.9 SUMMARY OF RULES FOR EVALUATION

The following is a summary of the rules for expression evaluation:

- Sequence of operations in an expression is strictly from left to right, except that a unary minus is evaluated before a Boolean NOT when they appear as adjacent operators. Parentheses can be used to cause operations to be evaluated in a different order than would be allowed by the normal sequence of operations.
- Expression elements are variables (global, local, and system), literals, constants, functions, and subexpressions (expressions enclosed in parentheses).
- Automatic data mode conversion is employed to convert a string datum to numeric datum and vice versa as required to complete any particular operation.

2.10 CREATING PROGRAMS

To create a program, the terminal user logged in under a PAC simply enters a Step Number at the beginning of a command line. This signals the system to store the line in the program buffer of the user's partition rather than to execute it immediately.

Example:

```
>1.03 TYPE 1+(2*3)-(5/10)
```

The programmer may type in as many program Steps as the size of his partition will permit. If too many Steps are input, however, a PGMOV error will occur. The $S System Variable can be interrogated to determine the number of characters (bytes) of storage remaining in the partition.

Example:

```
>T $S
44                          44₁₀ characters of storage remain
>
```

2.11 STORING PROGRAMS

There are several choices available to a programmer who wishes to store a program within a user partition:

- The program can be stored on the disk.
- The program can be stored on a secondary storage device such as magtape, DECtape, or paper tape.

To store a program on the disk, use the 'FILE␣pnam' command (pnam is a MUMPS identifier with a 3-character limit).

Example:

>F A3C	*files the program called 'A3C' and places its name in the Program Directory of the current UCI.*

There is also an argumentless FILE command, which assumes the current program name.

The ASSIGN and WRITE commands are used to save a program on a secondary storage device.

For example, the following command line writes the program currently in the user's partition onto DECtape unit 1, starting at address 2,56Ø:

```
>A 56:256Ø W  U 56
```

Note that the UNASSIGN is necessary to write the last buffer onto the DECtape, as well as to free the device for other users.

2.12 LOADING PROGRAMS

There are several ways to load programs. The techniques are just the reverse of those used to store programs, with the following addition. If the programmer merely wishes to run a program residing on the disk, loading can be effected by logging-in with the program name.

For example: to run the program 'B23' which is entered in the Program Directory of UCI 'BOB', log-in as follows:

```
CTRL/C
MUMPS-11 V3B      #6
UCI: BOB:B23
.
.                       Program B23 runs to completion,
.                       then exits.
.
EXIT
```

If the programmer wishes to make alterations to a program, continue development, fix bugs, etc., he may load the program in one of the following ways using the LOAD command.

If the program 'A3C' is listed in the Program Directory of the current UCI,

```
>LOAD A3C
```

brings the program into the partition's program buffer.

The CALL command may also be used for disk-stored programs, however it causes automatic program execution.

If the program to be loaded is on a secondary storage device, the LOAD command without the program name argument will load a program from the currently ASSIGNed I/O device.

The following example loads a program from the Paper Tape Reader (device 2):

```
>A 2 LOAD  U 2
```

Note that a LOAD of this form *merges* the program being loaded with the program, if any, currently in the partition. Steps of the program being loaded replace Steps of the same number currently in the partition. If a merge is not desired, ERASE should be issued before the LOAD, as:

```
>E  A 2 L  U 2
```

2.13 STARTING AND STOPPING A PROGRAM

The DO command can be used to start a program currently residing in the user's partition. The programmer can specify an entire Part:

```
>DO 1
>DO 11
```

or individual Steps or groups of Steps:

```
>D 5.32
>D 6.10,4,1.03
```

The entire Part, Step, or group of Steps will be executed and control then returned to the programmer at the terminal.

NOTE

A DO command does not effect complete execution of a program having more than one Part unless a GOTO command is also used within each Part to effect transfer of control from one Part to the next.

Programs are normally stopped by:

- Executing a HALT;
- Executing a QUIT at the lowest (outermost) program nesting level;
- Running out of Step Numbers in the current Part or reaching the end of a Step if 'DOing' a Step;
- Typing a BREAK or CTRL/C from the Principal I/O Device if CTRL C/BREAK recognition is enabled.

2.14 CHANGING, REFILING, AND DELETING PROGRAMS

Program creation takes place in Direct Mode. Once a program is resident in the user's partition, the programmer can:

- Add new Steps by typing them in;
- Replace existing Steps by entering a Step with the same number;

- Delete one or more Steps using the ERASE command;
- Modify Steps using the MODIFY command;
- Print out the entire program or parts thereof using the WRITE command.
- Modify the Program through use of the Editor (cf. Section 4.3).

After a program has been FILEd it remains in the partition until: it is ERASEd, the session at the terminal is HALTed, or another program is loaded from the disk. The programmer can continue to run, modify, and refile it.

Each time a FILE command is executed, the program currently present in the partition replaces any program previously FILEd having the same name.

Deleting a FILEd program is accomplished by FILing an empty Program Buffer. Simply issue an ERASE Command followed by a 'FILE pnam' Command (where pnam is the name of the program to be deleted). This will remove the program from the disk and delete that program's name from the current UCI's Program Directory.

Example:

To delete program 'JOE', type:

```
>E  F JOE
```

Changing a program's name involves a like procedure. First LOAD the program which is to have its name changed. Then FILE it using the new name. Last, delete the old program by issuing an ERASE followed by a FILE using the old program name. To keep several versions of a program, give each one a different name.

2.15 ERROR PROCESSING

The standard MUMPS error processing procedure considers all language syntax, program or system operation errors fatal. MUMPS normally reports errors by:

- setting the $I System Variable to the Principal I/O device number (the device at which the user logged-in)
- printing an appropriate message on the Principal I/O Device
- halting the program

If the user logged-in under a PAC, control returns to the user in Direct Mode. If the user logged-in simply to run a program, the user is automatically logged-out. Appendix C describes the MUMPS error messages.

The user may write an application program which performs its own error processing and avoids interrupting the job with a fatal error, by using the $E System Variable. Refer to Paragraph 5.3 for further information.

CHAPTER 3
USING I/O DEVICES

3.1 INTRODUCTION

MUMPS timesharing allows multiple users to have access to the same central processor via separate remote terminals. It also allows one user to have access to many terminals from one program. In addition to terminals, MUMPS systems also include ancillary Input/Output devices such as the high-speed paper tape reader and punch and DECtape transports. Each I/O device has a unique identification number within the system.

'Ownership' of a device is established by using the ASSIGN Command. In a timesharing environment, many programs may be competing for the use of a single device. Thus, before attempting input or output to a device other than the terminal with which the program is associated, an ASSIGN command must be issued.

Once ownership of a device is established, I/O may proceed using the I/O commands available. In general, the programmer need not be concerned with specific characteristics of MUMPS I/O devices since data transfers consist of ASCII lines of not more than 132 characters. There are, however, certain physical operating characteristics of these devices which may be of interest to the programmer: for example, how to rewind a magtape or access a particular location on DECtape. These characteristics are discussed in later paragraphs.

When an I/O device is no longer needed, it should be released for use by other programs by means of the UNASSIGN Command.

3.2 I/O DEVICE NUMBERS

The unique identification number of each MUMPS I/O device always represents the same device regardless of the hardware configuration of a particular system. For example, the line printer is always Device Number 3. If a particular system does not have a line printer, then Device Number 3 is nonexistent and any attempt to use (ASSIGN) it results in a 'NODEV' error message. The system reports a 'NOTSY' error when the user references a device for which there is no associated driver in the system. For example, a reference to device number 55 in a system not built for DECtape I/O causes a NOTSY error.

Table 3-1 shows the complete list of MUMPS-11 I/O device assignments. Note that device numbers 20 through 45 are specified as Program Interlocks. There are no physical devices associated with these numbers; instead, they are to be used for inter-program communication. Through the use of the ASSIGN and UNASSIGN commands, MUMPS application programs can signal one another. The significance of any particular interlock being 'owned' by any particular program is, of course, established by user's conventions (i.e., assignment of these 'dummy' devices has no particular significance to the MUMPS operating system). Typically, one Program Interlock would be associated with each UCI.

3.3 ASSIGNING I/O DEVICES

In all MUMPS programs, input and output operations are directed to the I/O device whose number is contained in the $I system Variable. When the user logs-in to the system, $I contains the terminal's Device Number. All error messages are directed to this device, unless the program controls its own error processing in another manner. Standard error processing resets $I to the principal I/O Device. Thus, the terminal at which a user signs in is called his Principal I/O Device.

Ownership of each device which a program is to use, other than the Principal I/O Device, must be established by the use of the ASSIGN Command. This command permits a program to reserve any number of I/O devices. Further, the command causes the last device number specified in its argument string to be the 'current device' by setting the $I System Variable to that device number.

Thus:

>A 3,4,5	*reserves devices 3, 4, and 5 and makes device 5 current.*

The device being current simply means that, in addition to be being owned, its identification number is stored in the $I System Variable. This $I may be referenced in any expression but can only be changed by ASSIGN. Each I/O Command is directed to the device whose number is in $I. A program may not communicate with more than one device at a time.

Table 3-1

MUMPS I/O Device Table

Number	Device	
1	Console Terminal	
2	Paper Tape Reader/Punch	
3	Line Printer	
4	Terminal #1 (Single line)	
5	Terminal #2	
6	Terminal #3	
7	Terminal #4	
8	Terminal #5	
9	Terminal #6	
10	Terminal #7	
11	Terminal #8	
12	Terminal #9	
13	Terminal #10	
14	Terminal #11	
15	Terminal #12	
16	Terminal #13	
17	Terminal #14	
18	Terminal #15	
19	Terminal #16	
20	Program Interlock #1	
.	.	
.	.	
.	.	
45	Program Interlock #26	
46	VIEW Command 'Device' (memory only)	
47	Magtape Drive #Ø	
48	Magtape Drive #1	
49	Magtape Drive #2	
50	Magtape Drive #3	
51	Reserved Device #Ø	
52	Reserved Device #1	
53	Reserved Device #2	
54	Reserved Device #3	
55	DECtape Drive #Ø	
56	DECtape Drive #1	
57	DECtape Drive #2	
58	DECtape Drive #3	
59	Sequential Disk Processor #Ø	
60	Sequential Disk Processor #1	
61	Sequential Disk Processor #2	
62	Sequential Disk Processor #3	
63	VIEW Command 'Device'	
64	DH Multiplexer #1, Term. #1	or: 2 DZ mult.* (device #1 and #2) with 8 terminals on each = 16
.	.	
.	.	
.	.	
79	DH Multiplexer #1, Term. #16	
80	DH Multiplexer #2, Term. #1	or: 2 more DZ mult.* (device #3 and #4) with 8 terminals on each = 16
.	.	
.	.	
.	.	
95	DH Multiplexer #2, Term. #16	
96	DH Multiplexer #3, Term. #1	or: 2 more DZ mult.* (device #5 and #6) with 8 terminals on each = 16
.	.	
.	.	
.	.	
111	DH Multiplexer #3, Term. #16	

*Device number assignments are based on the assumption that DH and DZ multiplexers are not combined in one system. If they are combined, DH multiplexers get the lowest device number assignments.

Table 3-1 (Cont.)

MUMPS I/O Device Table

Number	Device
112	In core job communication, transmitter for Unit 0,
113	In core job communication, receiver for Unit 0
.	.
.	.
.	.
127	In core job communication, receiver for Unit 7

When the terminal user or program issues an ASSIGN Ø command, I/O is directed to the Principal I/O Device. When MUMPS detects a program error and the user has not set the $E System Variable, the value of $I is changed to the Principal I/O device number so that error messages can be output to the terminal at which the user logged-in.

When a device is no longer required by a program, the UNASSIGN command should be used to release it for use by other programs. This command performs the reverse operation of ASSIGN. Each device specified is released and the value of the $I System Variable is changed to that of the Principal Device. However, in the case of multiple device ASSIGN-ments, UNASSIGNing a device does not automatically change the value of $I to that of the Principal Device. For example:

A␣5,3␣U␣5␣T␣"Hi"
sends "Hi" to the printer.

In the case of output devices, the command is not honored until the current output operation has terminated. Also, when a program is HALTed, or the user at the terminal ends a session, all currently owned devices are automatically UNASSIGNed.

3.4 I/O COMMANDS

The commands which effect input and output operations to the terminals and ancillary devices are: TYPE, READ, PRINT, WRITE, and LOAD. These commands may be used freely with any device except for illegal operations such as trying to READ from a Line Printer.

The TYPE command is used to output both local and global data, as well as literals, constants, and format control characters. The READ command is used to input data into local variables as well as to output text literals and format control characters. Programs can be saved and restored by the WRITE and LOAD commands, respectively. WRITE causes the program currently in the partition to be output to the device specified in the last ASSIGN command. LOAD reads the program residing on the current I/O device into the partition.

The PRINT command is used primarily to take advantage of special features of certain I/O devices, which are specified, generally, by non-printing ASCII codes. The PRINT command accepts numeric arguments, the low-order seven bits of which are taken as the decimal representation of an ASCII code. The code is transmitted without conversion to the currently assigned I/O device. Examples:

a. to ring the bell on a teleprinter, type:

    >PRINT 7

b. to output a LINE FEED without a Carriage RETURN, type:

    >PRINT 12

c. to rewind Magtape, type:

    >A 47 P 5

Refer to paragraph 3.7.1.4 for the special PRINT command ASCII codes.

3.5 OUTPUT FORMATTING

Three facilities in MUMPS provide for the formatting of output data. These are the form control characters, the ASSIGN command and the $X and $Y System Variables.

3.5.1 Form Control Characters

Three special form control characters can be used as arguments to the TYPE, READ, and PRINT commands:

#	Page Feed (FORM Feed)
!	Carriage RETURN/LINE FEED sequence (line terminator)
?nve	Horizontal TABulation - positions the PRINT mechanism 'nve' spaces from the absolute left margin.

When the characters immediately follow one another in an argument list, no intervening commas are necessary (i.e., #!?3 !!! or ## is legal)

Examples:

```
>TYPE ##?65,"PAGE ",PAG,!
```

outputs two FORM FEEDs followed by: 65 spaces, the string PAGE, the value of the variable PAG, and a Carriage RETURN/LINE FEED sequence.

```
1.40␣PRINT␣!!,7,7,">"
```

when output to a teleprinter causes two Carriage RETURN/LINE FEED sequences, followed by two rings of the bell, and a '>' character.

When using the TYPE and PRINT commands to a terminal or the line printer, each line output should be terminated by a '!' form control character to prevent overprinting on the device. When output is to a storage device (paper tape, magtape when the DOS format is selected, DECtape, or sequential disk processor) omission of a terminating '!' causes the concatenation of the line with the one which follows it.

On the paper tape punch, concatenated lines should not exceed 132 characters; otherwise, an 'LBOV' error occurs on input. The sequential disk processor (SDP), DECtape, and magtape force an EOM condition if more than 132 characters are input. Except when the "V" magtape format is selected, the next input request starts where the last terminated; thus no data is lost. Therefore, to conserve space on a tape, it may be desirable to concatenate lines.

Examples:

```
>S L1="NOW IS THE TIME FOR "
>S L2="ALL GOOD MEN"
1.20 A 55:2000 T L1,!,L2,! U 55
1.30 A 55:2000 R X1,X2 U 55
1.40 T "X1=",X1,!,"X2=",X2,!
>D 1
X1=NOW IS THE TIME FOR
X2=ALL GOOD MEN
```

the contents of both L1 and L2 are terminated by the '!' during output to DECtape (device 55) and are thus read back on input as two separate lines.

```
1.20 A 55:2000 T L1,L2,! U 55
1.30 A 55:2000 R X1,X2 U 55
1.40 T "X1=",X1,!,"X2=",X2,!
>D 1
X1=NOW IS THE TIME FOR ALL GOOD MEN
X2=next line from the input device
```

since there is no terminator between L1 and L2, they are treated as one concatenated line and on input X1 contains the original contents of L2 and L2 while X2 contains whatever line followed these on the tape.

The horizontal tabulation character '?' is particularly useful when formatting columns of data for output to terminal devices (device numbers 1, 4-19, 64-111.[1] Because tabulation is relative to the absolute left margin, each successive tabulation on a line must show an increasing number of spaces in order to effect a change in column position.

Example:

>TYPE ?10,"A",?10,"B"

results in

(*10 spaces*)AB

not

(*10 spaces*)A(*10 spaces*)B

However,

>TYPE ?10,"A",?21,"B" *produces the second result*

In any line of text, if one string overlaps the starting position for a '?nve' formatted string, the '?nve' string starts on the next available character position.

Examples:

```
>SET A=17

>TYPE A,?6,"METERS"
17    METERS
```

or

```
>SET A="SEVENTEEN␣"

>TYPE A,?6,"METERS"
SEVENTEEN METERS
```

3.5.2 Margin Control

The programmer may set a right margin for terminal devices (devices 1, 4-19, 64-111) and the line printer (device 3) when he ASSIGNs the device. When the device is UNASSIGNed, the system cancels margin control. The *MUMPS-11 Language Reference Manual* discusses the use of the ASSIGN command.

Examples:

A $I:72	*sets the right margin of the Principal I/O Device to character position 72*
A 3:80	*sets the right margin of the line printer to character position 80*

---

[1]The horizontal tabulation character can also be used to send a specific number of blanks through the in-core job communication devices, for example:

A 112 T ?5 send through 5 blanks.

3.5.3 $X and $Y System Variables

The $X and $Y System Variables provide the following information to assist the user in formatting output lines on terminal-type devices:

- $X - contains the running total of the number of characters output since a Carriage RETURN or FORM FEED on the current I/O device.
- $Y - contains the running total of the number of line feeds sent since the last Page or FORM FEED on the current output device. $Y is automatically reset to 0 when the 66th line is reached.

3.6 I/O ERROR PROCESSING

MUMPS considers DECtape, line printer, and disk I/O errors fatal. MUMPS reports an appropriate Operating System Error Message (Appendix C) on the Principal I/O Device and halts the program.[1] The programmer may choose to handle his own error processing and write a routine using the $E system variable (see 5.3).

MUMPS does not consider I/O errors associated with terminals, paper tape reader/punch, sequential disk processor, or CPU-CPU devices fatal. MUMPS reports the hardware status for the currently assigned device in the $A System Variable after each request[2]. The user must check $A after each I/O operation[3] to determine if it was successful, and if not, to determine what kind of error occurred. MUMPS does not consider line printer or magtape I/O errors fatal[4] unless the user SETs the $E System Variable. If $E is not used, the system only reports an error in the $A System Variable. If $E is used, the system generates an 'MTERR' or 'LPERR' and passes control to the user's error processing routine.

---

[1] If an error occurs during output to DECtape, MUMPS attempts to write out the contents of the current buffer before the program halts.

[2] If an error occurs during an I/O transfer to a terminal device, the current line of data being transferred is lost. If a BREAK or CTRL/C is received, the data being transferred is lost unless the programmer controls interrupts when using the $J System Variable.

[3] If the programmer wishes to check the hardware status of a device before the first I/O request (for example, to check a line printer-not-ready condition), he must use the $VIEW function to interrogate the hardware status register in memory.

[4] If an error occurs during output to magtape, MUMPS attempts to write out the contents of the current buffer before reporting the error. If an error occurs during either input or output, the system positions the tape after the block causing the error, unless an error occurs on the backspace or erase portion of a retry.

3.7 I/O DEVICE CHARACTERISTICS

The following paragraphs describe the programming characteristics of MUMPS I/O devices. Users interested in specific hardware characteristics should refer to the *PDP-11 Peripherals and Interfacing Handbook*.

3.7.1 Terminals

3.7.1.1 General Description - Terminals in MUMPS-11 systems include, but are not limited to, ASR33 Teletypes, VT05/VT50/VT52[1]/VT55 Video Terminals, and LA30/LA36 DECwriters. The system may have up to 17 remote or local single-line terminals, and up to 48 local or remote terminals on 3 DH11 multiplexer lines or 6 DZ11 multiplexer lines, or a combination of both (see Section 3.7.1.2 following). Any one of the single-line devices 4-19 may be a CPU-CPU device.

3.7.1.2 Device Numbers -

Number	Device		
1	Console Terminal		
4	Terminal #1 (single-line)		
↓	↓		
19	Terminal #16		
64	Terminal #1, DH Multiplexer #1	or	2 DZ multiplexers[2] (device #1 and #2) with 8 terminals on each.
↓	↓		
79	Terminal #16, DH Multiplexer #1		
80	Terminal #1, DH Multiplexer #2	or	2 DZ multiplexers (device #3 and #4) with 8 terminals on each.
↓	↓		
95	Terminal #16, DH Multiplexer #2		
96	Terminal #1, DH Multiplexer #3	or	2 DZ multiplexers (device #5 and #6) with 8 terminals on each.
↓	↓		
111	Terminal #16, DH Multiplexer #3		

[1]A special set of ESCape sequences can be used with the VT52. Refer to Section 3.7.1.4 for further information.
[2]The device assignments for the DZ multiplexers assume that DZ and DH multiplexers are not used in combination. When used in combination, DH multiplexers get the lowest device number assignments.

3.7.1.3 Applicable Commands -

Output

WRITE
TYPE
PRINT

Input

READ
LOAD

3.7.1.4 Special Characters and Functions - The special keyboard functions shown in the following list cannot be input as data from MUMPS-11 terminals, since these functions are used to provide the user with program and data I/O control. (Paragraph 2.4 describes in detail the operation of the control character functions.)

Function	Description
CTRL/C BREAK	Request to log-in or attempt to terminate a running program. (See the ASSIGN command in the *MUMPS-11 Language Reference Manual* for a discussion on enabling BREAK and CTRL/C.)
CTRL/U	Delete input line if it has not been terminated by a Carriage RETURN.
RUBOUT	Delete last character typed if previous character was not a Carriage RETURN.
ALT MODE *OR* Carriage RETURN	Terminates input lines from the terminal.
CTRL/O	Suppresses or restores output printing on terminal.
Line Feed	Ignored by MUMPS (not echoed).
CTRL/S	Stops printing at terminal until a CNTL/Q is hit. (CTRL/S is equivalent to X OFF on some terminals.)
CTRL/Q	Resumes printing at last character of output. (CNTL/Q is equivalent to X ON on some terminals.)

In addition to the control (CTRL) character functions, there are a special set of functions, used solely with the VT52, which use keyboard characters in combination with the ESC key. These combinations are called ESCape sequences.

The ESCape sequence is input to the computer in a different manner than the CTRL characters (see Section 2.4). The ESC key is pressed, but is then released before the associated character key is pressed.

Whenever an ESCape sequence is input, the $H system variable is set to a unique code (see Table 2-2). On input of a Carriage RETURN or an ALT MODE, $H is set to zero.

The following character codes have special meaning to the terminals and can be used with the PRINT command.

	Decimal Code	Description
Teletypes:	Ø7	BELL
	Ø9	Horizontal TAB
(not all codes have meaning to all versions of Teletypes)	1Ø	LINE FEED
	11	Vertical TAB
	12	FORM Feed
	13	Carriage RETURN
VTØ5 Displays:	Ø7	BELL (produces an audible sound)
	Ø8	Backspace (cursor left one space)
	Ø9	Horizontal TAB
	1Ø	LINE FEED
	11	Cursor Down (one line)
	13	Cursor RETURN
	14	Enter cursor addressing mode and accept the ASCII value of the next two characters as the Y and X coordinates of the new position of the cursor.[1]
	24	Cursor Right (one space)
	26	Cursor Up (one line)
	29	Home (cursor to top line, first character position)
	3Ø	Erase to end-of-line
	31	Erase to end-of-page
VT5Ø Displays:		The VT50 display has a number of device-dependent capabilities that are accessed via an ESCape sequence protocol. This information is contained in the *VT50 Programmers' Reference Manual, DEC-00-OUT5A-A-D* and in *DECscope User's Manual, EK-VT5X-OP-001.*

[1]Refer to the *VT05 Reference Manual,* DEC-00-H4AC-D for further information.

	Decimal Code	Description
VT52 Displays:		The VT52 display has a number of device-dependent capabilities that are accessed via an ESCape sequence protocol. This information is contained in the *DECscope User's Manual, EK-VT5X-OP-001.*

NOTE

See Section 3.7.1.4 for an additional set of ESCape sequences which work in conjunction with $H system variable.

(Used exclusively with the PRINT *command)*	27,61	Causes alternate mode to be entered which will generate the following sequences. Keys 0-9 will set $H to 0 to 9.
	27,62	Switches back out of alternate mode.
VT55 Displays:		The VT55 display has a number of device-dependent capabilities that are accessed via an ESCape sequence protocol. This information is contained in the *VT55 DECgraphic Scope User's Manual, EK-VT55A-TM-002.*

3.7.1.5 Error Conditions - If the currently ASSIGNed terminal is a data set, MUMPS reports its hardware status in the $A System Variable. Table 3-2 lists the bit assignments for the data set hardware status register.

Table 3-2

$A Bit Assignments for Data Set Devices[1]

First Word*		Second Word*	
Bit	Meaning when Set to 1	Bit	Meaning when Set to 1
0	Reader Enable[2]	0-1	Disconnected
1	Terminal Ready	2	Secondary Received Data[2]
2	Request to Send	3	Receiver Active
3	Secondary Transmitted Data[2]	4	Carrier Detection
4	Unused	5	Clear to Send
5	Data Set Interrupt Enable	6	Ring Indicator
6	Receiver Interrupt Enable	7	Data Set Status Change
7	Receiver Done or Ready	9-15	Unused
8-10	Unused		
11	Busy		
12	Receive Data Parity Error		
13	Framing Error[3]		
14	Data Overrun		
15	Data Error		

*The number obtained from $A is a 2-word number (i.e., a double numeric datum). Bits can be tested using procedures described in 5.1.
[1]CPU-CPU device $A bit assignments are discussed in paragraph 3.7.6.
[2]Not relevant.
[3]Treated as a BREAK character by MUMPS.

Local single-line terminals use only bits 7, 11, 15 of the first word. Multiplexer terminals also use bits 12-14 of the first word.

Remote single-line data set terminals use all assigned bits for status reporting. If the data set is disconnected, MUMPS sets bits 0-7 in the second word of $A. If the data set terminal is the currently ASSIGNed device and is disconnected without being UNASSIGNed, the program is hung. If there is no activity on the disconnected data set line for 15 seconds, MUMPS halts the program, unassigns the terminal and disconnects the data set from the telephone line.

3.7.2 Paper Tape Reader/Punch

3.7.2.1 General Description - The PC11 High Speed Reader/Punch station reads and punches folded-form 8-channel, oil-less, grey, 1-inch paper tape. Data resides on paper tape as 7-bit ASCII characters, one character per frame. Unless parity checking (even) is requested during system generation, parity is not computed and bit 8 (parity bit) is always punched. The reader/punch is programmed in the same way as are terminals.

3.7.2.2 Device Number - The reader/punch is device number 2.

3.7.2.3 Applicable Commands -

<u>Output</u>

TYPE
WRITE
PRINT

<u>Input</u>

READ
LOAD

3.7.2.4 Special Characters - CTRL/C (003) may be read from paper tape to effect job stream control and log-in.

3.7.2.5 Error Conditions - Errors are ignored on output. An error on input will force an EOM, thus ending the input.

In general, the errors generated by the reader/punch are 'not-ready' conditions: the punch is out of tape or the power is off; the reader is out of tape, off-line, or the power is off. MUMPS reports an error condition by setting bit 15 in the $A System Variable. After the programmer issues an I/O request, he may check the status as illustrated in the following example:

```
6.45 IF $A/100&327.68          (go process error)
```

The user can check for an initial 'not-ready' condition on the punch before issuing any I/O requests by $VIEWing the device's hardware status register. The address of the status register for the punch is 65388_{10}; for the reader, 65384_{10}. For both, bit 15 is set (to 1) if an error condition exists. The following command line checks the punch for a 'not-ready' condition.

```
4.34 IF $V(65388)/100&327.68   (go process error)
```

If parity checking (only even) was requested at system generation, the user may check for parity errors by testing out bit 12 in the $A System Variable after the I/O request.

```
2.10 IF $A/100&40.96           (go process error)
```

3.7.3 Line Printer

3.7.3.1 General Description - The MUMPS System operates with any one of the following four line printer models:

Type	Characters/Line	Graphic Character Set
LP11F	80	64
LP11H	80	96
LP11J	132	64
LP11K	132	96
LP11R	132	64
LP11S	132	96

3.7.3.2 Device Number - The device number for the line printer is 3.

3.7.3.3 Applicable Commands -

Output

TYPE
WRITE
PRINT

3.7.3.4 Special Characters and Functions - The following codes can be output with the PRINT command to effect format control as follows:

Code	Description
1Ø	Line Feed
12	FORM Feed *(Top of Form)*
13	Carriage RETURN

3.7.3.5 Error Conditions - When one or more of the following error conditions occur, the $E System Variable is set to a value of -0.38:

OFF LINE
OUT OF PAPER
YOKE OPEN
POWER OFF

If the $E System Variable is not being used, the programmer should first check the line printer status by using the $VIEW function to check the error bit (bit 15) of the device's Status Register. The address of the Status Register is 65356_{10}. The following command line checks this error bit.

```
8.66 IF $V(65356)/1ØØ&327.68        (go process error)
```

3.7.4 DECtape

3.7.4.1 General Description - Up to two TU56 Dual DECtape transports can be used on MUMPS systems, providing up to four logical units. DECtape is used as a linear storage device, similar to paper tape. The tape is organized into 578 contiguous blocks of 512 characters each, providing up to 295,936 characters of storage.

On output, MUMPS lines are packed into a 512-byte buffer. The buffer is output to the tape drive when it is determined that the next character of a line to be packed would exceed the size of the buffer. The null byte at the end of each line is discarded. On input, Carriage RETURNs are converted to nulls.

3.7.4.2 Device Numbers -

Number	Device
55	DECtape Unit Ø
↓	↓
58	DECtape Unit 3

3.7.4.3 Applicable Commands -

Output

TYPE
WRITE
PRINT

Input

READ
LOAD

3.7.4.4 Special Characters and Functions -

a. Positioning the DECtape:

DECtape is positioned through use of the special ':nve' syntax of the ASSIGN command. This argument specifies the address in bytes of the next character position to be read or written. The address must be a positive integer between Ø and 295,935. This address is placed in the $A System Variable. Thus, it is really $A that specifies the DECtape address. If no address is specified in an ASSIGN command, $A is not changed, and the drive is enabled for subsequent I/O transfers. $A is updated by the DECtape driver to the current address after each I/O request. The user can interrogate $A as needed.

b. Buffers

DECtape is a buffered device. A 256-word buffer is allocated on an ASSIGN and deallocated on an UNASSIGN. Further, part of the deallocation process is the writing of a partially filled buffer if the last I/O request was a WRITE, TYPE, or PRINT. Since a buffer is written out on the tape only when it is full, an UNASSIGN should be used to terminate the OUTPUT sequence to prevent the loss of data remaining in the last buffer. This same sequence is automatically performed on a HALT or a terminal error. If, however, the error originates from the DECtape unit, the last buffer may not be written out.

3.7.4.5 Errors - The 'DTERR' error message results from these error conditions:

DECTAPE OFF LINE

NOT WRITE ENABLED

UNIT NUMBER NOT SELECTED

HARDWARE ERROR ON TAPE

3.7.5 Magnetic Tape

3.7.5.1 General Description - Magnetic tape devices which are compatible with the MUMPS system are the TJU16 and the 7- or 9-track models of the TU1Ø. Up to four logical transports can be used, but TU1Øs and TJU16s cannot both be utilized on the same system. The recording mode is either industry standard 9-channel mode or 7-channel dump mode. The 800 bpi and lower densities use NRZI recording whereas 1600 bpi is phase encoded.

Labeling conventions, character sets (ASCII or EBCDIC), data formats, and physical block size are program selectable. Default values for these parameters may be modified by the MSP (modify system parameters) program. One possible default value is a DOS-11 compatible format. Thus the former MUMPS magtape handler which only permitted a DOS-11 compatible format is upwards compatible with the new MUMPS magtape handler.

Once a magnetic tape unit has been used for either input or output it must continue to be used for that same function until either the unit is UNASSIGNed or a backspace, forward space, or rewind (PRINT 1, 2, or 5) has been issued. Failure to use the unit in this manner results in an error.

3.7.5.2 Device Numbers -

Number	Device
47	Magtape Unit Ø
↓	↓
50	Magtape Unit 3

3.7.5.3 Applicable Commands -

Assignment

ASSIGN
UNASSIGN

Input

LOAD
READ

Output

PRINT
TYPE
WRITE

a. ASSIGN Command

The format of the ASSIGN command for magtape is as follows:

$$\text{Assign}\sqcup\text{nve}_1\{:\text{sve}\{:\text{nve}_3\{:\text{nve}_4\}\}\}$$

When the ASSIGN command is used to establish device ownership, the tape format parameters for subsequent I/O are established. The tape format which is used for the ASSIGN is the default format (specified by the MSP program) modified by the optional arguments of the ASSIGN command. The optional arguments in future ASSIGNs of the unit are ignored until the unit has been UNASSIGNed. The specification of a tape format which is different from the default format remains in effect only until the unit is UNASSIGNED.

Each character of the optional string (sve) represents a format switch. The effect of each of these switches is described in Table 3-3. Not all switch combinations are permissible; Table 3-4 indicates which switches are allowable.

The variable nve_3 specifies a fixed-length logical record size in bytes. Its **value** must be Ø if a fixed-length record format is not being used, and its value must be within the range of 1 through 132. The variable nve_4 specifies a physical block size in bytes. Its values may range from 14Ø through 512 but must be an even value. An odd value will give a magtape error.

Examples:

3.51 A 47	reserves magtape unit Ø with the default format
4.5 A 47:"AVL"	reserves magtape unit Ø and specifies the ANSI standard "D" format (labeled)
4.9 A 47:"EUF":8Ø:24Ø	reserves unit Ø and specifies unlabeled EBCDIC with 8Ø-character fixed-length records and 24Ø byte blocks (3 records per block)

An ASSIGN which establishes ownership of a magtape unit will poll that unit to determine its status, and that status is used for the $A reserved word. Subsequent ASSIGNs do not poll the drive, and $A then refers to the last I/O operation of the unit.

b. UNASSIGN Command

An implicit "PRINT 9" (write an EOF label) is performed if the program has been performing output. That is, the current contents of the buffer (if any) is output and then, if the unit is not on a tape mark, an EOF label is written. Note that an UNASSIGN immediately following a "PRINT 8" (write a file header label), for a non-DOS labeled format, does not perform additional output since the buffer is empty and the unit is on a tape mark.

c. PRINT Command

The control codes shown in Table 3-5 may be used as arguments to the PRINT command to effect special tape functions.

d. LOAD and READ Commands

An implicit "PRINT 7" (read a label) is performed on input if the tape is at the beginning-of-tape (BOT).

e. TYPE and WRITE

An implicit "PRINT 8" (write a header label) is performed on output if the tape is at the beginning-of-tape (BOT).

3.7.5.4 Operations and Tape Formats -

a. Normal Usage:

To output a single file to magtape, a program should simply ASSIGN the unit, issue a "PRINT 5" to rewind the tape if it is not already at the beginning of the tape, output the data, and then either UNASSIGN the unit or use "PRINT 5" to rewind the tape. A partially filled output buffer is written, and the appropriate labeling is automatically performed. The volume and header labels are automatically skipped on input, if the tape is at the beginning of tape location, and a tape mark condition indicates the end of a file's data.

b. Labels:

There are four labeling options:

1. DOS-11 compatible label

   A 7-word label appears at the beginning of the tape and a tape mark appears at the end of the file.

2. ANSI standard label

   Figure 3-1 illustrates a single and a multiple file tape using ANSI standard labels.

3. IBM standard EBCDIC label

   With the exception of minor internal field differences and the EBCDIC character set, this labeling convention is the same as the ANSI standard.

4. Unlabeled

ANSI standard labels are generated if the ASSIGN switches "L" and "A" are present (or are default switches). IBM standard labels are generated if the "L" and "E" switches are present (or are default switches). The volume identifier is "MUMPS1", and the file identifier for every file on the tape is "MUMPS.SRC". The file sequence number of the first file is 1. The sequence number is incremented by one for each subsequent file on the tape.

DOS-11 compatible labels use the file name "MUMPS.ØØ1".

c. Multiple Files:

To write multiple labeled ANSI standard or IBM standard files, the following sequence should be used.

... write file_{x-1} PRINT 9,1,8 write file_x ...

Note that if nothing is output for file_x, a subsequent UNASSIGN or rewind will not properly close that file. A "PRINT 9", however, will still properly close that null file.

To read multiple labeled ANSI standard or IBM standard files, the following sequence should be used.

... read file_{x-1}, read the tape mark PRINT 7,7 read file_x ...

d. Data Formats:

Any of three data formats can be selected.

1. Stream

   Characters are sequentially packed into blocks. On output, a carriage return/ line feed sequence is translated to a line feed, and if a line cannot fit into a block, the block is padded with NUL characters and the line is the first line of the following block. On input, line feeds, form feeds and vertical tabs are converted to EOMs[1], and carriage returns are discarded. On input, a NUL character is interpreted as the end of a block and input for that READ continues with the first character of the following block.

2. Variable Length Record

   Figure 3-2 illustrates the variable length format. Each string is preceded by a 4-byte numeric character offset whose value is the byte-length of the data string plus 4 for the offset length. If the next string plus offset cannot fit into the block, a caret (^) appears in the first character position of what would have been the next offset, and the string and its offset appear in the following block. This format is the ANSI standard "D" format, and it is selected by the ASSIGN switches "A" and "V".

   The ASSIGN switches "E" and "V" select the EBCDIC version of the ANSI standard "D" format. The EBCDIC version is the same except that every data block begins with a 4-byte numeric character block offset. This block offset equals the length of all of the strings and their offsets that are in the block plus 4 for the block offset length.

   On output, each argument of the TYPE command is treated as a separate string, and the EOM character is not output. There is no character translation, other than EBCDIC translation if that switch option was selected.

[1]End-of-message for which, internally, MUMPS uses a NUL character.

3. Fixed Length

The fixed-length data format requires the specification of record length (from 1 to 132 characters) in the ASSIGN command. Input occurs until the specified number of characters is read or until a NUL character is encountered. After a NUL character is encountered, the remaining characters in that record are skipped, and input resumes at the beginning of the next record. Output uses the same format as the stream format; there is no padding of record length. Thus, programs may output individual fields of a logical record at different points in a program. The specification of block size should be an integer multiple of the record size.

e. Status Register:

MUMPS uses the $A System Variable to communicate to the user the results of each magnetic tape operation. Table 3-6 defines the meaning of each bit when set to 1. $A represents the status of the unit at the completion of the last physical I/O operation, except when an ASSIGN command which establishes ownership of the unit is issued (see Section 3.7.5.3a). $A bit assignments for the TU16 are different from those for the TU1Ø or TSØ3.

f. Buffers:

A buffer is allocated for the magtape unit on an ASSIGN and deallocated on an UNASSIGN. Part of the deallocation process is the writing of a partially filled buffer if the last I/O request was a WRITE, TYPE, or PRINT. Since only full buffers are written on the tape, it is essential that the last request in an output sequence be an UNASSIGN or one of the special functions, "PRINT 9" (write an EOF) or "PRINT 5" (rewind) or, in the case of a DOS-11 label format, "PRINT 3" (write a tape mark). Otherwise, the contents of the last buffer will be lost. This same sequence is automatically performed on a HALT.

g. Compatibility with the Former MUMPS Magtape Handler:

Version 3 of the MUMPS magtape handler permitted only a DOS-11 compatible format. The old handler is upward compatible with the current handler through selection of the DOS-11 compatible format as the default format.

3.7.5.5 Error Conditions - No magtape errors are terminal unless the user has set the $E System Variable; they are reported back to the user in the $A System Variable. When $E is used, control transfers to the specified step or part for user-supplied error processing. If $E is not used, the user should interrogate $A after each I/O request to ensure that the request was successful.

- A logical error is a programming error which can result from an attempt to mix input and output. Once a tape has been selected for reading or writing, it must be used for the same function until either the unit is unassigned or a PRINT command argument of 1, 2, or 5 is issued.

  A logical error can also be caused by an error in the record or block size offset characters which are used in a data block which was recorded with the variable length data format.

- A tape-not-ready condition is caused by the following: the unit is not selected, the power is off, or an attempt is made to WRITE on a WRITE-protected tape. It is possible to check the latter before issuing the I/O request, as detailed in the examples below.

- The following errors usually indicate a hardware problem if they occur repeatedly: nonexistent memory, bus grant late, and cyclical redundancy.

- Bad tape and parity errors most likely indicate a physically bad tape.

- An EOF terminates an input operation.

For all errors except logical error and tape-not-ready conditions, the magtape is left positioned after the block that caused the error.

On input, the detection of an error forces an EOM and return to the user. In most cases, this results in a null string being returned; at best, only a partial string is returned.

If an error occurs during an output operation, the output buffer retains its data and may be output, after consideration of tape position and correction of the error condition, by "PRINT 4". The first successful output operation or a TYPE or WRITE, whichever occurs first, will zero the buffer.

To determine the presence of an error or EOT, $A must be interrogated after each I/O request. Each bit can be individually examined by using the masking procedure described in paragraph 5.1.

Examples for use with the TU10 drive:

- To check the tape on unit Ø for a not-ready condition:

  3.33 A 47 IF $A/100&327.68 *(go process error)*

- To check unit Ø for any error:

  22.02 A 47 IF $A/100&485.22 *(go process error)*

- To check for an EOT on unit Ø:

  3.56 A 47 IF $A/100&10.24 *(go process end-of-tape condition)*

To ensure that a tape is write-enabled, check the magtape hardware status register before issuing the output request. The address is 62800_{10} for the TU10 and 62762_{10} for the TJU16. Bit 2 for the TU10 or bit 11 for the TJU16 is set if the tape is Write-protected. Example for the TU10:

6.09 IF $V(62800)/100&.04 *(go process error)*

Table 3-3

Magtape ASSIGN Switches

Switch Character	Meaning	Effect
A	ASCII	Selects ASCII character set.
D	DOS-11 Compatible	Uses DOS-11 labeling, the ASCII character set, and the stream data format.
E	EBCDIC	Translates ASCII characters to EBCDIC on output, and EBCDIC characters to ASCII on input.
F	Fixed Length Records Data Format	Assumes fixed length logical records for input, and uses the stream data format for output. Thus, there is no automatic padding of record length on output. This switch requires the presence of an additional argument on the ASSIGN command which specifies record length.
L	Standard Labeling	Uses ANSI standard labels with the ASCII character set. Uses IBM standard EBCDIC labels with the EBCDIC character set.
S	Stream Data Format	On output, packs characters sequentially into the buffer, and CR-LF translates to LF. Strings are not split across block boundaries; instead, the buffer is padded with null bytes. On input, treats LFs, VTs, and FFs as string delimiters, and ignores CR. Thus, on output, data is effectively concatenated until terminated by a CR-LF.
U	Unlabeled	Does not provide labels. Programs that need a tape mark output at the end of a file must issue a "PRINT 3" after writing the file.
V	Variable Length Records Data Format	This data format corresponds to the ANSI standard "D" format or the EBCDIC "V" format, depending upon whether the ASCII or the EBCDIC character set has been selected. Each argument of a TYPE command corresponds to a logical record which can be read as a single argument of a READ command.

Table 3-3 (Cont.)

Magtape ASSIGN Switches

Switch Character	Meaning	Effect
digit	Density	The low order bits of the digit are used as the density bit pattern specification for the magtape unit. Note that there is no special translation to 6-bit characters for 200 or 556 bpi. For the TJU16, a "3" specifies 800 bpi, and a "4" specifies 1600 bpi.

Table 3-4

Legal ASSIGN Switch Combinations

Switch combinations denoted by an "X" are permissible.

SWITCH	A	D	E	F	L	S	U	V	digit
A	X	X		X	X	X	X	X	X
D	X	X				X			X
E			X	X	X	X	X	X	X
F	X		X	X	X		X		X
L	X		X	X	X	X		X	X
S	X	X	X		X	X	X		X
U	X		X	X		X	X	X	X
V	X		X		X		X	X	X
digit	X	X	X	X	X	X	X	X	X

Table 3-5

Magtape Control Codes

Code	Function	Effect
1	Backspace	Backspaces one record.[1]
2	Forward Space	Spaces forward one record or tape mark. If the format is DOS-11 compatible and the tape is at BOT, the label is skipped before the forward space is performed.[1]
3	Write Tape Mark	Writes a tape mark on the tape.[2]
4	Write Block	Writes out the current buffer. If the format is DOS-11 compatible and the tape is at BOT, a volume label is output before the buffer.
5	Rewind	If the last function was an output operation, an implicit "PRINT 9" is performed. The tape is then rewound.[1]
6	Read Block	Either the next block is read into the buffer or a tape mark is read. If the format is DOS-11 compatible and the tape is at BOT, the volume label is skipped before reading the block.
7	Read Label	If the DOS-11 format is selected, one block is read. If any other labeled format is selected, blocks are read until a tape mark is encountered. The buffer pointers are set to indicate that the buffer is empty. This code may be used for volume labels, file header labels, and EOF labels.
8	Write Header Label	If the DOS-11 format is selected, a DOS-11 label is output. If any other labeled format is selected, the sequence "HDR1,HDR2,tape mark" is output and that sequence is preceded by a volume label if the tape is at the beginning of tape (BOT). This code is ignored for unlabeled formats.
9	Write EOF Label	If the DOS-11 or an unlabeled format is selected, outputs a tape mark. Otherwise, outputs the sequence "tape mark, EOF1,EOF2,tape mark,tape mark".[2]

[1]The execution of this code clears the read only/write only switch. Subsequent magtape I/O establishes the new mode for that switch.

[2]If the last function was an output operation and data remains in the buffer, the contents of the buffer are written out before any other decisions are made or actions are taken.

Single File[1]

VOL,HDR1,HDR2*...data...*EOF1,EOF2**

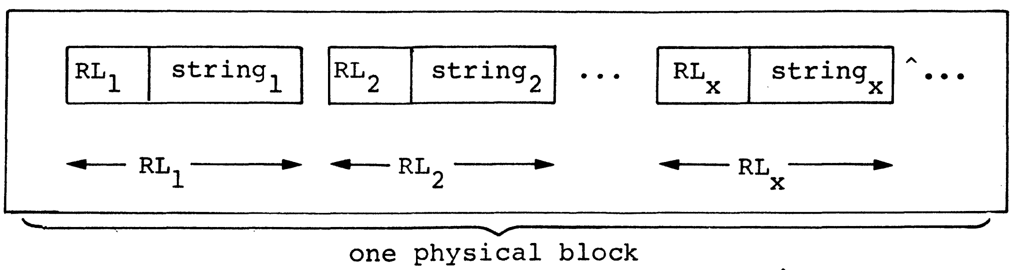
Multiple Files[1]

VOL,HDR1,HDR2*...data...*EOF1,EOF2*HDR1,HDR2*...data...
*EOF1,EOF2*HDR1...EOF2**

[1]Each asterisk represents a tape mark. VOL, HDR1, HDR2, EOF1, and EOF2 are each ANSI-specified 8Ø-character blocks.

Figure 3-1 ANSI Standard Labels

A data block for the ANSI Standard "D" Format (selected by the ASSIGN switch combination "AV...")



A data block for the IBM standard "V" Format (selected by the ASSIGN switch combination "EV...")

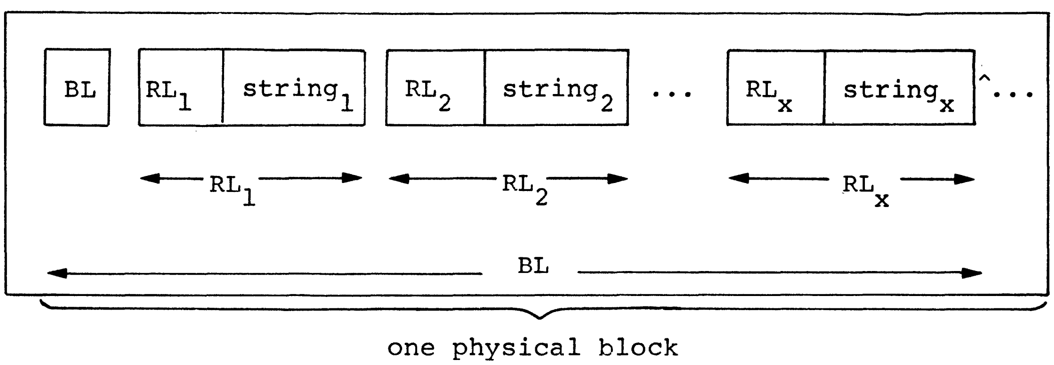


Figure 3-2 Variable Length Record Format

Table 3-6

Magtape Device $A Bit Assignments

Bit	TU1Ø Assignment	TU16 Assignment
Ø	*Logical Error[1]	*Logical error
1	-	Beginning of tape
2	Tape write protected	Tape mark
3	-	-
4	7-channel operation (Ø indicates 9-channel)	-
5	Beginning of tape	Phase encoded
6	-	-
7	*Nonexistent memory	Drive ready
8	*Bad tape error[2]	-
9	Record length error	-
1Ø	End of tape[3]	End of tape[3]
11	*Bus Grant late[2]	Tape write protected
12	*Parity error	-
13	*Cyclical redundancy[2]	-
14	Tape Mark	*Any error condition
15	*Tape not ready	-

*error condition

---

[1]A logical error is the software condition resulting from either an attempt to mix input and output or a record or block size error.

[2]The system retries 13 times before giving this error. If it is a WRITE operation, the retry attempts are made with an extended record gap.

[3]This bit is set on detection of an EOT. It stays set until a REWIND or BACKSPACE occurs. The user may continue.

3.7.6 CPU-CPU Device

3.7.6.1 General Description - The CPU-CPU device allows a MUMPS-11 program to communicate with a program running on another central processor (CPU). This device is an asychronous, half-duplex, serial communications line that connects the MUMPS-11 CPU to another CPU.

The CPU-CPU device has two operating modes or states: terminal state and message state. In terminal state the device operates exactly as if the remote CPU were a MUMPS-11 terminal with a 72-character maximum line length. In message state the device transmits and receives data as formatted messages. The state of the device is set by using the conditional syntax of the ASSIGN command:

```
A␣n:bve

where:  n = CPU-CPU device number
     :bve = True (-.Ø1) = Enter message state
            False (Ø) = Enter terminal state.
```

The default state for the device is terminal state. An UNASSIGN command directed to the device causes it to enter terminal state.

3.7.6.2 Device Numbers - A CPU-CPU device may use any one of the device numbers (4 through 19) allocated for terminal use as established by the system manager during system generation (described in the MUMPS-11 Operator's Guide).

3.7.6.3 Applicable Commands -

Output	Input
WRITE	READ
TYPE	LOAD
PRINT	

3.7.6.4 Message State Operation - In general, the CPU-CPU driver handles all communication operations, other than error detection and correction. The device does not attempt to interpret any special system characters, for example, CTRL/O or CTRL/C.

a. Data Transmission Protocol

1. The remote application program must send a complete message in the form shown in (b.) below, using a single output command. If the remote program transmits a message in several parts, the operating system may swap the program out between parts. The receiving program's message reception time interval may expire causing it to report that the remainder of the message was lost in transmission.

2. The two communicating programs must agree on the conditions that determine when each program becomes the sender or receiver. If both programs attempt to send messages simultaneously, an interlock occurs and neither program can complete a message transmission.

3. The receiving program must acknowledge the receipt of each message. A positive acknowledgment message (ACK - see below) is sent when a message from the other computer is received correctly. A negative acknowledgment message (NAK - see below) is sent if a message is not received correctly. When the sending program receives positive acknowledgment it may transmit another message. If the program receives a negative acknowledgment, it must retransmit the last message. In order to determine if a message was transmitted correctly, the remote receiving program must verify the message format and perform cyclic redundancy check (CRC) calculation. These tasks are performed automatically by the MUMPS-11 CPU-CPU device.

b. Message Formats

1. The CPU-CPU device buffer for I/O data is 80 characters (ASCII) long. The application program may, therefore, transmit messages in blocks no larger than 80 characters. The standard CPU-CPU device message format includes header and trailer information, as indicated below.

SOH	Start of Header (001)
Message Number	0-17 Octal
Character Count	Number of characters of data
STX	Start of Text (002)
Data	
.	Number of characters as specified
.	in the character count
.	
ETX	End of Text (003)
BCC1	16-bit cyclic redundancy check (CRC)
BCC2	generated from all previous characters
	in the message.

2. An acknowledgment message is in the format shown below:

SOH	Start of Header (001)
ACK, Message Number	Acknowledge Message -40_8 OR'ed with message number
-or-	
NAK, Message Number	Negative acknowledge -120_8 OR'ed with message number
BCC1 BCC2	16-bit CRC for the two previous characters

c. Calculating the Cyclic Redundancy Check (CRC)

The CRC is a method for verifying the accuracy of message data. For further information, see the subroutine which calculates the CRC in Appendix H.

d. Message Terminator

MUMPS normally interprets the character "!" as a format control character that initiates a Carriage RETURN/LINE FEED sequence on an output device. If the device is in terminal state and it encounters an "!" in an output message, it interprets the character as the standard message terminator and sends the message and an EOM (null string) to the recipient CPU. In message state, however, the device formats all output text. Each TYPE command argument initiates a new message. Thus, T␣A,B,! generates three messages; one for A, one for B and one for the !. Similarly, T␣A@B,! generates two messages; one for A@B, and one for the !. In an output message, the "!" is transmitted as a separate message. The device passes the ASCII code for Carriage Return (octal 15 or decimal 13) to the application program running on the other CPU. The programmer should not, therefore, use the "!" as a message terminator when the driver is in message state.

e. Message Transmission Count

The $X System Variable is used to report the current message number when the driver is in message state. The message count is incremented by 1 each time that a message has been successfully transmitted and acknowledged. The count can have a value in the range 0 - 17_8 (or 15_{10}). When the upper limit is reached, the count is automatically reset to 0. The user should first examine $X in order to determine if a transmission has been received correctly in message state. If its contents have been incremented by one since it was last checked, the transmission was successful. If its contents have not changed, and if bit 15 of the $A (see below) is zero, then the transmission is currently under way. If the contents of $X have not changed and if bit 15 of $A <u>is</u> set, then the transmission was not completed and bits 8 - 10 of $A will indicate the exact nature of the error.

3.7.6.5 Error Conditions - MUMPS reports the error conditions associated with a CPU device in the $A System Variable after the I/O operation is requested. Table 3-7 lists the bit assignments for $A.

The low byte of $A is meaningful only if the driver is in message state; it contains the number of unsuccessful transmissions which occurred before a successful one was achieved. This byte can provide a rough indicator of the quality of the communications link between the two CPUs.

The high byte of $A represents the driver error status byte. If the high bit of $A is not set, then no errors were encountered and the latest transmission can be assumed to have been successfully completed. If, however, the high bit is set, then the other bits in the high byte can be consulted to determine the exact nature of the error. The contents of $A remain intact until the driver processes the next I/O for that device.

Table 3-7

CPU-CPU Device $A Bit Assignments

Bits	Meaning	Explanation
0-7	Unsuccessful transmission.	Message state only.
8	Correct message too large for buffer.	Messages are limited to 72 characters in message state, and 80 characters in terminal state. Any lines longer than these maximum values will be ignored.
9	Synchronization error	In message state, the synchronization of application programs is strictly the responsibility of the users. If both sides of the communications link attempt to transmit data simultaneously, an error message is sent to the job that initiates output when input is active. In terminal state, the driver observes the usual MUMPS conventions for transmission priority (i.e., if output is active input is ignored).
10	Maximum number of retries exceeded	Bit 10 is set to indicate that a message was sent 8 times without reception of an acknowledgment (ACK). NOTE A situation in which bits 0-7 and bit 15 are set indicates that retries are still in process. A maximum time of 10 seconds elapses between retries when there is no response from the receiving end.
11-14	Unused	
15	CPU driver error indicator	Bit 15 is set whenever an error occurs in the I/O operations of the CPU driver.

3.7.6.6 Examples -

a) The following MUMPS command line would send a message to the remote CPU, wait for a response, and type the response on the system console. The CPU-CPU device number is 10 and the driver is in the message state.

```
1.01␣A␣10:-2␣T␣"TYPE IN A NUMBER"␣R␣X␣A␣1␣T␣!,X
```

b) The following command line would place the driver in terminal state and transmit the integers from 1 to 100:

```
1.01␣A␣10:0␣F␣I=1:1:100␣T␣I,!
```

c) The following example shows how one MUMPS system might sign-on another MUMPS system and start an application program.

<u>MACH. #1</u>

Contains program PG1 (listed below) under UCI #2 (EDP). The CPU-CPU device is #6.

```
0.1 ;PG1 SIGN-ON FOR CPU DEV

1.1 A 6 P 3; SEND CTRL/C
1.2 R MSG,UCI; GET TEXT
1.3 ;MSG="MUMPS-11,ETC."
1.4 ;UCI="UCI:"
1.5 T "USR:PG2",!
1.6 A 6:-.01 R TXT:60
1.7 G 2:TXT=""; ERROR IF NULL
1.8 ;ETC., ETC.

2.1 ; ERROR PROCESSOR
```

<u>MACH. #2</u>

Contains program PG2 (listed below) under UCI #4 (USR). The CPU-CPU device is #4.

```
0.1 ;PG2 START CPU-CPU I/O

1.01 A 4:-.01 T !,"HI THERE"
1.02 ;CPU-CPU INTERACTION
1.03 ;COMMENCES, RELATIVE TO
1.04 ;WHAT THE APPLICATION
1.05 ;CONVENTIONS ARE, STARTING
1.06 ;AT LINE 1.6 OR 1.7 OF
1.07 ;PG1 IN OTHER CPU
```

3.7.7 Sequential Disk Processor

3.7.7.1 General Description - The Sequential Disk Processor (SDP) allows the user to physically access the disk as an ASSIGNable sequential I/O device. The SDP can only access disk space that is explicitly set aside for its use; other disk space, including global data base, cannot be accessed. Disk space for SDP use is allocated by the SDP System Utility program described in the *MUMPS-11 Operator's Guide*. SDP allows the user to impose any file structure on his SDP disk space that he wishes to implement. Each SDP 'device' (up to 4 can be used) requires one 256-word buffer from the system's buffer pool when ASSIGNed. SDP transfers disk data in 256-word blocks to or from a buffer to permit access of any block or byte within a block.

I/O operations using the SDP are similar to DECtape I/O. The SDP anticipates two kinds of ASSIGN: explicit or implicit. The explicit ASSIGN tells the SDP driver specifically which device, which disk type and block, and which byte within the block to begin the I/O operation. The implicit ASSIGN is used subsequent to an explicit ASSIGN and assumes sequential addressing through the allocated disk space, beginning at the disk location currently pointed to for that device. Hence, if a MUMPS program using an SDP device accesses the disk in an indexed or random manner, each previous I/O operation would provide some type of key information which, in turn, would be presented as an explicit ASSIGN prior to the next I/O operation. Similarly, a MUMPS program may access the disk in a sequential manner; once the beginning disk block address had been requested in an explicit ASSIGN, each subsequent ASSIGN would be implicit, referring only to the device number, and the SDP would access blocks continuously until the MUMPS program terminated I/O, or until an attempt was made to exceed the allocated SDP space.

nve_1 = Device number (59-62)
nve_2 = Byte within block (0-511)
nve_3 = Disk block address

The Disk Block Address can be calculated by the formula

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

Where:

TYP = 0 for RK drives (RK03 or RK05)
1 for $\left\{\begin{matrix}\text{RM,}\\ \text{RF}\\ \text{RS}\end{matrix}\right.$ or $\left.\right\}$ drives (RK06, RS11, or RS04)
2 for RP drives (RP02 or RP03)
3 for RJ drives (RP04, RP05, and RP06)

UNT = Disk unit # (0-7)

BLK = Block # on unit

O-N where N = 4,799 for RK05
1,023 for RF11
2,047 for RS04
26,928 for RK06
39,999 for RP02
79,999 for RP03
170,543 for RP04 or RP05
341,086 for RP06

When a user's program issues a READ or LOAD command, SDP reads from the disk beginning at the disk block address specified in the ASSIGN command. SDP transfers data to the current user's partition line buffer until a logical EOM or until the line buffer is full. If an EOM is not detected or the line buffer is not full and the 256-word buffer boundary is reached, SDP reads the next contiguous disk block and transfer continues from byte 0 of the new 256-word block.

When the user's program issues a WRITE or TYPE command, SDP writes on the disk according to the following conventions:

- the transfer of data will exceed the size of the buffer,

- the device is UNASSIGNED,

- prior to a read, when the previous operation to the disk block was an output which did not exceed the 256-word buffer boundary.

When SDP writes a block because the output exceeds the size of the 256-word buffer, the SDP reads the next contiguous disk block into the 256-word buffer after the write, and continues the transfer into the 256-word buffer at byte Ø.

When a job requests an I/O operation the SDP transfers data beginning at the specified byte within the block and continues from that block to each contiguous disk block until:

- a null byte is detected,
- the next contiguous disk block is not allocated to SDP,
- the physical limit for one unit of the disk type being accessed is reached.

SDP requires that the MUMPS program detect these conditions and perform the logical operation desired at the MUMPS program level. This gives the MUMPS user full control of the disk access being requested and allows for the implementation of any access algorithm the user might desire.

3.7.7.2 Device Numbers - The device numbers 59-62 are reserved for use by the SDP driver. Each device number is associated with a control block, resident in memory. The disk type and unit to be accessed are determined by the number provided in the explicit ASSIGN syntax as 'nve_3'. On each explicit ASSIGN, information regarding the disk block being requested is initialized in the control block. On each implicit ASSIGN, information regarding the disk block being requested is obtained from the control block. On each UNASSIGN the information is set to zero.

Once an ASSIGN is granted, no other user is permitted to affect the user control block pointed to by the device number until ownership is released with an UNASSIGN.

Once use of a disk block is granted, no other SDP user is permitted to affect the block until the block is free again. It may become free by the accessing of another block or by UNASSIGN.

3.7.7.3 Applicable Commands -

<u>OUTPUT</u>

WRITE
TYPE

<u>INPUT</u>

READ
LOAD

3.7.7.4 Special Characters and Functions - SDP reports the disk block address of the block currently in the 256-word buffer in the $A system variable as an integer after each ASSIGN or I/O operation.

SDP reports the byte within the disk block where the next I/O will begin in the $B and $H system variables. A disk block can be pictured as two pages, each of which is 256 bytes long (Ø-255). The $B variable will report the address in the page and the $H will report the page (0 or 1). The equation ADR = $H*256+$B can be used to determine the byte address within the 512-byte (256-word) block.

3.7.7.5 Error Conditions - In the explicit ASSIGN of an SDP device, a 'MXNUM' error is generated if the 'nve_2' (byte within the block) is greater than 511 and a 'MINUM' error if the nve_2 is less than Ø. Specification of a disk block address (nve_3) which does not exist will result in a DKHER error. To test whether an existing disk block has been allocated to an SDP file, however, the $A system variable should be examined.

The $A system variable contains access code violations that occur during an ASSIGN or I/O operation. $A is set to -1 (all bits set) if the disk block address requested has not been allocated for use by the SDP. The error can occur on an explicit ASSIGN and during the logical overflow from one contiguous block to the next.

$A is set to -2 (all bits on except bit 0) if, during the contiguous access of the next disk block, the physical end of the unit is reached.

The $A access violation errors do not destroy any SDP user control block information. The MUMPS program should examine $A after each assign or I/O operation to detect the result of a requested access.

Examples:

```
2.Ø1 I $A/1ØØ&.Ø1 T !,"BLOCK NOT ALLOCATED"

2.1Ø I $A/1ØØ&.Ø2 T !,"END OF PHYSICAL DISK UNIT"
```

3.7.8 In Core Job Communication

3.7.8.1 General Description - In core job communication permits jobs to send information to other jobs without having to use the disk. Communication occurs through a series of pseudo-devices which occur in pairs; even-numbered devices are "transmitters" and odd-numbered devices are "receivers".

To send information, a job ASSIGNs a transmitter and outputs a message. Another job ASSIGNs the corresponding receiver and reads the message. Transmission occurs through an intermediate 64-character ring buffer which is permanently attached to the device pair. Transmission is fully buffered; i.e., messages may be output whether or not the corresponding receiver is ASSIGNed, and READs may be issued whether or not the corresponding transmitter has output any messages or is even ASSIGNed. Furthermore, several jobs may ASSIGN a particular transmitter, output a message, and UNASSIGN the transmitter before earlier messages are read through the receiver.

An attempt to output characters after the buffer has become full will suspend the output job until the receiver has removed one or more characters. Similarly, an attempt to read characters when the buffer is empty will temporarily suspend the input job.

The transmitter is output only and the receiver is input only. An attempt to receive input from a transmitter will result in a null string being returned; an attempt to output on a receiver will have no effect, and the job will continue as if output has occurred.

3.7.8.2 Device Numbers -

Number	Device
112	Transmitter for Unit Ø
113	Receiver for Unit Ø
114	Transmitter for Unit 1
127	Receiver for Unit 7

3.7.8.3 Applicable Commands -

Output

TYPE
WRITE
PRINT

Input

READ
LOAD

a) TYPE and READ

Every argument of the TYPE command by the output job corresponds to an argument of the READ command by the input job. The data that is received is thus an image copy of the transmitted data. For example, execution of the following code will result in A="HELLO", B="AGAIN", C=carriage return/line feed characters.

```
>A␣112␣T␣"HELLO","AGAIN",!␣A␣113␣R␣A,B,C
```

b) PRINT

In general, arguments of the PRINT command are treated the same as arguments of the TYPE command. However, a "PRINT 1" has the special control effect of resetting the buffer to an empty condition. It is thus possible for a program to insure that an input job will not receive residual data left unreceived by an earlier program. "PRINT 1" may be issued to the transmitter or the receiver of the incore job communication device.

3.7.9 DMC-11

3.7.9.1 General Description - The DMC-11 microprocessor is designed to provide for effective computer-computer communication thus permitting data base networking. Although it serves the same function as the CPU-CPU handler, it is more effective because:

- The microprocessor (DMC-11) handles all the line protocol including cyclic redundancy calculation, acknowledge/not acknowledge, retransmission, message sequencing.
- It is a direct memory address (DMA) device thus freeing up all CPU time until all messages are received/transmitted and fully checked.

A MUMPS system may not have both a CPU-CPU and a DMC-11 handler. Early experience suggests a limit of 4 DMC's due to both space and bus loading. Each DMC requires 256 words of buffer space and 128 words of scratch space in main memory. Data is exchanged a message at a time where the maximum message length is 132 characters.

3.7.9.2 Device Numbers - The DMC's will be assigned numbers in the range 4-19 and will be in sequence after DC's, KL's, DL's and non-standard devices. The actual device numbers are determined at SYSGEN time. The user must also provide vector and hardware address.

3.7.9.3 Applicable Commands

OUTPUT	INPUT
TYPE/PRINT	READ

3.7.9.4 Error Conditions - Error conditions will be reported by the system variable, $A. Error statistics can also be derived from the scratch memory used by each DMC. A precise layout of this region and the $A variable will be published in future release notes.

CHAPTER 4
LIBRARY UTILITY PROGRAMS AND GLOBALS

This chapter describes the facilities provided to the MUMPS programmer by the Library Utility Programs and Globals. Section 4.1 describes the Utility Programs; Section 4.2 discusses utilization of Library Globals.

4.1 LIBRARY UTILITY PROGRAMS

Library Utility Programs are listed in the Program Directory of the System UCI (UCI #1) and may be modified only by programmers having access to the system via that UCI and the PAC. These programs are accessible in a 'run-only' state for all users of the system, regardless of UCI. The naming convention which states that a Library Utility program name begins with the percent symbol (%) allows for the distinction between the utilities and all other programs filed under UCI #1 that are not accessible via other UCIs.

4.1.1 Features

The Library Utility Programs supplied with the MUMPS-11 system satisfy some of the basic needs of the MUMPS applications programmer[1]. Four of the programs currently provided, %T, %D, %OD, and %IO, are subroutines which can be called from other MUMPS programs. The remaining programs provide the capability for performing logical backup and restoration of program and global files as well as the ability to examine Program and Global Directory contents, and logical global structures and contents.

The Library Utility Programs and their features are summarized below.

[1]Special System Utility Programs, which are available to the System Manager and operator only, are described in the *MUMPS-11 Operator's Guide*.

Program	Description
%D	Subroutine to format the date kept in the $D System Variable for output to the currently ASSIGNed output device.
%FD	Program to provide a brief directory listing of the programs stored under the current UCI.
%GD	Programs to provide a directory listing of the globals filed under the current UCI.
%GL	Program to list the logical structure and data of specified global files.
%GR	Program to restore globals saved by %GS to the directory of the current UCI.
%GS	Program to save global files, listed in the Global Directory of the current UCI on paper tape, magtape or DECtape.
%GT	Program to list global nodes, their locations, levels, data types and contents for the current UCI.
%GU	Program to analyze a global; gives the number of nodes, total bytes, bytes per block and % utilization for each data type, for system overhead and for each free area in a global.
%GV	Program to dump a global disk block as seen by the system; prints the subscript, pointer, data type and data for each node.
%IO	Subroutine to assign a specified I/O device, if available. Upon return from this routine, an IF command with no arguments may be used to determine whether the device was assigned (TRUE) or was not assigned (FALSE).
%IU	Program to display the message "IN USE" on the currently assigned device.
%OD	Program to convert octal or decimal values to their decimal or octal equivalents.
%OP	Program to allow a terminal user to send messages to the console terminal.
%PD	Program to list the contents of the current UCI's Program Directory.
%PL	Program to restore user programs to the current UCI's Program Directory using a tape created by %PS.
%PS	Program to copy any program listed in the program directory of the current UCI to a specified output device.

Program	Description
%T	Subroutine to format the time of day kept in the $T System Variable for output to the currently ASSIGNed output device.

4.1.2 Developing and Filing Library Utility Programs

Although DIGITAL supplies a number of Library Utility Programs, each installation will probably require additional programs to suit its particular applications. Library programs are no different from other MUMPS programs except that they may use the VIEW command and $VIEW function. This means that a program must be completely debugged before incorporating it into the system, since the integrity of the entire system can be seriously affected by the careless use of VIEW and $VIEW.

Another difference is external and results from the way Library Utility Programs are named and filed. By employing a special naming convention in which the percent character (%) is always used as the first character of a Library Program name, these programs can be uniquely identified by the operating system. To file a Library Program, the System Manager simply logs-in to the system with the System UCI and the PAC, issues a PRINT␣1024 command and a 'FILE␣program name' command. The filed program is then accessible to all system users.

4.1.3 Running Utility Programs

This section describes the common operating characteristics of the various utility programs supplied with the MUMPS-11 system. Except for the %T, %D, and %IO subroutines, all Library Utility Programs are completely interactive and provide the user with complete text messages. Loading procedures are similar for each program as are the methods of error processing. By convention, Part Ø of each MUMPS System Utility Program contains a complete description of the program for user reference.

4.1.3.1 Starting Programs - There are several methods by which Library Utility Programs can be loaded and started.

a. Any user who has logged-in to the system using the Programming Access Code (PAC) may CALL a program:

><u>CALL %PD</u>	*causes the Directory Lister to be loaded and started.*

b. Programs can also be loaded using the log-in syntax:

```
CTRL/C
MUMPS-11 V3B     #6
UCI: JOC:%GL            loads and starts the Global
                        Listing Program
```

Library subroutines including %T, %D, %OD, and %IO can be called by other programs using either the CALL or the OVERLAY command. The other utility programs could also be loaded this way but except for very specific circumstances it would make little sense. For example:

```
1.03 C %D T "  " C %T T !!,"THIS PROG..."   causes the Date and Time
                                            to be output to the cur-
                                            rently ASSIGNed device,
>D 1                                        followed by the message
4 OCT 73  11:31AM                           "THIS PROG ... "

THIS PROG...
```

4.1.3.2 Stopping Programs - A program's operation can be terminated at any time by either typing CTRL/C or depressing the BREAK key, provided that this feature is enabled[1]. If this is done while the program is processing some data, or while I/O is in progress, the user can be reasonably certain that the results produced are at best incomplete. In any case, the program cannot be restarted from the point of termination and must be either reloaded or started by a Direct Mode DO Command referencing the Step or Part which begins the program.

4.1.3.3 Error Detection and Recovery - Errors which occur during program operation are typing errors, program detected errors, or system errors.

Typing Errors	- A typing error can be corrected, prior to line termination, by using RUBOUT to delete a single character or CTRL/U to delete the entire line.
Program Detected Errors	- All MUMPS Library Utility Programs perform error checking to assure the validity of user responses. When an error occurs, the program types an appropriate message and waits for the user to type a correct response.
System Detected Errors	- System error processing and messages are discussed in paragraph 2.15 and Appendix C.

[1]Refer to ASSIGN command in *MUMPS-11 Language Reference Manual*.

4.1.4 Library Utility Program Descriptions

The paragraphs which follow describe the functional operation of the Library Utility Programs. Detailed descriptions of operations are not given since all programs use similar loading and error processing techniques. Further, these programs tend to be highly interactive and normally contain lengthy text messages of explanation. Details are provided here only if the program does not provide the information.

4.1.4.1 Calendar Date Subroutine (%D) - The %D subroutine formats the representation of the calendar date contained in the $D System Variable and outputs this value to the currently ASSIGNed output device. The calling program must perform all desired page formatting (see paragraph 3.5). The date is output in the form shown below:

dd mmm yy *where:* *dd = day of month (1-31)*
mmm = name of month (JAN, FEB, etc.)
yy = year count minus 1900 (for 1974, yy = 74)

Example:

The following command line outputs the current date on the 30th line of a page, 35 spaces from the left margin:

```
34.98 TYPE # F I=1:1:29 T !
34.99 TYPE !?35 C %D
```

4.1.4.2 Fast Program Directory Lister (%FD) - The %FD program gives the user a brief (name only) listing of the programs filed under his UCI, on the currently assigned (calling) terminal.

4.1.4.3 Global Directory Lister (%GD) - The %GD program lists the names of all the globals listed in the Global Directory of the current UCI. Either the calling terminal or the line printer may be selected as the output device.

4.1.4.4 Global Lister (%GL) - The %GL program allows the contents of one or more global files, listed in the Global Directory of the calling program's UCI, to be listed on the line printer or calling terminal. The program facilitates the development and debugging of globals by providing not only a listing of the data outlined therein, but also a graphic representation of the logical structure itself.

Once the last response has been typed, the program initiates the listing operation on the specified output device. The listing output contains the subscript for each node as well as the data value. The format of the listing is shown in Figure 4-1 below.

```
-S D
--S D
---S D
----S D
          4th level of subscripting
          3rd level of subscripting
          2nd level of subscripting
          1st level of subscripting

where:

          S = subscript value at given level
          D = data value of the subscript
```

Figure 4-1 %GL Output Format

When the current listing operation is complete, %GL restarts, if the 'S' option (list selected globals) was selected, and requests another name. If no further globals are to be listed, the program can be terminated by typing a Carriage RETURN in response to the request for another global name. If the program was operating under the 'A' option (list all globals), termination automatically occurs when all globals have been listed.

For example, if the structure and contents of global A were as shown in Figure 4-2, the output listing of %GL would appear as shown in Figure 4-3.

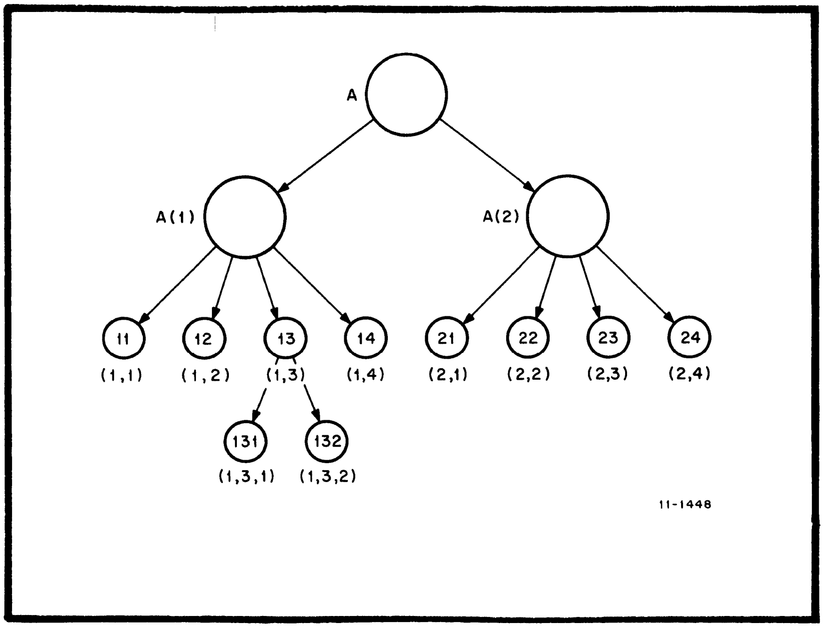


Figure 4-2 Example Global Layout

```
A.

1
- 1 11
- 2 12
- 3 13
- - 1 131
- - 2 132
- 4 14
2
- 1 21
- 2 22
- 3 23
- 4 24




END OF LISTING
```

Figure 4-3 Example of %GL Output

4.1.4.5 Global Restore (%GR) - The %GR program restores the global files residing on DECtape, magtape, or paper tape to the disk under the Global Directory of the UCI of the calling terminal. The program accepts only tapes created by the %GS program. When DECtape or magtape is used, the tape must be mounted on the selected unit of the appropriate drive. As each global file is restored, its name is output to the calling terminal.

4.1.4.6 Global Save (%GS) - The %GS program permits all or selected global files listed in the Global Directory of the calling terminal's UCI to be saved on DECtape, magtape or paper tape. If globals are to be saved on DECtape or magtape, mount the tape on an appropriate drive. The save operation begins at the beginning of the tape (address Ø for DECtape). As each global is saved its name is output to the calling terminal.

4.1.4.7 Global Trace Program (%GT) - The %GT program traces down each node of all or selected globals contained in the current UCI, and produces a listing on the line printer or currently assigned device. The listing indicates the physical level, the block address in decimal, and the contents and data type of each node, in the format illustrated by the example shown in Figure 4-4.

GLOBAL NAME? ABC

NODE	LEVEL	TYPE	BLOCK #	DATA
ABC	1	6	4194605	THIS IS A GLOBAL
1	2	6	4195359	ABC
2	3	4	4195433	
1	4	6	4195407	AGE
2	5	4	4195472	
2	6	2	4195306	NAME
2	2	6	4195359	VALUES
4	3	3	4195443	364.90
2	3	4	4195443	
4.50	4	3	4195417	832.01
3	2	4	4195359	
87	3	2	4195453	ZZZ

Figure 4-4 Sample Global Trace

The program is useful for locating the cause of a corrupted data base. If the Disk Block Tally program (described in the *MUMPS-11 Operator's Guide*) does not show any discrepancies in the disk block structure, the user may check his globals by calling this program. The block address is given in decimal so that it may be used with the $VIEW function.

4.1.4.8. Global Utilization Program (%GU) - The %GU program allows the user to analyze a global. This can be helpful in determining if a global has been designed efficiently. %GU will request a global name which should be entered without the preceding ↑ (up arrow). A listing is produced which contains the number of nodes, total bytes, bytes/block, and % utilization for system overhead, and for each data type and free area in a global. For the format of the listing, see Figure 4-5.

4.1.4.9 Global View Program (%GV) - The %GV program dumps a global disk block as seen by the system, and prints a listing containing the subscript, pointer, data type and data for each node, as shown in Figure 4-6. This program is very useful in locating the cause of a corrupted data base. When called, %GV will ask for a global reference which can be either a global node name, or the MUMPS block number in decimal.

4.1.4.10 I/O Device Assignment Subroutine (%IO) - The %IO subroutine permits the user to assign an I/O device and still retain control, even if the device is not available (unlike the operation of the ASSIGN command). Before calling %IO, the user must create a variable called %IO and assign to it the number of the device to be assigned (e.g., SET %IO=3, specifies the Line Printer). The program uses the IF command described in the *MUMPS-11 Language Reference Manual* to set an internal condition which may be tested by the programmer. If the device assignment is successful, the internal condition is set to TRUE (-.01). If the assignment can not be made (device is busy or nonexistent), the internal condition is set to FALSE (0). In either case, control is returned to the calling program. To test the status of the device assignment, the program should use an IF command without arguments or an ELSE command following the %IO call.

```
GLOBAL REFERENCE: ^SYS(1)


GLOBAL DUMP OF DISK BLOCK: 4194416
OFFSET: 258      CONTINUATION BLOCK: 0

SUBSCR.      POINTER    $D   DATA
^(1                     2    RP
^(2                     1    2
^(3                     1    0
^(4                     1    1
^(4.50                  1    0
^(5                     2    N
^(6                     2    5
^(7                     2    2
^(8                     1    0
^(9                     1    0
^(10                    2    1
^(4.01                  1    0
^(4.02                  1    0
^(4.03                  1    0
^(4.04                  1    0
^(4.05                  1    0
^(4.06                  1    0
^(4.07                  1    0
^(4.08                  1    0
^(0                     1    8
^(0.01                  1    6
^(0.02                  1    201
^(11                    2    N
^(0.05                  1    0
^(0.06                  1    0
^(0.07                  1    0
^(0.03                  1    1
^(12                    2    19
^(13                    1    32
^(200        4194719    0
^(18.10                 3    37921
^(19                    1    0
^(22                    1    0
^(20         4194729    1    0
^(21                    1    0
^(40                    1    0
^(41                    1    0
^(50                    1    0
^(51                    1    0
^(60                    1    0
^(61                    1    0
^(18.20                 3    38210
^(30         4194739    1    14
^(31                    3    48578
^(18.30                 3    49351
^(18                    3    38210

EFFICIENCY = 50%
```

Figure 4-5   Sample Global View Dump

Example:

The following program attempts to assign the paper tape punch. If the paper tape punch is busy, the program types BUSY and quits.

```
4.43 SET %IO=2 C %IO ELSE TYPE !,"BUSY" Q
```

```
C %GU

GLOBAL NAME SYS
GLOBAL UTILIZATION FOR SYS

TOTAL NUMBER OF BLOCKS 5   TOTAL NUMBER OF NODES 75


                  NO OF NODES   TOTAL BYTES   BYTES/BLOCK   % UTIL
SYSTEM OVERHEAD      2             315           63           12.30
SINGLE NUMERIC       39            78            15.60        3.04
STRING               24            256           51.20        10
DOUBLE NUMERIC       10            40            8            1.56
FLOATING POINT       0             0             0            0
FREE AREA                          1871          374.20       73.08

GLOBAL NAME
>
```

Figure 4-6 Sample Global Utilization

4.1.4.11 IN USE Message Program (%IU)- The %IU program displays the message "IN USE", followed by the date and time, on the currently ASSIGNed device. This is a means of warning other users that a seemingly idle terminal is in use.

4.1.4.12 Octal/Decimal Conversion Program (%OD) - The %OD program converts integer octal values to their decimal equivalents or converts integer decimal values to their octal equivalents. When the program starts, it checks for the existence of a variable called "%OD". If the variable is defined, its contents are taken as the number to be converted; if %OD is a string variable, the program assumes it is an octal number and reports its decimal equivalent in the %OD variable. If %OD is a numeric value, the conversion program assumes it is a decimal number and reports its octal equivalent in the %OD variable.

Examples:

9.27 S %OD=123 C %OD T %OD *(the octal number 173 is TYPEd on the currently assigned device)*

3.45 S %OD="123" C %OD T %OD *(the decimal number 83 is TYPEd on the currently assigned device)*

If the %OD variable is not defined, the program will operate interactively with the terminal user. The program prints an **asterisk (*)** to request that the user enter a number for conversion. If the number is preceded by an "O", the program assumes octal to decimal conversion. If the number is preceded by a "D", decimal to octal conversion is assumed. To stop the program, the user presses RETURN key when another value is requested.

4.1.4.13 User to Operator Communicator (%OP) - The %OP program allows a terminal user to communicate with the system operator at the console terminal (device number 1). Communication can be established only if the console terminal is not in use. To improve the readability of the messages, the program encloses messages being sent in double angle brackets (<<message>>), messages being received in triple angle brackets (<<<message>>>), and messages being sent by %OP itself in single angle brackets (<message>).

The program begins operation at the user's terminal by printing two left angle brackets (<<) to signal the user that it is ready to accept a message. Messages are sent one line at a time; each message is terminated by a Carriage RETURN. %OP notifies the user when a message is received at the console terminal by printing:

<OPERATOR NOTIFIED>

The program then waits for the operator to reply.

Messages received at the console terminal are prefaced by a preamble consisting of the current time and the calling terminal's device number. The operator can respond to a message by typing:

- a responding message
- Carriage RETURN only (null message)
- CTRL/C to terminate program operation

If the operator types a responding message, the program sends it to the user, then waits for a reply. If a null message is typed, the program sends the message:

<OPERATOR'S ANSWER WAS NULL>

to the calling terminal, then waits for the user to reply. The user can type:

- another message to continue the communication
- Carriage RETURN only, or CTRL/C to terminate communication

If the console terminal is in use, %OP sends the message:

<OPERATOR BUSY - PLEASE WAIT>

to the calling terminal, then attempts to send the message at 5-second intervals. If the console terminal remains busy after 60 seconds have elapsed, %OP sends the message:

<OPERATOR STILL BUSY - TRY LATER>

to the calling terminal, then quits.

Example:

The following example shows user/operator dialog while running %OP:

```
            User Dialog                          Operator Dialog

>                                      |                  .
>C %OP                                 |                  .
                                       |                  .
<<PLEASE MOUNT MY MAGTAPE>>            |
                                       |                  .
<OPERATOR BUSY - PLEASE WAIT>          |           operator busy
                                       |                  .
<OPERATOR STILL BUSY - TRY LATER>      |                  .
                                       |                  .
>C %OP                                 |                  .
                                       |
<<PLEASE MOUNT MY MAGTAPE>>            |   >H
                                       |
<OPERATOR BUSY - PLEASE WAIT>          |   EXIT
<OPERATOR NOTIFIED>                    |   <9:41AM  MESSAGE FROM DEVICE NUMBER 6>
                                       |   <<<PLEASE MOUNT MY MAGTAPE>>>
<<<GIVE NAME & SETUP STATUS>>>         |
                                           <<GIVE NAME AND SETUP STATUS>>
```

```
<<J. O'CONNOR - WRITE ENABLED>>
<OPERATOR NOTIFIED>

<<<OK. YOUR TAPE ON DRIVE 2>>>

<<THANKS>>
<OPERATOR NOTIFIED>

<OPERATOR'S ANSWER WAS NULL>

<<>>
```

```
<9:42AM  MESSAGE FROM DEVICE NUMBER 6>
<<<J. O'CONNOR - WRITE ENABLED>>>

<<OK. YOUR TAPE ON DRIVE 2>>

<9:42AM  MESSAGE FROM DEVICE NUMBER 6>
<<<THANKS>>>

<<>>
```

4.1.4.14 Program Directory Lister (%PD) - The %PD program lists the contents of the Program Directory of the current UCI. Either the calling terminal or the line printer can be selected as the listing device.

The directory listed output by %PD is in the format shown in Figure 4-7.

```
PROGRAMS FILED FOR YOUR UCI      DATE         TIME

PROGRAM NAME          LENGTH          DISK BLOCK

XXX                   MMMM            NNNNNNN
 .                     .                 .
 .                     .                 .
```

where:

xxx = one to three-character program name
nnn = disk block on which the program begins
mmm = program length in words

Figure 4-7 %PD Output Format

When the listing is complete, %PD exits automatically.

4.1.4.15 Program Load (%PL) - The %PL program allows programs residing on DECtape, magtape or paper tape to be loaded into the system and entered in the Program Directory of the current UCI. The program accepts only input tape which was created by the %PS program.

Before loading %PL, the user must observe the following prerequisites:

a. The user must have logged-in to the system, using the PAC.

b. If the program being input are library programs, the user must log-in to the system using the System UCI code and the PAC.

c. When DECtape is used, the starting address of the programs to be restored must be specified. This is the address that was originally specified to the %PS program when the programs were saved.

During operation, the program prints the name of each program restored at the calling terminal.

4.1.4.16 Program Save (%PS) - The %PS program allows either selected programs or all programs residing in the Program Directory of the calling program's UCI to be saved on DECtape, magtape, or paper tape, or to be listed on the terminal or line printer. Before loading %PS, the user must have logged-in to the system using the PAC. When DECtape or magtape is used the tape must be mounted on drive Ø of the tape unit.

During operation, %PS prints the name of each program on the calling terminal. If the programs are saved on DECtape, remember to save the starting address specified in response to the program's question: "WHAT IS DECTAPE STARTING ADDRESS?". This address must be used when restoring the programs to the disk via the %PL program. The %PS program does not allow multi-reel saves.

4.1.4.17 Time of Day Subroutine (%T) - The %T subroutine formats the current time of day value contained in the $T System Variable and outputs this value to the currently ASSIGNed output device. The routine does not perform any page formatting (i.e., tabulating, indenting, etc.) This must be done by the calling program's use of the standard format control characters: ! # and ?nve . The time is output in the following form:

hh:mm$\left|\begin{matrix}A\\P\end{matrix}\right|$M *where: hh = hours (Ø-12)*
mm = minutes (Ø-59)

Example:

Assume the time to be half-past four in the afternoon:

```
1.Ø6 TYPE #?2Ø,"THE TIME IS:  " C %T
```

Execution of the above line results in the output of a FORM Feed followed by a tabulation of 20 spaces from the left margin, followed by:

```
THE TIME IS:  4:3Ø PM
```

4.2 LIBRARY GLOBALS

Library Globals are like other globals in the MUMPS-11 system except that they can be read by all UCI users. Library Globals permit the MUMPS applications programmer to create common data bases for information retrieval between different UCI's and their associated application systems. Library Globals use the same naming scheme as the Library Utility Programs (i.e., % is the first character in the name) and are afforded the same type of protection (i.e., they can be modified only by the System UCI user). All globals are referenced in MUMPS commands using the up-arrow prefix (^ or ↑).

4.3 THE EDITOR

4.3.1 Introduction

The MUMPS-11 Editor supplies the user with an easy means of editing MUMPS-11 programs and globals. It enables the user to make all of the changes allowed by the MODIFY command and it also offers several other features, notably the ability to modify globals, perform searches, and change every occurrence of a particular string within a program. Prior to using the MUMPS-11 Editor, the user should LOAD the program to be modified if it is not already in core. The Editor itself may be invoked by typing the following command (all user input is underlined):

<u>>D␣^%</u>

The Editor will then output its prompting message and wait for the user to enter a command

EDIT:

The following sections describe the various editing options available. Note that entering a null command in response to the Editor's prompt will return the user to direct mode.

4.3.2 Editing Program Lines

4.3.2.1 General - The instruction for editing a single program line is of the following general form

EDIT:␣$\underline{\text{SPN}}$↵␣R␣$\left\{\begin{matrix}\text{sve}_1\\ \underline{\text{END}}\end{matrix}\right\}$↵␣W␣$\underline{\text{sve}_2}$

In the above example, SPN specifies the step to be changed, sve_1 pinpoints the characters to be replaced (R), and sve_2 defines what sve_1 will be replaced with (W).

As an example, suppose that in the following line

1.1␣T␣!"HERE IS AN EXAMPLE"

the "IS" is to be replaced with "WAS". The instruction needed to accomplish this change is

EDIT:␣$\underline{\text{1.1}}$↵␣R␣$\underline{\text{IS}}$↵␣W␣$\underline{\text{WAS}}$↵
1.1␣T␣!"HERE␣WAS␣AN␣EXAMPLE"
EDIT:

Note that after any editing command has been executed the MUMPS-11 Editor echoes the new line and outputs its prompting message.

The user may delete characters or add new characters to the beginning of a line by omitting sve_1 or sve_2. Specifically, if sve_1 is null, sve_2 will be inserted at the beginning of the specified step; if sve_2 is null, sve_1 will be deleted from the step. The following is an example of how this can be done.

```
2.8␣T␣!"CHANGE␣CHANGE␣THIS␣LINE"
EDIT:␣2.8↵␣R␣↵␣W␣I␣X↵
2.8␣I␣X␣T␣!"CHANGE␣CHANGE␣THIS␣LINE"
EDIT:␣2.8↵␣R␣CHANGE␣↵␣W␣↵
2.8␣I␣X␣T␣!"CHANGE␣THIS␣LINE"
EDIT:
```

The user may also wish to add a string of characters to the end of a line. The END feature may be used to do this:

```
.
.
.
.
18.35   T !,X,Y,"␣IS"
EDIT:   18.35↵ R  END  W ,A,B↵
18.35   T  !,X,Y,"␣IS",A,B
EDIT:
```

4.3.2.2 The Dot.Dot.Dot Feature - The dot.dot.dot feature of the MUMPS-11 Editor allows the user to modify lines with a minimum amount of typing. The command which incorporates this feature is of the form

$$\text{EDIT:␣}\underline{\text{SPN}}\text{↵␣R␣}\left\{\begin{array}{l} S_1\ldots S_n \\ S_1\ldots \\ \ldots S_n \end{array}\right\}\text{↵␣W␣}\underline{\text{sve}_2}\text{↵}$$

The Editor will insert sve_2 in place of the string of characters from the first occurrence of S_1 to the subsequent first occurrence of S_n.

To remove the SET clause from the step

```
3.1␣S␣X=$E(Y,A,10)␣I␣↑A(3,4)␣G␣4
```

the user might do the following

```
EDIT:␣3.1↵␣R␣S␣X=$E(Y,A,10)␣I↵␣W␣I↵
3.1␣I␣↑A(3,4)␣G␣4
EDIT
```

However, considerable typing time, and possibly typing errors, could be avoided by the use of the dot.dot.dot feature:

```
EDIT:␣3.1↵␣R␣S...I↵␣W␣I↵
3.1␣I␣↑A(3.4)␣G␣4
EDIT:
```

Either S_1 or S_2 or both may be of length greater than one. Utilizing this fact can insure that the desired string is the one modified. For example, suppose the user wished to change the line

```
4.1␣T␣!"THIS␣TESTS␣THE␣TERMINAL",X,Y
```

to

```
4.1␣T␣!"THIS IS RIGHT",X,Y
```

the command

```
EDIT:␣4.1↵␣R␣T...L↵␣W␣IS␣RIGHT↵
```

would cause the line to read

```
4.1␣IS␣RIGHT",X,Y
```

This could be prevented by instead typing

```
EDIT:␣4.1↵␣R␣TE...L␣W␣IS␣RIGHT↵
4.1  T␣!"THIS IS RIGHT",X,Y
EDIT:
```

The user also has the option of omitting either S_1 or S_n. If no starting point (S_1) is indicated, the beginning of the line is assumed:

```
  .
  .
  .
5.3  S  X=3, Y=$E(Z,1,4) G4
EDIT:␣5.3↵␣R␣...3,↵␣W␣I␣ ↵
5.3  I␣Y=$E(Z,1,4) G 4
EDIT:
```

Likewise, omitting the ending point (S_n) implies that the end of the line should be assumed:

```
  .
  .
  .
7.9  S  M=N*P,Y=$E(Z,1,M) D 9
EDIT:␣7.9↵␣R␣$...↵␣W␣A+B␣G␣9↵
7.9  S  M=N*P,Y=A+B G 9
EDIT:
```

As in the general editing command, omitting sve_2 causes the string $S_1,,,S_n$ to be deleted.

```
.
.
.
6.45␣S␣X=Y+Z,R=S*Y␣T␣!"THE RESULTING ANSWER IS",(X+R)
EDIT:␣6.45↵␣R␣RE...␣↵␣W␣↵
6.45␣S␣X=Y+Z,R=S*Y␣T␣!"THE␣ANSWER␣IS",(X+R)
```

4.3.2.3 The AGAIN Feature - Frequently, several changes may have to be made to the same line. The AGAIN option saves the user from repeatedly having to retype the step number of a line undergoing multiple changes. After the first modification has been made, an "A" may be typed in place of the step number. The Editor prints the step number before outputting its normal replace (R) request. As an example:

```
.
.
.
19.27␣T␣SUM␣I␣R␣Q␣14.97
EDIT:␣19.27↵␣R␣I↵␣W␣D↵
19.27␣T␣SUM␣D␣R␣Q␣14.97
EDIT:␣A↵␣19.27␣R␣Q↵␣W␣G↵
19.27␣T␣SUM␣D␣R␣G␣14.97
EDIT:
```

4.3.2.4 The SEARCH Feature - By responding to the Editor's prompt with an S, the user can direct the Editor to search for all occurrences of a particular string of characters. If the Search feature has been specified, the Editor outputs a prompt (SEARCH FOR:) for the desired string, performs the search, and prints all lines in which the string occurred. As an example, consider the following:

```
EDIT:␣S↵
SEARCH FOR:␣^QRS↵
1.4   S␣↑QRS(Q)=Q+R*S
2.6   S␣R=R+1␣I␣↑QRS(R)␣G␣7.2
8.41  K␣↑QRS
```

4.3.2.5 The CHANGE EVERY Feature - The MUMPS-11 Editor gives the user the capability of changing all occurrences of a particular string within all or part of a program. To utilize this feature, the user should first type a "C" in response to the Editor's prompt.

```
EDIT:␣C↵
CHANGE EVERY:␣SVE1↵␣TO:␣SVE2↵
FROM LINE:␣SPN1↵␣THROUGH␣LINE:␣SPN2↵
```

In the interchange that follows, the user must specify the string to be changed (sve_1), the string which is to replace it (sve_2), and the range of lines (spn_1 through spn_2) over which the change should be made. Note that if spn_1 is null, the modifications will begin at the first line in the program; if spn_2 is null, the Editor will continue through the end of the program. The Editor will echo each line that is modified. Suppose that in a program all references to program TST are to be changed to refer to program NEW:

```
EDIT:␣C↵
CHANGE EVERY:␣TST↵␣TO:␣NEW↵
FROM LINE:␣↵␣THROUGH LINE:␣↵
3.2  S X=Ø,Y=A+B C NEW
4.98  O NEW:3
```

4.3.2.6 The RE-NUMBER Feature - By responding to the Editor's prompt with an R, the user can direct the Editor to renumber a line. The Editor outputs a prompt (LINE TO BE RENUMBERED:), and the user responds with a valid line number (spn_1). The Editor then outputs the prompt (NEW LINE NUMBER:) and the user responds with a second line number (spn_2) that does not currently exist in the program. The line specified by spn_1 is then renumbered to spn_2 and the old spn_1 is erased.

```
EDIT: ␣R↵
LINE TO BE RE-NUMBERED: ␣spn1↵
NEW LINE NUMBER: ␣spn2↵
```

4.3.3 Editing Globals

The MUMPS-11 Editor can also be used to edit globals consisting of string data. The procedure for editing global data is the same as the procedure for editing a single program line except that a full level global reference, rather than a step number, is the proper response to the computer's "EDIT:" prompt. Note that the "END", "AGAIN", and the "dot.dot.dot" features of the editor apply both to program steps and global nodes (see sections 4.3.2.1, 4.3.2.2, and 4.3.2.3).

As an example of editing a global node, assume the following global is defined:

```
^LDP(3)="TOWARE␣ED␣BLDG"
```

This node may be edited in the following manner:

```
EDIT:␣^LDP(3)↵ ␣R␣ARE␣ED↵␣W␣ER␣        ↵
 ^LDP(3)="TOWER␣BLDG"
EDIT:␣A↵␣^LDP(3)␣R␣...G↵␣W␣ING␣INFERNO↵
 ^LDP(3)="TOWERING␣INFERNO"
EDIT:␣A↵␣^LDP(3)␣R␣...T↵␣W␣THE␣T↵
 ^LDP(3)="THE␣TOWERING␣INFERNO"
```

4.3.4 Entering MUMPS-11 Commands from the Editor

MUMPS-11 commands may also be executed from the Editor. This capability allows the programmer to perform calculations, examine and modify local and global data, and execute almost any command which could be invoked in direct mode, without having to leave the Editor. As an example, consider the following:

```
>D␣^%↵
EDIT:␣L␣ABC␣W␣3↵
3.1  S X=3 K DG
EDIT:␣3.1↵␣R␣3...D↵␣W␣3,Y=C␣K␣AB,D↵
3.1␣S␣X=3,Y=C␣K␣AB,DG
EDIT:␣F␣ABC↵
EDIT:↵
>
```

4.3.5 Summary of Editor Questions

USER	SYSTEM	MEANING	LEGAL RESPONSE
D↑%↵	EDIT:	MUMPS-11 Editor invoked	1. Null entry, exits to direct mode. 2. Line Number for editing a line. 3. Global Node. 4. "S" for Search. 5. "C" for Change Every. 6. "R" for Renumber. 7. "A" for Again, which repeats previous EDIT. 8. A valid line of MUMPS code.
Step Number or Global↵ Node	R	Replace string that follows R	1. A string of characters derived from the program or global node being edited. 2. A string containing three consecutive dots; e.g., A...B or C...or ...D.

USER	SYSTEM	MEANING	LEGAL RESPONSE
			3. The word END. 4. A null entry; causes insertion at the beginning of the line.
	W	String to replace R string	1. A string of characters. 2. A null entry; causes the R string to be deleted.
S ↵	SEARCH FOR:	entry will be searched for	A string of characters.
C ↵	CHANGE EVERY:	All occurrences of entry will be changed....	A string of characters.
	TO:	...to the following	A string of characters.
	FROM LINE:	Lower limit of CHANGE EVERY question	1. A Step Number. 2. A null entry; causes 1st Step to be assumed.
	THROUGH LINE:	Upper limit of CHANGE EVERY question	1. A Step Number. 2. A null entry; causes last Step to be assumed.
R ↵	LINE TO BE RENUMBERED:	This line wil will be renumbered....	A valid Step Number.
	NEW LINE	with this line.	A valid Step Number.
A ↵	Previous Step # or Previous Global Node R	Allows re-edit of previous Step	Any response used for R.

ERROR MESSAGES	MEANING
1. ??	A. Invalid step number. B. The step does not contain the characters which were to be replaced. C. A global node was specified that does not have defined data.
2. TOO LONG	If editing had taken place, result would have exceeded the maximum string length.

ERROR MESSAGES	MEANING
3. THE GLOBAL (g) IS EQUAL TO (n) TO EDIT, USE THE SET COMMAND where: g = global name n = numeric quantity to which global was set equal	This message will be output if an attempt is made to edit a global containing numeric data.

CHAPTER 5
PROGRAMMING TECHNIQUES

This chapter provides the MUMPS programmer with supplemental information on several elements of the MUMPS Language. Sections 5.1 through 5.5 discuss particular elements of the language, including the $J and $E System Variables, the VIEW command and $VIEW function, and a method for retrieving global data. Sections 5.6 and 5.7 discuss two considerations for program design: debugging and program size. The last sections are concerned with the use and design of globals.

5.1 MASKING

The programmer may extract individual bits of a word in storage by using the AND operator (&), which performs true Boolean AND. Table 5-1 lists the mask value for each bit.

Table 5-1
Bit Mask Values

Bit	Mask	Bit	Mask	Bit	Mask
0	.01	11	20.48	22	41943.04
1	.02	12	40.96	23	83886.08
2	.04	13	81.92	24	167772.16
3	.08	14	163.84	25	335544.32
4	.16	15	327.68	26	671088.64
5	.32	16	655.36	27	1342177.28
6	.64	17	1310.72	28	2684354.56
7	1.28	18	2621.44	29	5368709.12
8	2.56	19	5242.88	30	10737418.24
9	5.12	20	10485.76	31	21474836.48
10	10.24	21	20971.52		

If the bit to be examined is in a word stored as a MUMPS number, the programmer simply ANDs the word with the appropriate mask value for that bit. For example, the following MUMPS command returns a True (-0.01) or False (0) result, depending on whether bit 4 of the $J System Variable contains a 1 or a 0 value.

```
IF $J&.16
```

If the bit to be examined is in a word *not* stored as a MUMPS number, the word must first be divided by 100, (e.g., values obtained from the $A System Variable or from use of the $VIEW function). For example, the following command return a True or False result, depending on whether bit 15 in $A contains a 1 or 0 value.

```
I $A/100&327.68
```

To prepare a location for bit manipulation, first divide by 100 (if the word is not stored as a MUMPS number) as follows:

```
$V($V(44)+46)/100
```

To shift right (end off), divide by corresponding powers of 2:[1]

2^x where x is bit positions to shift

2^1 =2 shift 1 bit

2^2 =4 shift 2 bits

2^8 =256 shift 8 bits (high byte to low byte)

I.e.,

```
$V($V(44)+46)/(100*256);  shift high byte into low byte
$V($V(44)+46)*256;        shift left 8 bits
```

To mask, logically "AND" or "OR" using standard MUMPS masks. For example:

```
$V($V(44)+46)/100&10.24;            check if bit 10 set
```

Add masks together to check multiple bits:

```
$V($V(44)+46)/100&(5.12+10.24);  check if bits 9 & 10 set

$V($V(44)+46)/100&2.55;          check for any bit set in low
                                 byte
```

Combining operations:

```
$V($V(44)+46)/25600&2.55;        shift high byte to low byte
                                 and check for any bit set in
                                 that byte
```

[1]To shift left (zero fill), multiply by corresponding powers of 2.

The following routine checks the status of any given bit in the $A System Variable by shifting the contents of the word to the right (dividing by 2) for successive bit positions. The selected bit position is determined by the value SET to N.

```
Given:      S N=15

4.1 S A=$A/100
4.2 F I=0:1:N S A=A/2
4.3 I A&.01  (go process error)
4.4 (otherwise, continue I/O operations)
```

5.2 $J SYSTEM VARIABLE

The $J System Variable is available to the MUMPS programmer as a job status word for special applications checking. $J is stored as a MUMPS number and contains three bits which are of interest to the programmer. Table 5-2 describes these assignments.

Table 5-2

$J Bit Assignments

Bit	Meaning When Set to 1
2	Recognition of CTRL/C and BREAK for currently ASSIGNed terminals is enabled.
3	A CTRL/C or BREAK has been received on the Principal I/O Device (cleared when the ASSIGN 0:bve syntax is invoked.)
4	Timed READ or LOCK overrun.

5.2.1 CTRL/C and BREAK Recognition

Bits 2 and 3 of the $J System Variable allow a programmer to control processing of CTRL/C and BREAK, to determine the status of a particular job. For example, if the user logged-in simply to run a program, CTRL/C and BREAK are normally disabled. The program itself may check bit 2 to determine if this is the case, and enable them if it is necessary. In another situation, a program might disable CTRL/C and BREAK during a particular processing operation and use bit 3 to monitor attempts to use CTRL/C or BREAK. The program could notify the operator that special processing is taking place.

5.2.2 Timed READ or LOCK Overrun

The programmer may check the value of bit 4 in $J for a timed READ or a LOCK overrun.

Bit 4 is set when the specified interval in the READ command expires and either

- no response was detected, or
- input was detected, but the user took more than the specified interval to type another character or a line terminator.

In either case, MUMPS returns a null string to the program. A timed READ in which the user successfully entered the data, or a non-timed READ, would clear bit 4.

Example:

Assume that device number 17 is a data set terminal, and the user wishes to discontinue processing if the terminal is disconnected on a request for input.

```
4.1 A 17 R !"ARE YOU READY?",!,X:3ØØ`Q:$J&.16
4.2 (process data from remote terminal)
```

Bit 4 can also be set if the optional timeout argument is present on a LOCK command argument. If the specified number of seconds pass and the system has not "locked" the requested argument, then bit 4 is set and the program continues execution. If the system had succeeded in "locking" the argument, or if there was no time switch on the LOCK, bit 4 would be cleared.

Example:

```
4.1␣L␣↑A(1,2,3):2␣I␣$J&.16␣T !"NOT AVAILABLE"!␣G␣3
```

5.3 WRITING ERROR PROCESSING ROUTINES

The programmer may write his own error-processing routines using the $E System Variable. MUMPS normally considers all errors listed in Appendix C[1] fatal. If one of these errors occurs, MUMPS enters an error index (in the range 0 to -.38) in the $E System Variable and reports the error on the Principal I/O Device.

[1]Except GARB errors. MUMPS reports MTERR or LPERR only if the user sets $E.

If the programmer has set the $E System Variable to a Step or Part number and an error occurs, the system:

- sets the $W System Variable to the value of $L when the error occurred
- sets the $E System Variable to the error index for the error encountered (in the rage 0 to -.38)
- transfers program control to the step or part number referenced by $E
- resets the user stack, causing currently active DO, CALL and FOR commands to be lost.

The programmer examines $E to determine which error occurred and examines $W to determine where the error occurred. Note that the programmer must reset $E to a Step or Part number to control further error processing in his program; otherwise, error processing reverts to system control.

If $E contains any negative MUMPS number or Ø, MUMPS performs its standard error processing when an error occurs (see 2.15). In this manner, the programmer may "turn off" applications error processing.

EXAMPLE 1

A programmer wishes to output explicit error messages for his application programs. He creates an error message program named 'ERR' with 38 parts, one part for each error generated by MUMPS. The error processing routine in his application program calls the part of 'ERR' which corresponds to the error index in $E. For example, part number 15 corresponds to the error index -.15, which indicates the illegal use of a MUMPS command.

```
'PRG'

1.10 S $E=2
1.20    .
1.30    .
1.40    .          Suppose this line contains
1.50               an invalid command

2.10 C ERR:-$E*100

'ERR':

0.01 ; ERROR PROCESSOR

  .
  .
  .
15.10 A 0 T !"SORRY, BUT LINE",$W," CONTAINS AN INVALID COMMAND!",!!
```

EXAMPLE 2

In this case, the programmer wishes to control error processing only when a DECtape error occurs.

```
3.1 S $E=7
3.2 A 56:3166 T "LABEL" S $E=0
3.3 (continue I/O processing)

7.1 S X=$E,Y=$W G Y:X<>-.3 A 0 T !"UNIT NOT READY!!",!
7.2 T "CHECK IF OFF-LINE OR NOT WRITE ENABLED.",!
7.3 T "IF NEITHER TYPE 'N' TO HALT PROGRAM; REPORT HARDWARE MALFUNCTION'
7.4 R "OTHERWISE, CORRECT AND TYPE 'Y' <CR>",!,Z I Z="Y" G Y
7.5 H
```

5.4 VIEW COMMAND AND $VIEW FUNCTION PROTECTION

Several levels of protection are applied to the use of the VIEW command and $VIEW function. The highest level of protection is applied to the VIEW command when used to write into internal memory or onto disk. The lowest level is applied to $VIEW, which can only *read* memory locations.

The following protection applies to the VIEW command:

- Using VIEW to write to disk or memory:

    a. The user must be logged-in under the System UCI.

    OR

    The command must be executed from a Library Utility Program (see Chapter 4).

    b. For writing memory, the device number ASSIGNed must be either 63 or 46. For writing to the disk, device number 63 must be ASSIGNed.

    c. The "PRINT␣256" command must be issued prior to using VIEW.

- Using $VIEW to read from a disk or using $VIEW to read from memory:

    a. The user must be logged-in under the System UCI

OR

The command must be executed from a Library Utility Program (see Chapter 4).

b. For reading memory, the device number ASSIGNed must be either 63 or 46. For writing to the disk, device number 63 must be ASSIGNed.

NOTE

The VIEW command is not protected for addresses residing in the 'external memory page' of the PDP-11 (i.e., an address greater than or equal to 160000_8 or 58444_{10}).

The $VIEW Function can be used under the following conditions:

- The user must be logged-in under the System UCI

OR

- The command must be executed from a Library Utility Program (see Chapter 4).

5.5 USE OF THE IF COMMAND AND INDIRECTION SYNTAX TO RETRIEVE GLOBAL DATA

This section outlines the elements of a program designed to retrieve global data. The programmer may wish to write a similar program if the application system occasionally requires an unusual type of data search. A program of this type does not, however, retrieve data quickly. It may be useful in a particular application situation only because the programmer does not have to spend time writing an efficient program to search for the needed data.

The retrieval program first issues a "READ␣X" command so that the user may enter the parameters on which to base the selection. The user's input entirely determines the record selection. A record usually has several data fields which may serve as the parameters. The user input is in the form of a parameter name, followed by a Boolean condition and the parameter value, followed by any other parameters. For example, a patient record may include the patient's age, sex, blood type and activity code. In an unusual situation, the user may need to know the name of all female patients over 30 who have blood type AB and are going into surgery. The input line might be:

parameters

AGE=>30&(SEX="F"&(TYP="AB"&(ACT="S"))) *=X*

Boolean Condition

parameter value

successive conditions are enclosed in parentheses as nested Boolean functions

OR, since all subsequent parameters are "AND" conditions, the input line could be:

AGE=>30,SEX="F",TYPE="AB",ACT="S"

where the commas are implied "ANDs" for the IF command (see below)

The program then enters a loop which gets a record and sets up the local variables to compare with the user-selected parameters. For example:

1.3 S N=17538	
1.4 S N=N+1 I N=17968 Q	*-this line gets records 17,539 through 17,968*
1.5 S AGE=↑AGE(1,N),SEX=↑SEX(1,N),...	*-this line sets up the local variables.*

The program next compares the contents of the local variables with the parameters stored in the local variable 'X' by using the IF command and indirection syntax. When the indirection syntax is used to determine the argument of an IF command, the execution of commands on the remainder of the line is not dependent on the logical result of the IF. To test the result of the IF, the programmer must use the ELSE command. For example:

1.6 I ←X ELSE GOTO 1.4	*if the record does not match, go get another record*

If the selected record *does* fit the parameters, the lines following the IF command lines process the record, and then return control to the Step or Part which gets another record.

5.6 DEBUGGING PROGRAMS

Often the programmer may wish to interrupt the operation of his program at predetermined points to examine his program data in detail. After examination, he may wish to resume the normal sequence of operation from the point of interruption. This technique is especially useful in the early steps of program debugging.

The BREAK command can be inserted anywhere in a stored program. Upon execution of the BREAK, the program is suspended and a '?' is printed, followed by the Step number where the BREAK was found, followed by the message 'BREAK'. The programmer has the option of examining and changing the contents of variables or of writing out all or part of the program. The program itself may not be changed until after a CTRL/C has been typed to remove the program from the BREAK state. If the programmer attempts to modify the program in the BREAK state a PROT error is generated.

The GO command causes program execution to resume from the point where the BREAK occurred.

If an error occurs while in the BREAK state or if CTRL/C is typed, continuation is not possible using the GO command, since system error processing removes the BREAK state from the user's partition.

5.7 PROGRAM SIZE CONSIDERATIONS

After System Generation, the amount of core space allocated to the user's partition is fixed and therefore is the critical factor which limits the size of a core resident program. Each MUMPS program must fit into the Program Buffer of the partition in which it is to run[1].

The following paragraphs discuss the factors which affect the size of MUMPS programs and the techniques to use the available internal memory efficiently.

[1]During system generation, the System Manager/Operator may specify a standard partition size and one or more non-standard sizes. The programmer may select the partition size in which a program is to run by using the optional syntax of the START command.

As mentioned earlier, during execution MUMPS programs reside in the Program Buffer which is part of the user's partition. Since the size of each partition is fixed, so is the amount of space available for a program. This significant factor limits the size of an executable program. Although the user does not have the ability to increase his partition size, there are a number of programming techniques to efficiently utilize available space. If a program has reached an irreducible size and is still too big, the user must consider segmentation, using the CALL and OVERLAY commands.

5.7.1 Conserving Available Space

The ultimate size of a program depends on: (1) the actual number of characters in the program, (2) the number of local variables, (3) the amount of data stored in these variables, (4) the manner in which these various elements are used with respect to one another.

In order to create programs that make efficient use of space, the programmer should understand the basic structure of his partition and the dynamic nature of some of its component parts. Figure 5-1 shows a simplified diagram of a MUMPS partition.

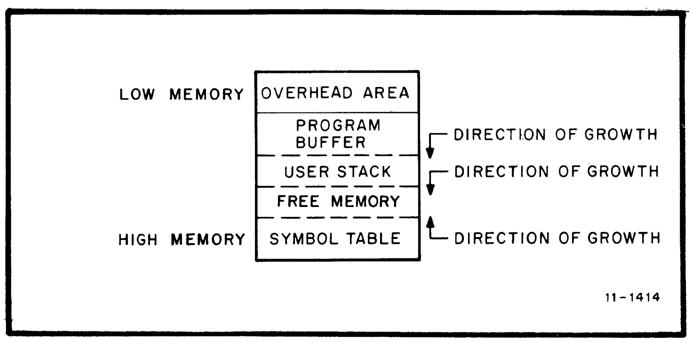


Figure 5-1 Basic Partition Layout

There are three areas in the user's partition which compete for free memory: the Program Buffer, the User Stack, and the Symbol Table. The Program Buffer is the area which contains the actual MUMPS-11 program. The User Stack contains transient information used by the operating system in processing the user's program. The Symbol Table is the area where all locally defined symbols reside. All three of these areas in the partition are dynamic; they grow and shrink in response to program operation. Both the Program

Buffer and the User Stack grow toward high memory, and the Symbol Table grows from the upper limit of the partition toward low memory. The area between the top of the Symbol Table and the bottom of the User Stack is free memory which can be utilized by any of these areas. Specific factors which directly affect the amount of storage required for each of these areas are discussed below.

The size of the Program Buffer increases and decreases, within the limits of available free memory, in response to LOAD, CALL, and OVERLAY commands. Since all elements of a MUMPS program are brought into the Program Buffer, the size of a program can directly depend on the way it is constructed.

a. To reduce the space required for a program:

- Abbreviate all commands and function calls to the legal limit.
- Omit leading and trailing zeroes from all numeric strings.
- Keep program names, variable names, and comment lines as short as possible.
- Put as many commands on a line as possible to reduce the total number of individual Step Numbers which must be stored.
- Do not duplicate arguments or literals unnecessarily. Whenever an argument string or literal is used more than twice in a program, store it in a local variable and make multiple references to it (use the Indirection Syntax Operator in the case of arguments).

b. To conserve User Stack space:

- Avoid deep nesting of DO, CALL, and FOR commands by employing a different algorithm.
- Use the XKILL command judiciously; use KILL instead, if possible.

c. To save Symbol Table space:

- If time is less important than available space, store variables in Global Files.

- If a large number of local variables are required, it may advantageous to concatenate them into one large string for storage as one variable and to extract each via the $Extract Function when needed.

5.7.2 Segmenting Programs to Conserve Space

The amount of space allocated to the partitions in which programs run is fixed in size, and each program must fit into this space. However, through the use of the CALL and OVERLAY commands, a program can cause the loading and execution of other programs in the UCI's Program Directory as well as Library Utility Programs filed in the System UCI's Program Directory. When a program is brought in by either CALL or OVERLAY, it replaces the invoking program and, unless otherwise specified, execution begins at the first non-zero part. A program accessed in such a way is treated like any other program; it may CALL or OVERLAY still more programs. This allows the effective size of a program to be extended indefinitely. However, because local variables remain intact on execution of a CALL or OVERLAY, a size problem may exist if the incoming program is larger than the one invoking it. While developing a program, the user may want to CALL another FILEd program (OVERLAY can only be used in Indirect Mode). The programmer must be sure that he has FILEd the program currently in his partition before using CALL; otherwise the original program is destroyed.

5.8 GLOBAL ACCESS CONSIDERATIONS

The scheduling algorithm used by the system's Executive (paragraph 1.3.1.1) is designed to give jobs performing Global accesses (disk I/O) preference over other jobs in the system. When a disk input/output task has completed, the requesting job is given an additional short time slice to process its next command. During this time slice the job retains control of the disk. This enables the job to perform another disk access, if desired, without having to wait for other jobs to complete their disk tasks. The programmer should exercise care when using this feature, since it is possible, through faulty programming techniques, to completely lock-out other MUMPS jobs from use of the disk. This prevents other programs from accessing Globals, and terminal users from filing and loading programs. In particular, a high frequency of disk I/O requests or interminable program loops containing disk I/O requests should be avoided.

The programmer must assiduously avoid the creation of interminable loops which request disk input/output or lengthy disk I/O functions such as result from many global accesses.

a. `1.2 G $L:$D(↑A(I)) H Ø`

The above command line causes an interminable disk access loop if ↑A(I) is defined. The 'HANG␣Ø', intended to effect a job swap-out will not be executed. The correct procedure is shown below.

b. `1.2 H Ø G $L:$D(↑A(I))`

This command line will cause the job to be swapped-out prior to each Global reference, thereby allowing other jobs to obtain use of the disk.

5.9 GLOBAL DESIGN CONSIDERATIONS

The following paragraphs describe some of the methods for creating and designing globals. The programmer should refer to the *Introduction to MUMPS Language* tutorial manual and Appendix G in this manual for detailed discussions on how MUMPS stores data in globals.

5.9.1 String Data Storage

As mentioned in paragraph 5.8, simultaneously running jobs compete with each other for disk access time. MUMPS makes at least one disk access each time a job references a global variable using full syntax. A particular job using a large number of global variables for its processing operations, therefore, takes a long time to run when there are many other jobs on the system.

If a particular application program processes a large amount of string data, the programmer may reduce the number of necessary disk accesses by storing as much string data as possible in a global variable. The programmer may then store the contents of the global variable in a local variable to extract and process the relevant data.

A global variable may contain many individual string data fields as long as the total length of all the fields does not exceed the maximum string length of 132 characters.

The programmer should store the fixed length string data fields in the first part of a global variable. This allows him to use the $EXTRACT function to retrieve any particular field. The programmer may then store variable length fields, set off by delimiting characters, after the fixed length fields in the global. The programmer may use the $PIECE function to extract the individual variable length fields.

As an example, suppose that a patient's hospital record contains a billing code, ward name, room number, doctor code, blood type, Rh factor, name, and activity. Each of these fields could be stored as separate global variables in the same global, or even as separate globals. The following command, however, stores all information in one global variable.

```
                  billing code  blood type               delimiters
S ↑REC(1700,97)="0458OBS13228AN/NARTEN,SUSAN/BILIRUBIN/"
                     ward                  name        activity
                        room
                          doctor
                           code
                fixed length      variable length
                fields            fields
```

If the user wished to find the names of all the patients in the obstetrics ward (OBS), he could use some form of the following command:

```
IF $E(X,5,7)="OBS" T !,$P(X,"/",2)
```

where "X" is a local variable containing one of the records in the global variable ↑REC.

5.9.2 Downward Pointers

The disk block address stored with a global variable that points to global variables on lower levels is called a downward pointer. MUMPS chooses downward pointers by using an algorithm that takes advantage of the rotational latency of the disk. In the time it takes to perform a disk READ operation, the disk turns to the block on the lower level. The position of the global variable containing the downward pointer is therefore critical. If the global variable containing the downward pointer is moved onto a continuation block, the optimal rotational latency value is lost. This increases the amount of time it takes to search down the levels of a global.

In order to maintain the proper rotational latency between the blocks, the user should ensure that the global variable containing the downward pointer is never moved onto a continuation block by:

- storing data in the global variable, if any, before creating the first global variable at a lower level
- storing only fixed length data in the global variable containing a downward pointer

NOTE

Occasionally it is necessary to make major modifications of a global's structure or contents. The user can restore the optimal block allocation by saving the global data using the Global Save Utility Program, killing the global, and then restoring the global using the Global Restore Utility Program.

5.9.3 Storing Large Amounts of Data

The two primary considerations involved when storing large amounts of data are access speed requirements and storage limitations. The following paragraphs discuss the ways to balance these conditions.

The fewer the number of physical disk block accesses required to retrieve data, the faster the retrieval time. The fastest way to retrieve data is by searching down the levels of a global. If there are no continuation blocks on any one level of a global, MUMPS accesses only one block for each downward pointer. For example, a three-level global that uses no continuation blocks requires only three block accesses to retrieve any particular global variable, excluding the directory block access.

Each level of a global adds at least one block to the number of blocks required to store the data. If the amount of storage space is limited, the user may wish to store data on as few levels as possible, though he increases retrieval time.

The following example illustrates the method for calculating the number of block accesses and the number of blocks required to store global data. Using this method, the programmer may design a global structure appropriate for his application situation.

Example:

A user has 1000 records containing string data. Each record is 45 bytes long. If one record is stored per global variable, there will be 1000 global variables containing data, and each variable will occupy 50 bytes of storage.[1]

A disk block is 512 bytes long. MUMPS uses six of those bytes to store system information. There remain 506 bytes per block in which to store global variables. Since it is possible to store 10 global variables in one block, it will take 100 blocks to store all the data (i.e., 506 bytes per block divided by 50 bytes per variable).

The user may choose to store all the data on one level as a series of continuation blocks, requiring only the 100 disk blocks. The average number of blocks accessed to retrieve any particular global variable is 100 blocks divided by 2, or 50 blocks (excluding the directory block access).

The user may also choose to create a 2-level global where the first level contains 2 pointers, each pointing to a group of 500 global variables. Each group of 500 nodes resides on 50 continuation blocks. For example, the first level might contain two pointer nodes ↑A(1) and ↑A(2). The second level would then contain 1000 global nodes, ↑A(1,n) and ↑A(2,n) where n = 1 - 500. The two pointer nodes would reside in one block; the total number of blocks required to store the global is 101. The average number of block accesses required to retrieve any particular global variable is 50 blocks per pointer divided by 2, plus 1 block for the pointer, or an average of 26 accesses (excluding the directory block access).

If the user creates a 3-level global, he might have 4 pointers on the first level, each pointing to a group of 25 pointers on the second level, each of which, in turn, points to a single block containing 10 global variables storing data on the third level. This global requires 100 blocks to store the data, plus 5 block to store the pointers -- one block for the first level

[1]The programmer should refer to Appendix G in this manual to determine the number of bytes required to store any particular data type.

pointers, and 4 blocks for the second level pointer--or a total of 105 blocks for the entire global. The maximum number of blocks accessed to retrieve any global variable is only three (once again excluding the directory level access). It is evident that increasing the number of blocks required to store a global can significantly decrease the retrieval time.

The user may also try another possible arrangement. In the 2-level structure described above, the "data" nodes reside in two groups of 50 continuation blocks each. The large number of continuation blocks account for the major portion of the retrieval time. If the user decreases the number of continuation blocks, he can decrease the average retrieval time.

A global variable which contains a downward pointer and no other data requires only 6 bytes of storage space. It is therefore possible to put as many as 84 "pointer" nodes in one disk block. If the user creates a 2-level global with 100 pointers on the first level which each point to a block on the second level containing the 10 global variables, he needs only 2 blocks to store the 100 pointers. The entire global requires only 102 blocks of storage space. The number of blocks accessed to retrieve any global variable on the second level is either 2 or 3, depending on whether MUMPS has to read the continuation block on the second level to find the proper pointer. Most of the time only 2 blocks will be accessed because most of the pointers are in the first block. This compromise is the most effective solution for this particular data base.

5.10 USING SWITCH REGISTER SWITCHES FOR PROGRAM CONTROL

In addition to being used to initialize and control the operation of the MUMPS system software[1], the PDP-11 console SWITCH REGISTER switches can also be used to effect control of any MUMPS program. Many of the system utility programs described in the MUMPS Operator's Guide use this feature.

The console SWITCH REGISTER consists of sixteen switches (Ø thru 15). Several of the switches perform specified predefined functions in the operating system (Table 5-3). In general, these switches, except for switch number 6, should not be used since their primary function could conflict with the user's intended application. The programmer should instead use switch number 6 to effect a shut-down of the application system software. The switches that have no specific MUMPS system assignment can be used, but are subject to future assignment by DIGITAL. The use of a switch to cause program shutdown is particularly important for STARTed programs that are not controlled through a device keyboard or other logical input. Unless switch control is used, programs of this sort can only be stopped by actually HALTing the MUMPS operating system.

Since the SWITCH REGISTER switches are a form of input device, they communicate with the processor through a status word in the PDP-11's 'external page' at address 77757Ø (octal) or 654ØØ (decimal). Each bit in the status word corresponds to one of the switches. For example: bit Ø corresponds to switch Ø, bit 1 corresponds to switch 1, etc. When a switch is ON (raised) its status word bit is set to one.

Using the $V function, MUMPS programmers can obtain the current status of the switches. The appropriate bit or bits can be examined in a Boolean expression that uses the bit masking techniques described in 5.1. The last column in Table 5 shows the masks for each bit. The following examples below show how the switch status can be obtained.

---

[1]Described in Chapter 4 of the *MUMPS-11 Operator's Guide*.

Table 5-3

MUMPS SWITCH REGISTER Assignments and Bit Masks

Switch Number	Bit Mask	MUMPS System Function
Ø	.Ø1	Console Terminal Control
1	.Ø2	
2	.Ø4	Unused
3	.Ø8	
4	.16	Garbage Collector Control
5	.32	Unused
6	.64	System Shutdown Control
7	1.28	
8	2.56	
9	5.12	Unused
1Ø	1Ø.24	
11	2Ø.48	
12	4Ø.96	Partition Grant Control
13	81.92	System Processing Control
14	163.84	Unused
15	327.68	Interpreter Control

Examples:

1. To see if switch number 6 is ON:

   ```
   6.23 I '($V(654ØØ)/1ØØ&.64=.64) T "RAISE SWITCH NO. 6",! G $L
   ```

2. To HALT if both switch number 6 and switch number 1 are ON:

   ```
   3.45 I $V(654ØØ)/1ØØ&.66=.66 H
   ```

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: a. enter commands and/or functions for immediate execution b. 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
ALT MODE	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␣0 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 may 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 tape mark)†	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 (Read label)†	034	028	FS
010	008	BS* (Write header label)†	035	029	GS
011	009	HT (Write EOF label)†	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	^ {↑}
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 −0.38 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: a. Command is undefined in the language; b. 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 (C.4), this user software error indicates an attempt to: a. use VIEW command to access a block number larger than size of the referenced disk, or a nonexistent disk; or b. 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. a. exponent overflow b. exponent underflow c. division by 0 d. 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
MODER	-0.23	a. An nve was encountered where an svl was expected or vice versa. b. Argument to $TEXT is not numeric. c. 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).

Message	$E Index	Meaning
NAKED	-0.29	Indicates that the present user attempted to reference a global variable using "naked" syntax: a. prior to any full syntax reference; or b. 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: a. too many program steps in the program being created via the terminal or in the program being loaded; (LOAD, CALL and OVERLAY commands) b. too many local variables; c. expression or subscript nesting too deep.
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.

Message	$E Index	Meaning
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.

Message	$E Index	Meaning
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 C.1, this may indicate that disk block pointers in the global data base reference nonexistent or invalid disk blocks. *Notify system manager.*
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: a. not set to ON LINE; b. not set to WRITE ENABLE; c. unit number not selected.
LPERR	-0.38	Indicates a line printer hardware error. Common causes are: a. device off line b. out of paper c. yoke open d. 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.
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 a. that the previous swap-out overflowed the user partition stack. The error is not reported until the next swap-in. b. 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
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: a. 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); b. as an expression operator, it is followed by a Pattern Specification Code (psc). c. 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: a. A command followed immediately by two spaces indicates the command has no arguments; b. One space separates a command from its arguments, or the last argument of a preceding command from the next command on the line.

Symbol	Definition
:	Colon is multiply defined: a. a delimiter for field separation in the argument of FOR, MODIFY, and ASSIGN commands. b. used to indicate the presence of an optional expression appended to a command or the argument of a command (where allowed). c. 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. a. precedes the first character of a System Variable. b. 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 used 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

Addition | Multiplication

Binary Scale

$$0 + 0 = 0 \qquad 0 + 1 = 1 + 0 = 1 \qquad 1 + 1 = 10$$

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

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 (Decima

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 (Decim

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

APPENDIX F
REFERENCE DATA FOR SYSTEM TABLES

F.1 INTRODUCTION

This Appendix provides reference data on the various information tables which reside in the MUMPS-11 Operating System. These tables are physically contiguous and reside in low memory below the system's time-sharing Executive (Figure F-1).

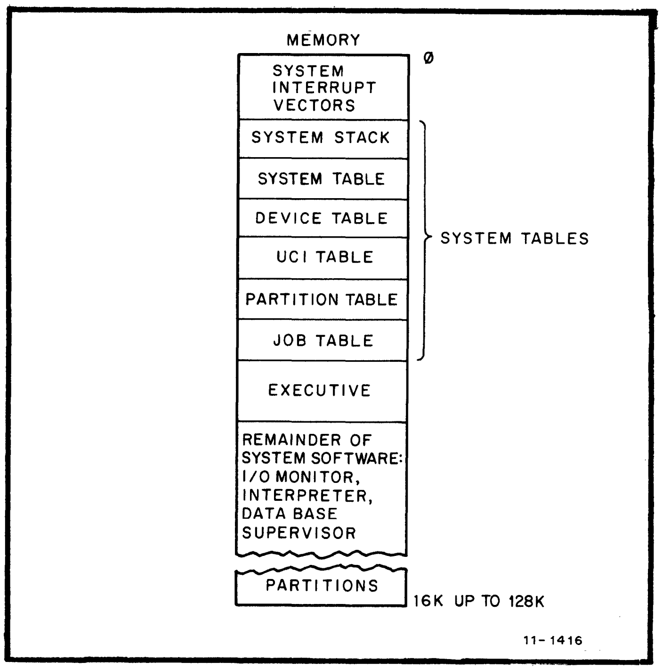


Figure F-1 Relative Table Position

These tables are maintained by various routines in the Operating System and contain system control and status information including: general system configuration data, time and date values, base addresses for other tables, I/O device and partition utilization, User Directory areas, and job status. All table information is essential for system operation. Also, some of this information may be especially useful to the MUMPS System Manager/Programmer, for developing and troubleshooting MUMPS application system programs and the system itself.

Using the special system features of the MUMPS Language which include the VIEW Command and the $VIEW function, the System Manager/Operator can access and alter table data as required. Modification of internal memory (or disk) by careless or inexperienced individuals can have disastrous results on system operation.

The following paragraphs describe the System Table, the Device Table, the UCI Table, the Partition Table, and the Job Table.

The System Table specifies a number of system constants and parameters. The Device Table specifies current I/O device information including the device being used ('owned') and devices not physically present. The UCI Table contains all legal UCI codes (input during system generation or modification) and the addresses of associated Global and Program Directories. The Partition Table specifies the size and location of each partition in the system. The Job Table specifies job status.

A word of explanation about the term "job" is in order at this point. A job as defined by MUMPS-11 is any user activity which requires the use of a partition. Thus, logging-in to the system initiates a job. A program started in another partition via the START command is also deemed a job.

F.2 THE SYSTEM TABLE (SYSTAB)

The System Table (Table F-1) is the repository of basic system constants, indicators, control information, and special buffer addresses. In addition, it contains address pointers to all other system tables. This is important to note, since the specific addresses of these tables and the System Table itself may change from time to time in succeeding versions of the Operating System. Thus access to system tables should always be made initially through location 44 (54 octal). This location will always contain the base address of the System Table. The relative position of System Table entries is also fixed so that all tables can be located via location 44 and the System Table.

Example:

To obtain the address of the System Table, type:

>SET A=$V(44)

To obtain the address of the first entry in the Device Table, type:

>SET B=$V(A+4)

or to combine the operations:

>SET A=$V($V(44)+4)

Table F-1

System Table

Location (decimal)	Contents
SYSTAB+Ø	Address of Job Table (JOBTAB)
+2	Address of Partition Table (PARTAB)
+4	Address of Device Table (DEVTAB)
+6	Base Address of Device Descriptor Buffers (DDB)
+8	Address of UCI Table (UCITAB)
+1Ø	Address of Disk Storage Allocation Table
+12	Address of Garbage Table +2 (disk blocks to be deallocated)
+14	Address of first 256-word buffer in Buffer Pool
+16	Count of number of illegal interrupts the system has detected
+18	Disk block address of data contained in UTLBUF (SYSTAB+1ØØ)
+2Ø	Logical disk number for block in UTLBUF
+22	Address of Disk Buffer #1
+24	Address of Disk Buffer #2
+26	Garbage Indicator (Ø = no garbage)
+27	Physical Disk Overflow Switch for Logical Disk Ø (Ø = overflow allowed)
+28	Number of ticks remaining in this second
+29	Number of ticks per second
+30	Number of ticks left in current job's time slice
+31	Initial value of slice (ticks) for current job
+32	Number of ticks in basic time slice
+33	Time in seconds since midnight (high-order bits)
+34	Time in seconds since midnight (low-order bits)
+36	Date in the form: (yy*500)+ddd where: yy = year -1900 ddd = day count since December 31
+38	Base address of System Stack
+4Ø	First available address above device buffers
+42	Base address of first partition (other partitions follow sequentially)
+44	Status Register address of system clock
+46	Number of job in the run queue on system restart
+47	Number of job in the disk I/O bound queue on system restart

(continued on next page)

Table F-1 (Cont.)

System Table

Location (decimal)	Contents
SYSTAB+48	Line Buffer starting address
+50	End Address of Symbol Table+1
+52	Pointer to Global Directory (3 bytes)
+55	Physical Disk Overflow Switch (0 = overflow allowed)
+56	Buffer Address given to most recent Job in Run Queue
+58	Address of System Information Block on system disk (3 bytes)
+61	Most recent Job in Run Queue (when 0, job is being Swapped out; error not reported until next swap-in)
+62	Programmer Access Code, initially set
+64	to CTRL/X CTRL/X CTRL/X
+66	Disk error tally counter, incremented by 1 for each disk I/O hardware error
+68	Magtape unit 0 buffer address from pool
+70	Magtape unit 1 buffer address from pool
+72	Magtape unit 2 buffer address from pool
+74	Magtape unit 3 buffer address from pool
+76	Reserved Device #0
+78	Reserved Device #1
+80	Reserved Device #2
+82	Reserved Device #3
+84	DECtape unit 0 buffer address from pool
+86	DECtape unit 1 buffer address from pool
+88	DECtape unit 2 buffer address from pool
+90	DECtape unit 3 buffer address from pool
+92	Sequential Disk Processor #0 buffer address from pool
+94	Sequential Disk Processor #1 buffer address from pool
+96	Sequential Disk Processor #2 buffer address from pool
+98	Sequential Disk Processor #3 buffer address from pool
+100	UTLBUF Address of utility buffer (view device) from pool
+102	Magtape Error Count
+104	DECtape Error Count
+106	RK11 Disk Error Count
+108	Disk Address of latest error (low order word)
+112	RK11 Hardware Status Register
+114	RF11 Disk Error Count
+116	Disk Address of Latest error
+118	Disk Address of latest error (high order word)
+120	RF11 Hardware Status Register
+122	RP11, RP04 Disk Error Count
+124	Disk Error Address: section in bits 0-3, track in bits 8-12
+126	Disk Error Address: cycle in bits 0-8, drive in bits 10-12
+128	RP11, RP04 Hardware Status Register
+130	Base Address of Ring Buffer area
+132	Base Address of Ring Buffer Queue (address of next available Ring Buffer)

(continued on next page)

Table F-1 (Cont.)

System Table

Location (decimal)	Contents
SYSTAB+134	Base Address of All Multiplexer DDB's
+136	End Address of All Multiplexer DDB's
+138	RP11 Disk Drive Number Remapping Table (each byte corresponds to a physical unit (0-7); initially assigned to successive logical units 0-7 in that order)
+146	RK11 Disk Drive Number Remapping Table (each byte corresponds to a physical unit (0-7); initially assigned to successive logical units 0-7 in that order)
+154	Number of Jobs waiting to run
+155	Write-check Switch (if non-zero, all disk write operations will be checked)
+156	Index into System Bootstrap for Disk Descriptor Table
+158	Low limit address for System Stack (LOWSTK); normally set to 450_8, but may be altered during system generation.
+160	Address of First Global Buffer Descriptor
+162	Address of Interrupt Service Routine for Multiplexer
+164	Multiplexer #1 Hardware Status Register
+166	Multiplexer #2 Hardware Status Register
+168	Start Address of Global Disk Buffers
+170	Count of logical Disk Reads
+172	Count of Buffer Swaps
+174	Count of Physical Disk Reads
+176	Number of job being killed by RSJ
+178	Number of Bus errors
+180	TM11 Status Register
+182	TM11 Command Register
+184	TM11 Byte Record Counter Register
+186	TM11 Current Memory Address Register
+188	TM11 Data Buffer
+190	TM11 Read Lines
+192	DECtape Control and Status Register
+194	DECtape Command Register
+196	DECtape Word Count Register
+198	DECtape Bus Address Register
+200	DECtape Data Register
+202	RP11 Interrupt Service Address
+204	RP04 Interrupt Service Address
+206	RP11 Control Status Register
+208	RP04 Control Status Register
+210	RP04 Disk Drive Number Remapping Table (each byte corresponds to a physical unit, 0→7, initially assigned to successive logical units 0→7 in that order)
+218	RP04 Storage Allocation Table Base Address
+220	TJU16 Control Status 1 Register
+222	TJU16 Control Status 2 Register
+224	TJU16 Drive Status Register
+226	TJU16 Error Register
+228	TJU16 Tape Control Register
+230	Magtape Unit 0 Device Descriptor Buffer Base Address

Table F-1 (Cont.)

System Table

Location (decimal)	Contents
SYSTAB+232	11/70 Group of Cache Memory Error Count (1 byte)
+233	11/70 Group I Cache Memory Error Count (1 byte)
+234	11/70 Memory Parity Error Low Address Register
+236	11/70 Memory Parity Error High Address Register
+238	11/70 Memory Parity Error System Register
+240	11/70 Memory Parity Error - Job Number of Last Hung Job
+242	Multiplexer #3 Hardware Status Register
+244	Logical Address of Partition Base Address if >28K System (24576 or 20480)
+246	Address of EBLMEM Subroutine (used by Bootstrap Loader)
+248	Address of REQJOB Subroutine (used by Bootstrap Loader)
+250	Address of SETJOB Subroutine (used by Bootstrap Loader)
+252	Address of MUMPSØ Subroutine (used by Bootstrap Loader)
+254	RP02-RP03 Table (each byte corresponds to a unit, non-zero implies an RP02, zero implies an RP03) 8 bytes
+262	Status Register Address for DZ-11 #1
+264	Status Register Address for DZ-11 #2
+266	Status Register Address for DZ-11 #3
+268	Status Register Address for DZ-11 #4
+270	Status Register Address for DZ-11 #5
+272	Status Register Address for DZ-11 #6
+274	Beginning Address of DZ-11 DDB's
+276	Pointer to Card Reader Table if One in System. Card Reader Table consists of 3 words, as follows: .Word 0 ; SYSGEN Enters CR-11 CSR Address .Word 0 ; SYSGEN Enters CR-11 Vector Address .Word 0 ; SYSGEN Enters Device Number of Card Reader (must be 51.-54.)
+278	CSR for DM11-BB Attached to DH11 #1
+280	CSR for DM11-BB Attached to DH11 #2
+282	CSR for DM11-BB Attached to DH11 #3
+284	Interrupt Entry Code for DM11-BB #1 This Location +14. for DM11-BB #2 This Location +28. for DM11-BB #3
+286	Communication Device Flag (0=CPU,1=DMC-11)
+288	Address of DMC-11 Interrupt Service Routine
+290	Pointer to Last Byte of RJ Maps in DSKSAT
+292	Pointer to Last Byte of RK06 Map Area in DSKSAT
+294	RK06 Drive Remapping Table (each byte corresponds to a physical unit (0-7); initially assigned to successive logical units 0-7 in that order)
+302	Address of RK06 Interrupt Service Routine
+304	Address of Control Status Register for RK06
+306	Count of RK611 Errors
+308	RK06 Error Address, Sector # and Track #
+310	RK06 Error Address, Cylinder # and Drive #
+312	RK06 Error Status Register
+314	RH11/RP04,05,06 Error Status Register (RPER2)
+316	RH11/RP04,05,06 Error Status Register (RPER3)

F.3 DEVICE TABLE (DEVTAB)

The Device Table (Figure F-2) is a table of byte entries, one for each possible I/O device in the system. The first entry in the table (DEVTAB+0) specifies the length of the table; all other entries are associated with the various I/O devices in the system. If a device is available, its table entry is zero. If a device is being used or is 'owned' by a job, the device entry contains the Job Entry Number. The ASSIGN Command is used to place job entry numbers in the table. When a device does not exist in the system, its table entry is set to -1 (equivalent to 377_8).

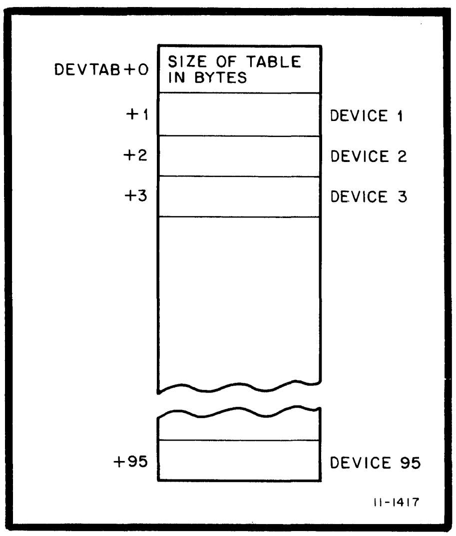


Figure F-2 Device Table

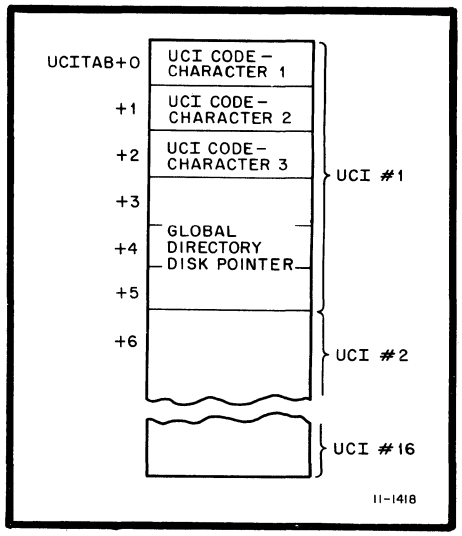


Figure F-3 UCI Table

F.4 USER CLASS IDENTIFICATION TABLE (UCITAB)

The UCI Table (Figure F-3) specifies all legal UCI's acceptable to the system and the location of the Global and Program Directories associated with each. Each entry in the table is three words long. Up to 16 entries can be made; one for each UCI. These entries are initially set by the MUMPS System Generation Program (SYSGEN) and can be modified by the Modify System Parameter Program (MSP). The first entry in the table is defined as the System UCI or UCI #1.

The first three bytes of an entry contain the ASCII representation of the UCI code. The fourth, fifth and sixth bytes contain information which points to a Global Directory. By convention, the Program Directory is located in the next contiguous disk block after the Global Directory.

F.5 PARTITION TABLE (PARTAB)

The Partition Table (Figure F-4) specifies the size and base address of each partition defined in the system. The entries in this table are set during System Generation[1]. The table consists of single-word entries. The first entry is a header word, the low (even) byte of which specifies the standard partition size in 128-word multiples; the high byte specifies the maximum number of partitions that can be described by the table. At System Generation, the user may specify a maximum of 18 partitions for systems with no more than 28K words of memory, or a maximum of 40 partitions with more than 28K words of memory. Each of the remaining entries in the table specifies partition information.

The low byte of a partition entry word specifies the size of the partition as the number of 128-word increments minus one. For example, a 4K-word partition would have an entry of 31_{10}. (4,096/128=32; 32-1= 31). The high byte of the partition entry word contains the high-order bits of the partition's base address (location of the first word).

In systems with no more than 28K words of memory, the knowledgeable system manager or operator may examine partitions other than the one currently owned. To do so, he calculates a partition's base address as described below, and then uses the address with the $VIEW function to examine locations in the partition.

A partition's base address is an integer multiple of 256. Therefore, the significant digits of the base address (as stored in a binary 16-bit word), occupy only the high-order byte of the word. To obtain the integer address to use with $VIEW, the system programmer must zero (mask) the low-order byte of the partition table entry word.

Example:

To obtain the address of Partition 2:

$$\texttt{>SET A=}\underbrace{\texttt{\$V(}\underbrace{\texttt{\$V(}\underbrace{\texttt{\$V(44)}}_{1}\texttt{+2)}}_{2}\texttt{+4)}}_{3}\underbrace{\texttt{/25600}}_{4}\underbrace{\texttt{*25600}}_{5}$$

[1]Described in the *MUMPS-11 Operator's Guide*.

1. Get the base address of the System Table.

2. Get the base address of the Partition Table (SYSTAB+2).

3. Get the entry word for partition #2 (PARTAB+4).

4. Divide that by 25600 to shift out the low byte (shifts one byte to the right). This works because MUMPS truncates fractional parts to two decimal places and $V returns only integers.

5. Multiply by 25600 to shift one byte to the left to get the base address of the partition as an integer for later use with $V.

The memory management characteristics of systems with more than 28K words of memory do not allow examination of partitions other than the one currently owned.

Each Partition Table entry corresponds with entries in the Job Table (JOBTAB). The job associated with the n^{th} entry in the Job Table always uses the partition specifies by the n^{th} entry in the Partition Table. The value n is always even and in the range: (1 through maximum number of partitions)*2. That is, partition 2 is the second entry in both tables and its address is 'Table base address +4'. In this way, the value of n can be used for indexed accesses both to PARTAB and JOBTAB.

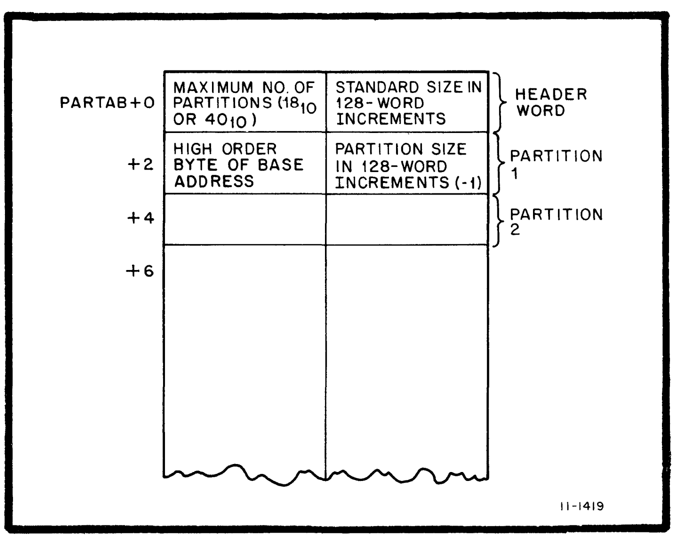


Figure F-4 Partition Table

F.6 THE JOB TABLE (JOBTAB)

The Job Table (Figure F-5) is where all system queue information is kept. All system queues are contiguous with the Job Table and all entry numbers (job numbers) are relative to the base of the Job Table.

The low (even) bytes of Job Table entries are used for system queue space and the high bytes for job hung status. If bit 15 of a Job Table entry is set, then the job is in a hung state and by definition *not* in a system queue.

The first word in the Job Table is a header word containing the following information:

> (Even) low byte = number of currently available partitions
> (Odd) high byte = number of partitions in the system.

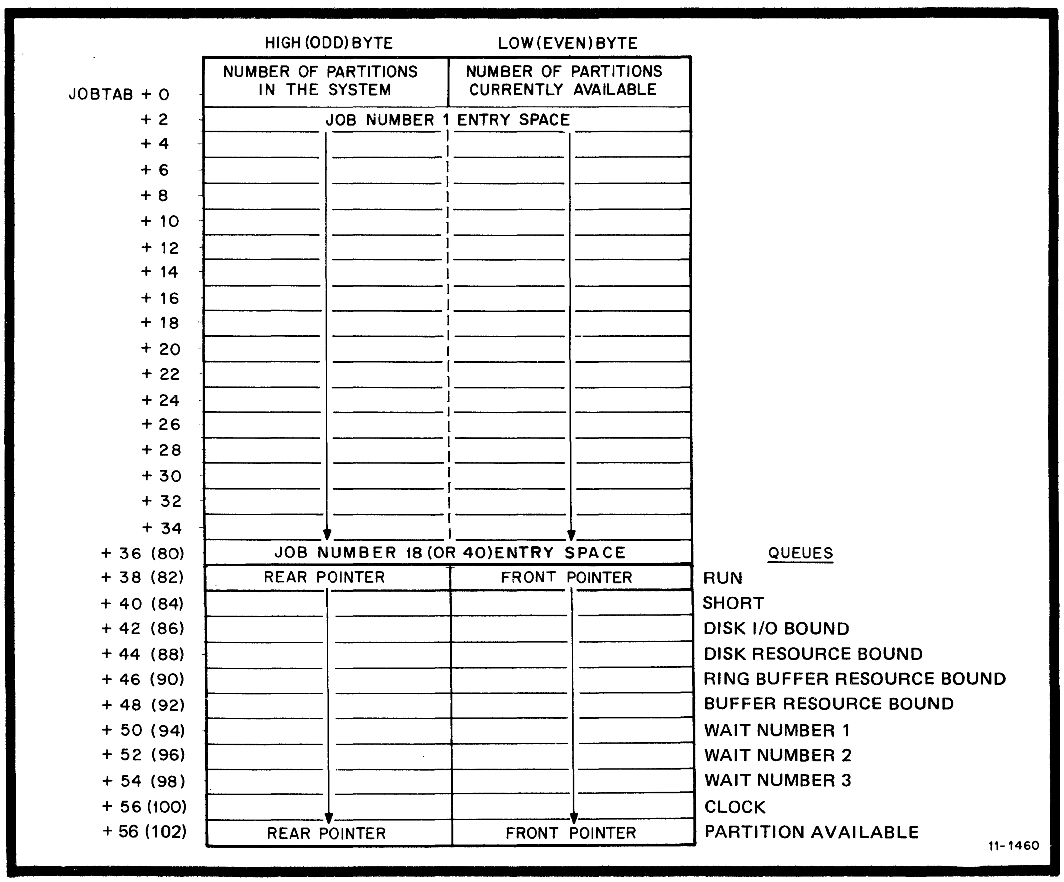


Figure F-5 Job Table (JOBTAB)

F.7 DEVICE DESCRIPTOR BUFFER (DDB)

Each terminal device that exists in the system (device number 1, 4-19, and 64-111) has an associated 16-word device descriptor buffer (DDB). DDBs also exist for devices 2 and 3, regardless of whether or not they are physically present in the configuration. Their format differs, but they can be found using the same formula. The DDB is used to contain both static and dynamic terminal device information. The base address of a terminal's DDB for devices 1, 4-19 is found as follows:

DDB Address = ((TRM-1)*32) + BASE

where: TRM = Terminal's device number

BASE = Base address of DDB #1 (contained in SYSTAB+6 of the system table).

Table F-2

Device Descriptor Buffer for Devices 1, 4-19

WORD (decimal)	Location (octal)	Contents
1-2	DDB+0	Contains a "JSR,R1,ROUTINE" instruction sequence. Where: ROUTINE is the address of the entry point into the device's interrupt service routine.
3	+4	Address of the device's first receiver register.
4	+6	Bit setup for receiver status register.
5	+10	Bit setup for transmitter status register.
6	+12	Terminal error status ($A data).
7	+14	Low byte: Horizontal position in the line ($X). High byte: Vertical position on the page ($Y).
8	+16	Low byte: Device status 0 = No activity >0 = Input <0 = Output 200 = CTRL/O typed - suppress output. High byte: Device Information Bit Meaning 0 0 = Teletype-like terminal 1 = Video Terminal 1 0 = Echo Character 1 = No echo, no BREAKs 2 0 = Device can input or output 1 = Device can only output 3 1 = LP11 line printer 4 1 = Modem (DL11E, DH11, or DZ11)

continued on next page

Table F-2 (Cont.)

Device Descriptor Buffer for Devices 1, 4-19

WORD (decimal)	Location (octal)	Contents
8	DDB+16	Bit Meaning 5 1 = CPU-CPU Device 6 1 = Device on a DZ11 MUX 7 0 = No parity 1 = Compute even parity (For a VT52, bits 0 and 3 are set.)
9	+20	Low byte: character currently being echoed. High byte: Data set (if present) status indicator.
10	+22	Low byte: Number of characters +1 to stall. High byte: If CPU-CPU: CRC If not CPU-CPU: Bits 0-7, character to output when stalling. Bit 8, "X␣OFF" in effect if =1.
11	+24	Pointer to last character input from ring buffer.
12	+26	Pointer to last character output to ring buffer.
13	+30	Low byte: Partition size needed (0 = standard size) High byte: Index into UCI table "Tied" Terminal Information
14	+32	Program name (3 bytes)
15	+34	High byte: Right page margin as specified in ASSIGN command.
16	+36	Low byte: $B - Current message counter for CPU-CPU Handler. High byte: $H - Temporary variable for CPU-CPU Device, VT52 ESC code character.

The base address of a terminal's DDB for devices 64-111 is found as follows:

DDB Address = ((TRM-64)*32) + base

Where: TRM = Terminal's device number

BASE = Base address of DDB's for multiplexers (contained in SYSTAB+134 of the system table).

Table F-3

Device Descriptor Buffer for Devices 64-111 (DH11)

WORDS (decimal)	Location (octal)	Contents
1-2	DDB+0	Unused.
3	+4	Line parameter register. Bits 0,1 - Character Length 00 5 bits 10 7 bits 01 6 bits 11 8 bits (default-All TTY's, LA30, VT05) Bit 2 - # Stop bit 0 - 1 Stop bit 1-2 for 6-8 bit characters, 1.5 for 5 bit characters (default). Bit 4 - Parity 0 - Disable (default) 1 - Enable Bit 5 - Type parity (only if bit 4=1; ignored if bit 4=0). 0 - Even 1 - Odd Bits 6,7,8,9 - Receiver speed Bits 10,11,12,13 - Transmitter speed 0 - Zero baud 10 - 600 baud 1 - 50 11 - 1200 2 - 75 12 - 1800 3 - 110 13 - 2400 4 - 184.5 14 - 4800 5 - 150 15 - 9600 6 - 200 16 - External input A* 7 - 300 17 - External input B* *Special order hardware.
4	+6	Not used.
5	+10	Low byte: Use =-1 Stuffing buffer for output =+1 Read hung = 0 Some other state, do not wake on interrupt High byte: MARSW - Set non-zero When outputting CR/LF at right margin.
6	+12	TRMSA - Error Status of terminal.
7	+14	Low byte: $X - Indicates horizontal position in the line. High byte: $Y - Indicates vertical position on the page.

continued on next page

Table F-3 (Cont.)

Device Descriptor Buffer for Devices 64-111 (DH11)

WORDS (decimal)	Location (octal)	Contents
8	DDB+16	Low byte: Status-0 = No activity >0 = Input <0 = Output 200 = ^0 on to suppress output High byte: Cond 10: Information particular to this device. Bit 0: 0 = TTY 1 = Video terminal Bit 1: 0 = Echo char 1 = No echo and no breaks Bit 2: 0 = Input/output device Bit 3: 1 = LP11 Bit 4: 1 = Modem (DL11E, DH11, or DZ11) Bit 5: 1 = CPU/CPU Bit 6: 1 = Device on DZ11 multiplexer Bit 7: 1 = Compute even parity. 0 = No parity check For VT52, bits 0 and 3 are set.
9	+20	Low byte: Echo - Current char being echoed. High byte: DATAST - Dataset status.
10	+22	Low byte: Stall - Number of characters (+1) to stall on FF. High byte: (Bits 0-7) STCHAR - The character to output when stalling. Bit 8 "XOFF" in effect if = 1
11	+24	RINGIN - Pointer to last char inputted from R.B.
12	+26	RINGOUT - Pointer to last char taken from R.B.
13	+30	Low byte: Size - Partition size needed; 0 = Standard High byte: UCIn - Number of UCI
14	+32	Pname - Program name (3 bytes)
15	+34	Low byte: Pname High byte: Margin - Right margin, as specified in 'ASSIGN' CMMD.
16	+36	High byte: VT52 ESC code
		The DDB is found by multiplying the device number ($I(JOBASE)) by 32 (bytes), the length of the buffer, and adding that to the base DDB address (MLXBEG). The routine "SETUP" in 'UTIL' does the actual calculation.

Table F-4

Device Descriptor Buffer for Devices 64-111 (DZ11)

WORD (decimal)	Location (octal)	Contents
1-2	DDB+0	Contains a "JSR,R1,ROUTINE" instruction sequence. Where: ROUTINE is the address of the entry point into the device's interrupt service routine.
3	+4	Line parameter register. Bits 0,1 - Character Length 00 5 bits 10 7 bits 01 6 bits 11 8 bits (default-All TTY's, LA30, VT05) Bit 2 - # Stop bit 0 - 1 Stop bit 1-2 for 6-8 bit characters, 1.5 for 5 bit characters (default). Bit 4 - Parity 0 - Disable (default) 1 - Enable Bit 5 - Type parity (only if bit 4=1; ignored if bit 4=0). 0 - Even 1 - Odd Bits 6,7,8,9 - Speed 0 - baud 10 - 1200 baud 1 - 50 11 - 1800 2 - 75 12 - 2000 3 - 110 13 - 2400 4 - 134.5 14 - 3600 5 - 150 15 - 4800 6 - 300 16 - 7200 7 - 600 17 - 9600
4	+6	Internal constant
5	+10	DZ11 CSR Address
6	+12	Terminal error status ($A data).
7	+14	Low byte: Horizontal position in the line ($X). High byte: Vertical position on the page ($Y).

8	+16	Low byte: Device status
		0 = No activity
		>0 = Input
		<0 = Output
		200 = CTRL/O typed - suppress output.
		High byte: Device Information
		
		Bit Meaning
		0 0 = Teletype-like terminal
		    1 = Video Terminal
		1 0 = Echo Character
		    1 = No echo, no BREAKs
		2 0 = Device can input or output
		    1 = Device can only output
		3 1 = LP11 line printer
		4 1 = Modem (DL11E, DH11, or DZ11)

APPENDIX G
SYSTEM DATA STRUCTURES

G.1 INTRODUCTION

This appendix provides reference information on the various data structures used internally by the MUMPS Operating System to contain user programs and data both in core and on the disk data base.

G.2 DISK DATA STRUCTURES

All data stored on the disk is in 256-word (512-byte) blocks. The first word and last two words in each block always contain the same type of data, even though the rest of the contents of the block may differ widely. The first word is a relative pointer to the first available byte in the block. If the block contains data, the pointer is always 2 or greater, the 2 being necessary to index around the first word itself. The last two words contain the continuation pointer to the next block in the chain. A pointer value of zero (Ø) indicates the end of a chain.

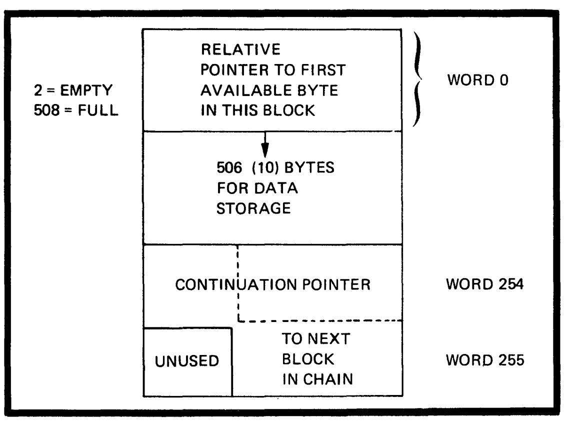
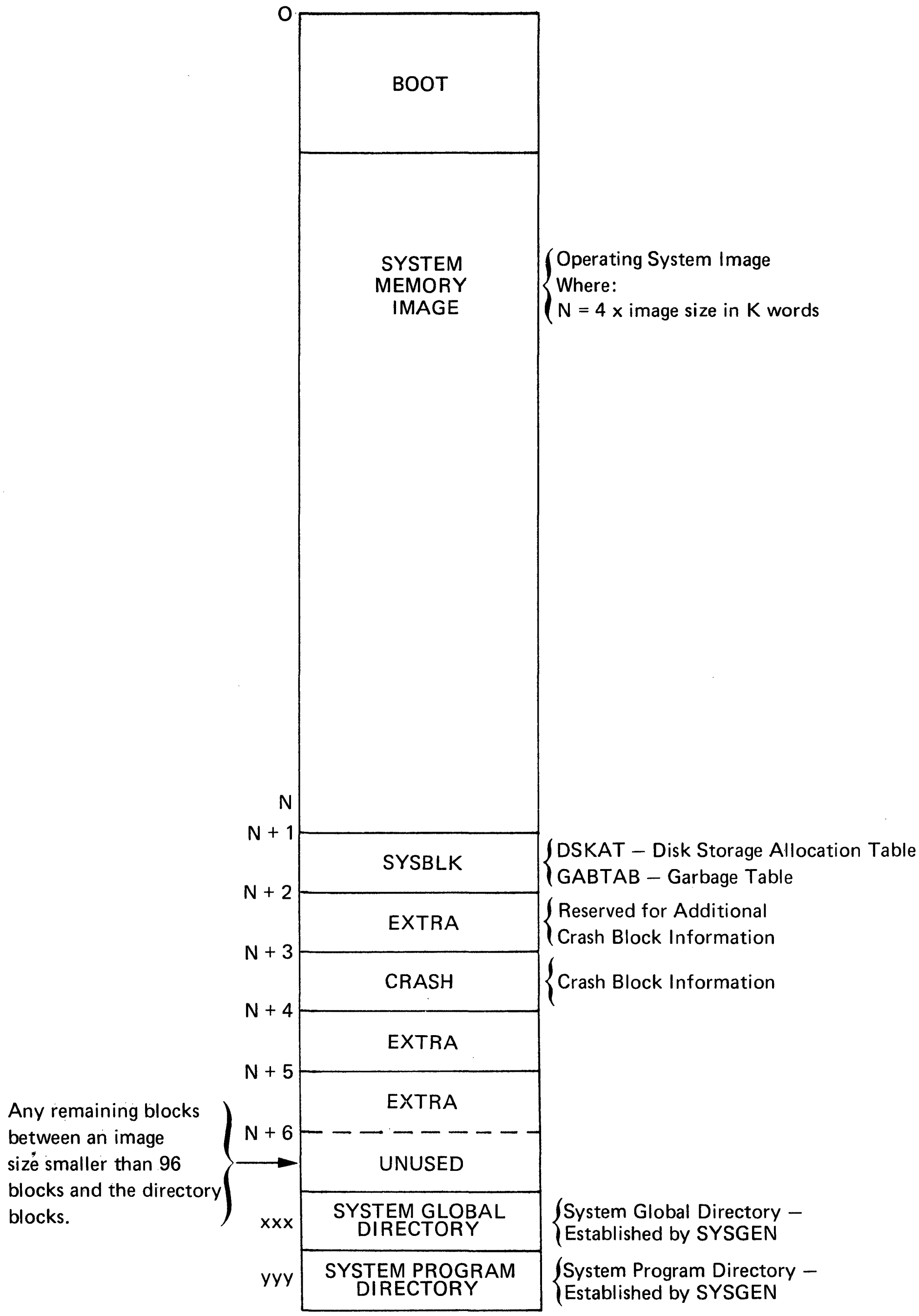


Figure G-1 Basic Disk Data Block

The five kinds of data blocks generally kept on the system disk are global data blocks, bit map blocks, program data blocks, Global Directory Blocks and Program Directory Blocks. These five types of data blocks have a common format, as shown in Figure G-1. In addition, the first 104-108 blocks of the system disk contain the system image. The contents of these blocks are shown in the system disk block layout (Figure G-2).



where:

Disk Type	xxx	yyy
RK	102	103
RF, RS	106	107
RP, RJ or RM	108	109

Figure G-2 System Disk Block Layout

G.2.1 Global Data

Global data are held in four basic ways: single numeric, double numeric, string, and 4-word double precision floating-point numeric. Further, each can be combined with a pointer. The data type is encoded in bits 5 through 7 of the first byte of each global entry. Table G-1 defines these codes. These are the values returned by the $D Function[1].

Table G-1

MUMPS-11 Data Type Codes

Data Code	Description
Ø	Undefined variable
1	Single numeric
2	String
3	Double numeric
4	Pointer
5	Single numeric plus a pointer
6	String plus a pointer
7	Double numeric plus a pointer
8	4-word floating point numeric[2]

A pointer node, if present, points to a block that contains the next lower level element in the array. The block can also have a continuation pointer in words 254 and 255 that points to more data at the same level. To illustrate this, consider the following example:

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(2,4.5Ø)	832.Ø1
↑ABC(3,87)	"ZZZ"

A diagram of the array ↑ABC is shown in Figure G-3.

[1]Refer to Chapter 4 of the *MUMPS-11 Language Reference Manual*.
[2]Stored internally with a data code of zero.

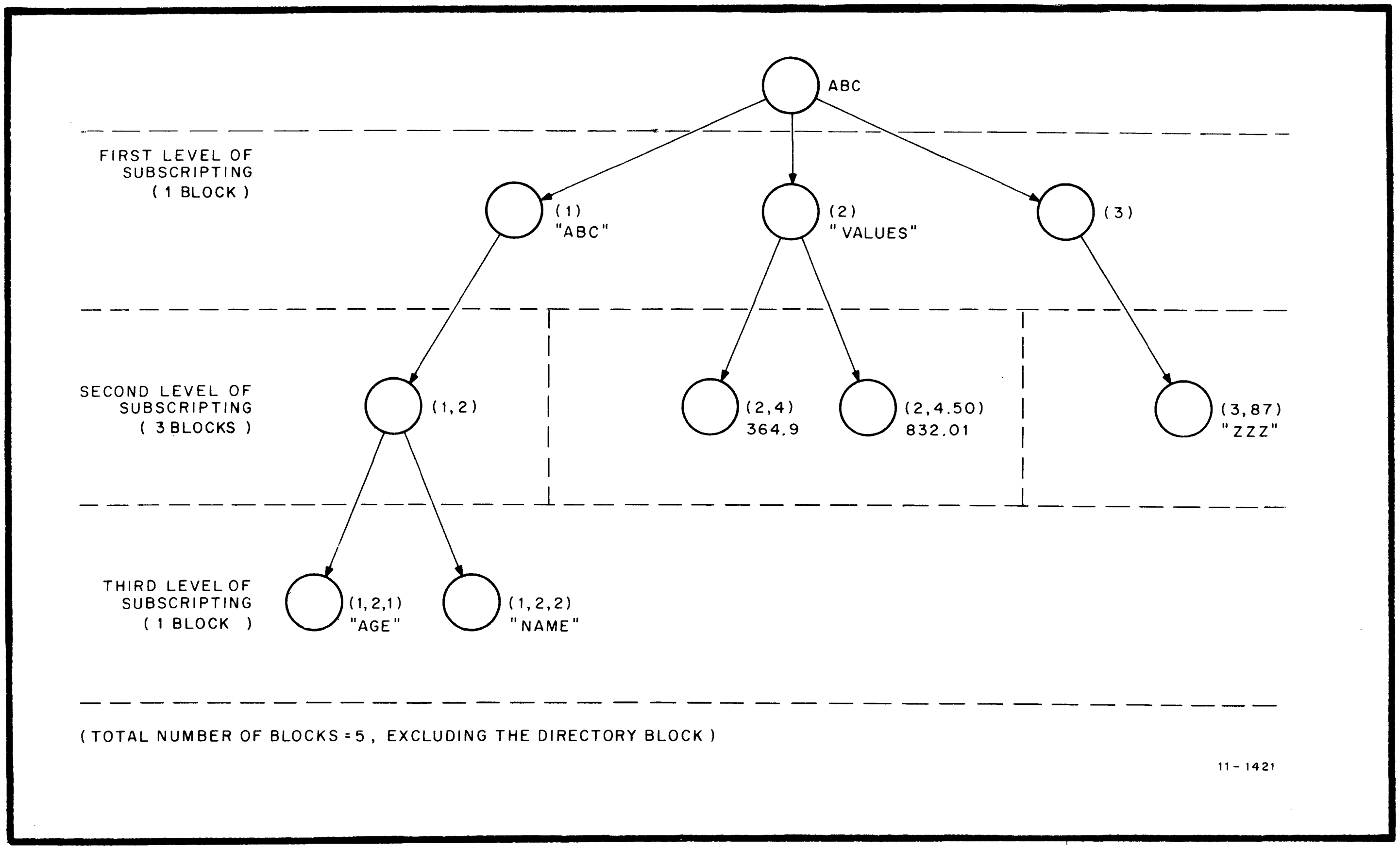
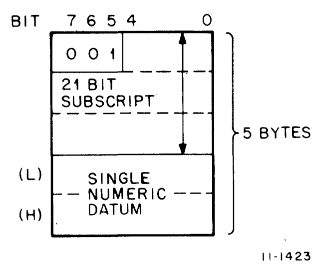


Figure G-3 Global Array Structures

All the data for level 1 are in the same block or in continuation blocks (i.e., blocks pointed to by word 255 of the previous block). Data for level 2 are pointed to by the data in level 1 and reside in a different data disk block. In no case does a disk block contain data on more than one level.

Each mode requires three bytes to specify the data type and subscript. The bytes following these are used to store the pointer (three bytes), if there is one, and then the data, if any, associated with that node. A node cannot exist if it has neither a pointer nor data. Thus, the smallest node in the system requires five bytes (either single-numeric or null string, and no pointer).

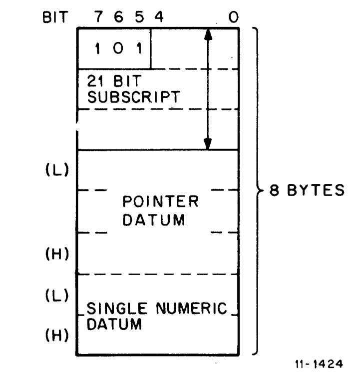
Figures G-4 through G-10 illustrate all possible configurations of global data.



($D() returns 1)

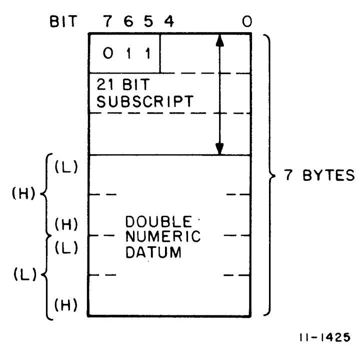
Figure G-4 Single Numeric Node

The 'L' and 'H' on the numeric datum (the last two bytes) show how the datum is represented internally in a word. That is, when the datum is assembled into a word, the first byte becomes the low-order byte of the word and the second byte the high-order byte.



($D() returns 5)

Figure G-5 Single Numeric with Pointer Node



($D() returns 3)

Figure G-6 Double Numeric Node

A double numeric datum is assembled into a 2-word grouping for internal calculations. The first two bytes shown above constitute the high-order word of the grouping; the second pair, the low-order word. Within each word, the bytes are assembled as for a single numeric datum.

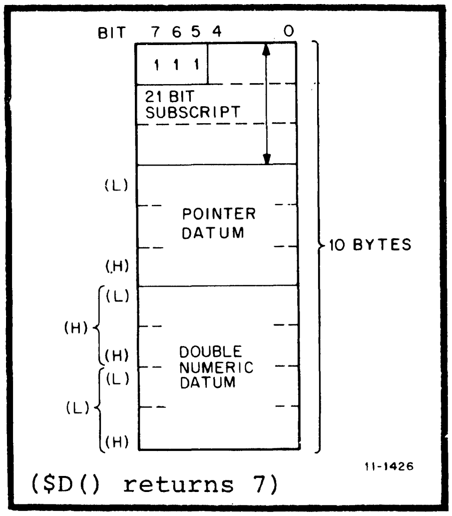


Figure G-7
Double Numeric with Pointer Node

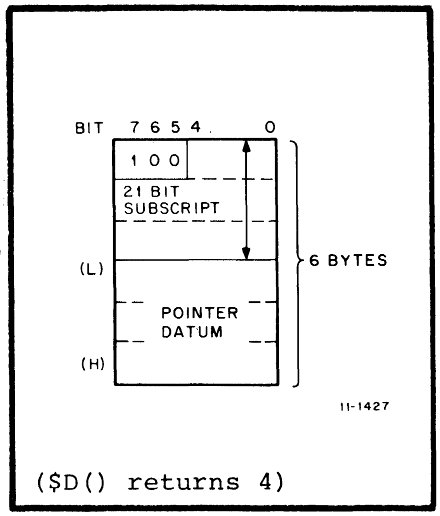


Figure G-8
Pointer Node

The three-byte disk pointers are stored with bits 0-7 in the low-byte, bits 8-15 in the middle byte and bits 16-23 in the high byte. If assembled into two words, the high byte (bits 16-23) of the pointer would occupy the low-order byte of the high-order word. The pointer bits are assigned as indicated below:

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Ø	Type		Unit Number			Block Number on Unit																	

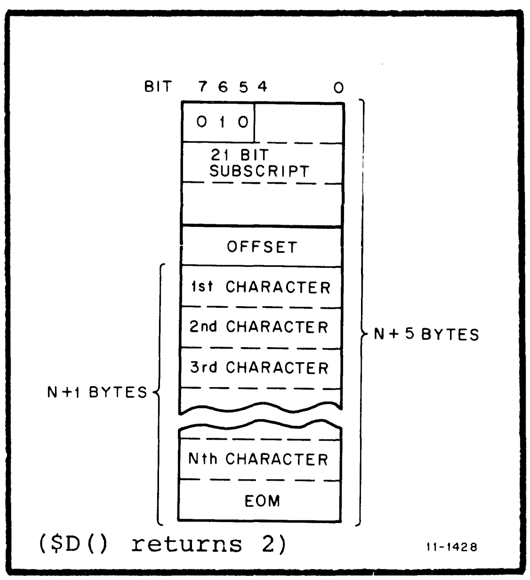
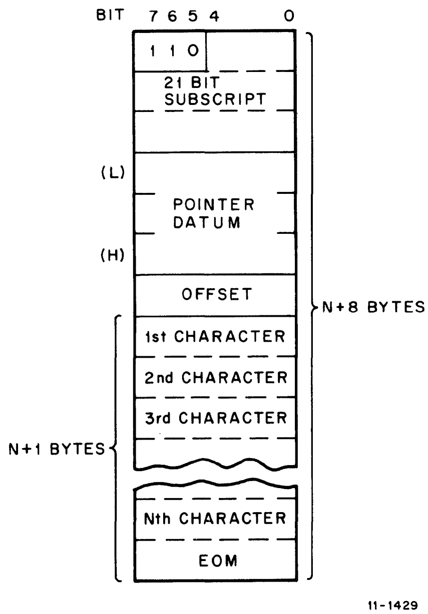


Figure G-9
String Node of "N" Characters

In string data, each character is stored in one 8-bit byte, with a null byte ending the string. An offset is stored as a byte at the beginning of the string to point to the next string entry.



EOM=End of message and is held internally as a null (0) byte.

($D() returns 6)

Figure G-10 String of "N" Characters with Pointer Node

A double-precision floating-point numeric datum requires 14 bytes of storage. The data type and 21-bit subscript are contained in the first three bytes. A pointer datum occupies the next three bytes. This entry is always present and contains a 0 if there is no lower level. The remaining eight bytes (four words) are used to contain the double-precision floating-point number as shown in Figure G-12. This format is identical to that employed by the PDP-11 Floating Point Processor (FPP-11B) and the PDP-11 Floating Point Math Package (FPMP-11).

NOTE

> The format is limited to normalized numbers. The high-order bit of the mantissa (which is always 1) is omitted in order to allow one more bit in the exponent field.

The sign of the number is bit 15 of the first word (word n). When this bit is set to zero, the number is positive. When it is set to one the number is negative.

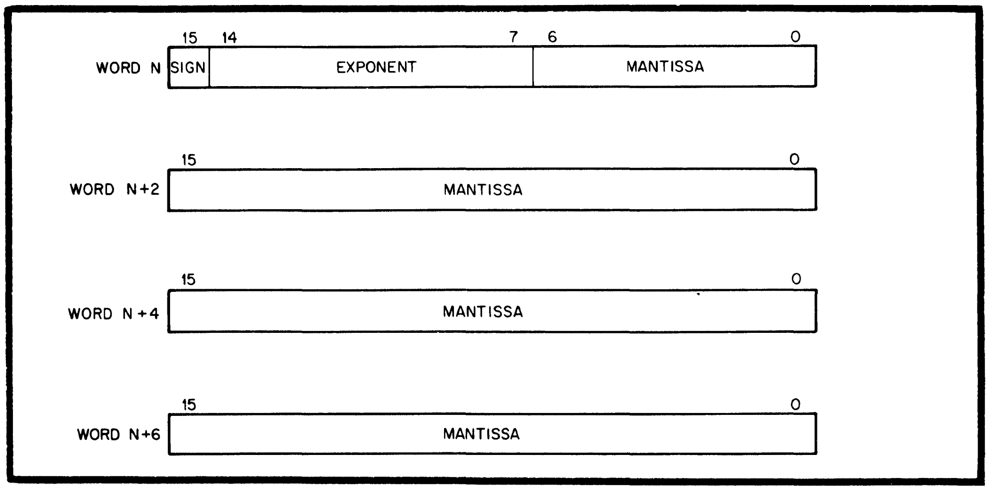


Figure G-11 Double Precision Floating Point Format

The exponent of the number is stored in bits 14 - 7 of word n, using excess 128_{10} (200_8) notation. The value of the exponent is obtained by subtracting 200_8 from the number formed by these bits.

The mantissa is stored in the remaining portion of word n and in words n+2, n+4, and n+6 as follows: the high-order portion in word n (bits 6 - 0), the intermediate-order portion in words n+2 and n+4, and the low-order portion in word n+6.

The byte representation of a double-precision floating-point datum is shown in Figure G-12.

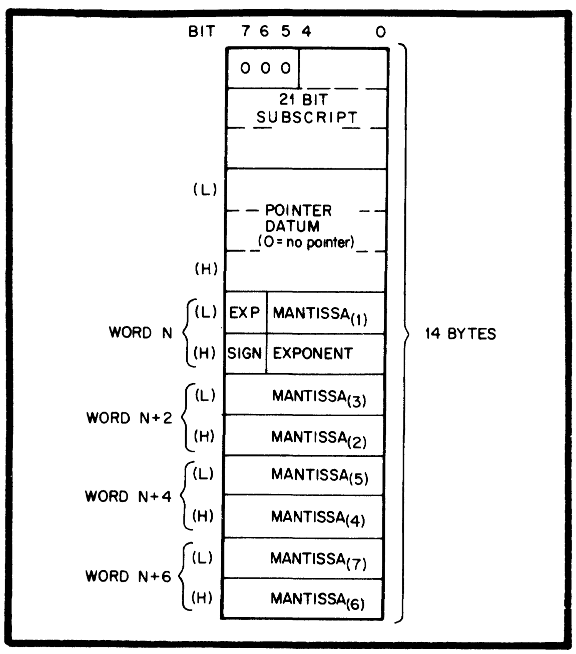


Figure G-12 Double Precision Floating Point Numeric Datum

G.2.2 Bit Maps

Bit Maps are used to keep track of which blocks are available for use (bit set) and which blocks are in use (bit clear). The exact number of maps, words, and the number of bits per word used in a map is dependent upon the physical characteristics of the disk being represented. The maps are two-dimensional; each word in a map represents a surface in a cylinder, and each bit position, starting at bit 0, represents a sector (block) on that surface (starting at sector 0). A collection of N contiguous words describe a cylinder and each map is made up of M complete cylinders. The number of maps needed to describe a disk is then the total number of cylinders divided by M.

A graphic representation of the bit maps is shown in Figure G-13.

G.2.3 Global Directories

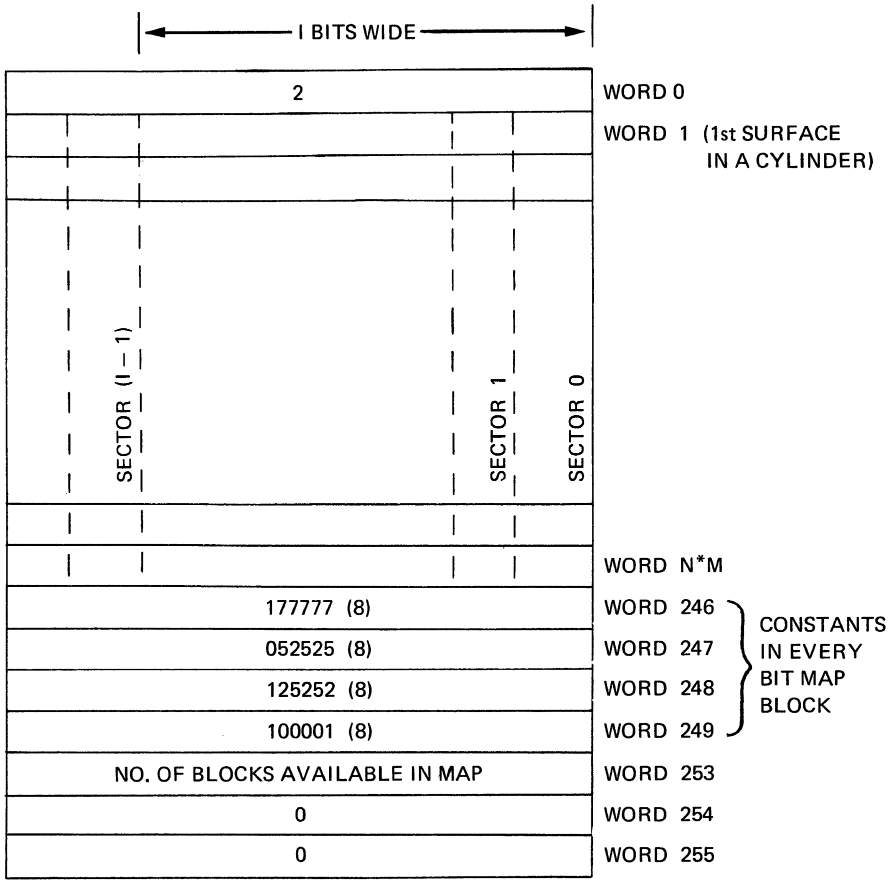
A Global Directory is created just like any other node. There may or may not be a pointer or other data associated with the node. The only difference is that the "subscript" is derived from the ASCII representation of the global's name. The pointer is to the first level of that particular global. If necessary, the block may have a continuation pointer in words 254 and 255 to another block of directory entries.

The directory block itself is pointed to by bytes 3-5 of the UCI Table entry. Byte 3 gives the logical disk number, and bytes 4 and 5 contain the block number of the first directory block.

G.2.4 Program Directories

A Program Directory is in the form of a single numeric datum with a pointer node. The pointer is to the first block of the program, and the single numeric portion is the two's complement of the program's word count.

The first program directory block is the next higher adjacent block from the global directory block. The global directory block number is in bytes 4 and 5 of the UCI Table.



N = NUMBER OF SURFACES IN A CYLINDER
M = NUMBER OF CYLINDERS PER MAP
I = NUMBER OF SECTORS ON A SURFACE

DISK TYPE	I	N	M	NO. OF MAPS PER – DRIVE/PLATTER
RK	12	2	100	2
RF	8	128	1	1
RS	16	128	1	1
RM	11	6	8	51
RP	10	20	1	200 (RP02)
				400 (RP03)
RJ	11	38	1	408

Figure G-13 Bit Map Example

G.2.5 Programs

Programs are stored on disk blocks in the same format as they are held in the Program buffer (Figure G-14). The Step Number is stored in two bytes as a 15-bit positive integer. The next byte is an offset to the next Step or Part Number. The text starts in the next byte and goes on to an EOM (a null byte). The end of the program is denoted by a null byte or by a null byte followed by a -1 byte (this is necessary to keep word boundaries intact for data following the program buffer).

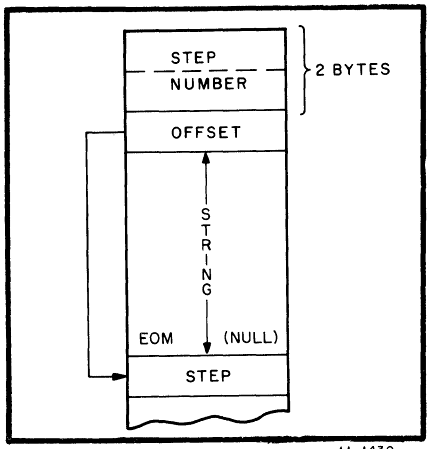


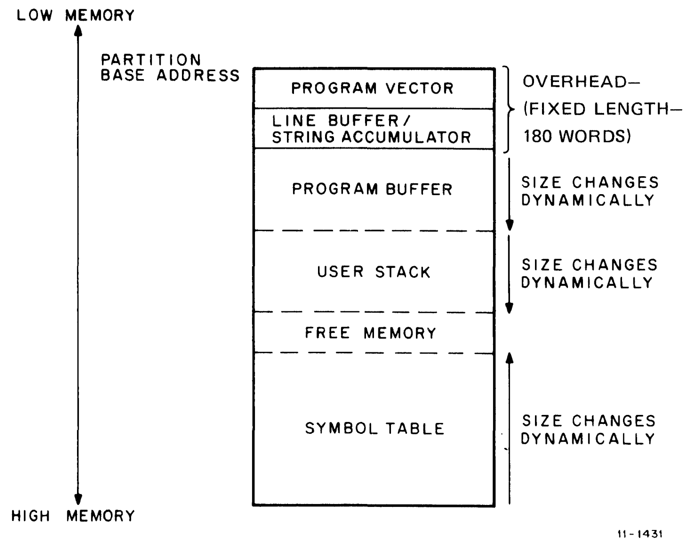
Figure G-14 Internal Program Format

On the disk, the first word of a block that contains a program is always a 2. This simply indicates that the block does not contain global data but is being used. The last two words of the block contain a continuation pointer to the next block in the chain or a null (0), if there are no more blocks.

G.3 PARTITIONS

Memory space above the MUMPS-11 system (executable code and tables) is divided into partitions (Figure G-15). The use of memory partitioning (as opposed to job swapping to and from secondary storage) allows the Executive to switch from one job to another in minimal time. Each job, as it enters the system, is assigned a memory partition and the job resides within that partition until termination (whether job-controlled or system-controlled). A partition is subdivided into four major subsections:

- Overhead
- Program Buffer
- User Stack
- Symbol Table



Note that while the direction of growth of the Symbol Table is from high to low memory, the actual Symbol Table entries are in ascending memory locations.

Figure G-15 MUMPS-11 Partition Layout

G.3.1 Program Vector

The Program Vector (Table G-2) is a section at the beginning of a partition which describes the 'status' of the job residing in that partition.

G.3.2 Line Buffer/String AC

The Line Buffer/String AC is a dual purpose buffer, 133 bytes long. It serves both as a buffer to contain lines being input or output via the terminal and as an accumulator for string expression results.

G.3.3 Program Buffer

The Program Buffer is the storage area for all Steps and Parts of a MUMPS-11 program. The contents of the Program Buffer can be modified in Direct Mode by adding and deleting Steps or Parts. In addition, commands which cause program loading such as CALL, OVERLAY and START affect buffer contents.

Table G-2

Program Vector Layout

Location	Contents
PV+Ø	Base address of partition; contains address to which control transfers at swap-in
+2	Temporary location
+4	Temporary location
+6	Stack Beginning (always contains a word address): pointer to the front of the stack (will be equal to or one greater than PV+16)
+8	Stack Pointer (always contains a word address): pointer to next free partition stack location (note that the partition stack grows from low-memory to high memory)
+1Ø	Line Pointer: points to current location in the line buffer/ string accumulator
+12	Character Pointer: points to current interpreter character location
+14	Program Pointer: contains address of current Step ($L)
+16	New Step Pointer: pointer to the next available location (byte) in the program buffer
+18	Program Header: points to the beginning of the partition's Program Buffer
+2Ø	Symbol Table beginning: points to the first byte of the last (top) entry in the local Symbol Table
+22	Principal I/O Device: identification number of principal device for current job
+23	Command: identifier for command currently being executed
+24	FOR Switch: Counter of the number of levels of nested FOR Commands on current line
+25	Indirection switch: Counter of the number of levels of indirection (←) on current line
+26	Global Header: pointer to most recently referenced Global Header Block
+30	Global Block: pointer to most recently referenced Global Data Block

(continued on next page)

Table G-2 (Cont.)

Program Vector Layout

Location	Contents
+34	Library Global Switch: set non-zero if a Library Global is the last referenced global
+35	Argument Switch: byte which indicates whether or not a command has arguments (Ø = no).
+36	Program Name (3 bytes): name of the program currently in the user's partition
+39	User Code (1 byte): User identification number (set at sign-on) Index into UCI table
+40	I/O device (1 byte): identification number of device currently assigned ($I)
+41	IF switch (1 byte): value (true or false) of the most recent IF command expression
+42	Job Status Word: bit Ø = programmer mode, when set (i.e., Login with PAC) bit 1 = software BREAK check; set = got BREAK bit 2 = external break enable/disable; set = enable bit 3 = CTRL/C received on terminal bit 4 = timed READ overrun bit 6 = Write via VIEW enabled bit 7 = Library Program and Global Write enable bit 12 = (delayed) 11/70 parity error in partition bit 13 = swap error bit 14 = delayed disk I/O error (error on submerged write) bit 15 = hardware break interrupt check; set = got interrupt
+44	$E System Variable
+46	$W System Variable

As shown in Figure G-16, the Program Buffer is byte-oriented. Entries consist of the Step Number in the first two bytes (a 15-bit positive integer in the range Ø-327.67), followed by an offset which points to the beginning of the next Step. The remainder of the entry consists of up to 132 bytes containing the characters in the Step. The last byte of an entry is always a null which is the internal representation of an EOM.

The last byte entry in the Program Buffer contains one of the following two data values:

a. Ø if the Program Buffer is an even number of bytes in total length (in fact, the Ø is the null at the end of the last Step), or

b. 377 if an extra byte is needed to pad the Program Buffer to be an even number of bytes.

These conditions are necessary because the User Stack, which follows the Program Buffer, must begin on a word boundary.

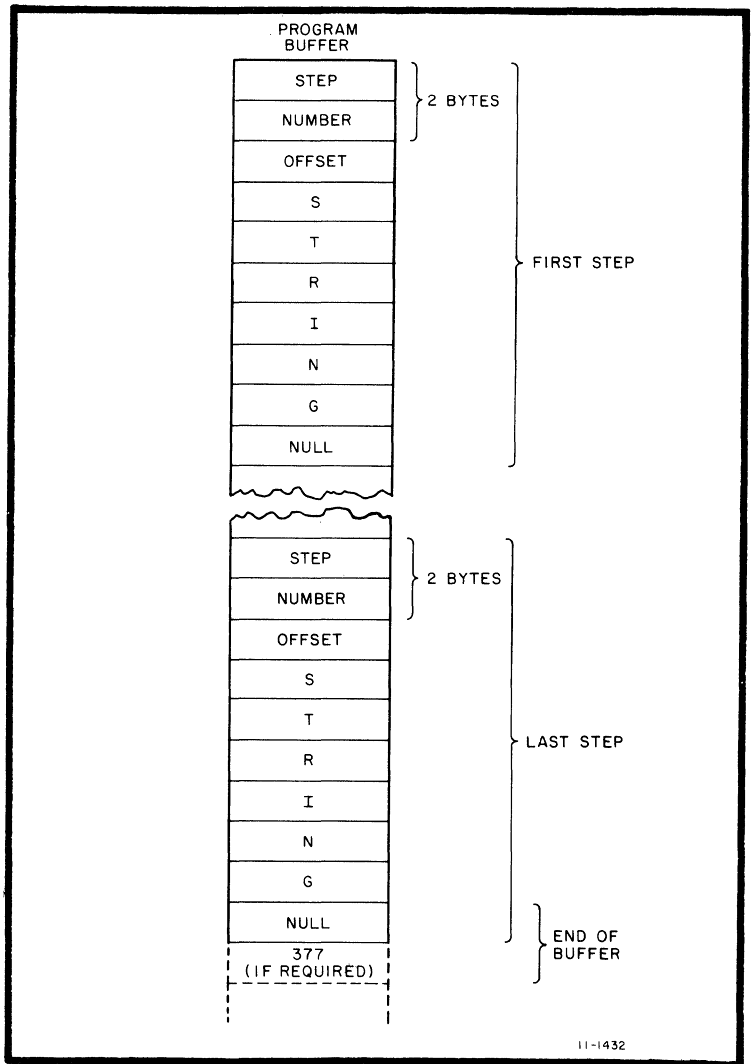


Figure G-16 Program Buffer Layout

G.3.4 User Stack

The User Stack dynamically increases and decreases in size relative to the requirements of the current Job. Its base address and hence the whole stack as a block, shift as a program increases or decreases in size, since the stack always follows the Program Buffer.

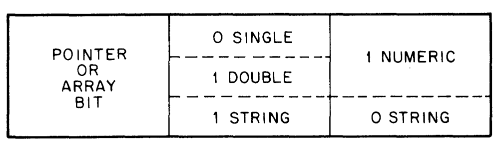
G.3.5 Free Memory

The area between the User Stack and the Symbol Table is defined as Free Memory. The amount of Free Memory available at any one time is contained in the $S System Variable. The Free Memory area is used to permit expansion of the dynamic areas of the partition. The Program Buffer and User Stack grows from and shrinks back to the low end of Free Memory. The Symbol Table grows from and shrinks back to the high end of Free Memory.

G.3.6 Symbol Table

The Symbol Table contains entries for all defined local variables, subscripted or unsubscripted. Its size varies as variables are defined (SET Command) and deleted (KILL and XKILL Commands) by the job running in the partition. The top of the Symbol Table (i.e. next available byte for an entry) is pointed to by PV+2Ø. A Symbol Table entry is variable length. The entry descriptor defines the fields of the entry. Decimal numbers in MUMPS-11 are manipulated as 2-word quantities; however, they are stored as 1-word quantities in a Symbol Table entry if their range is ±327.67.

The maximum range of numbers is ±21474836.47. String data in the Symbol Table is followed by a null byte to indicate EOM. Arrays are stored in the Symbol Table as ordered (but sparse) elements. Subscripts are 21-bit quantities, hence the largest subscript is 20971.51. Arrays in the Symbol Table have an associated simple variable preceding them. This simple variable (having the same name as the subscripted variable) contains a pointer around the array (to facilitate table searches) and may also contain a string or numeric data. Thus, a subscripted variable array with only a single element has an associated simple variable which may or may not be explicitly defined. Entries in the Symbol Table for each array element, therefore, do not include the array name, but only the subscript value followed by associated data. The code bits (high order three bits of the first byte of a Symbol Table entry) determine the type of entry in the table, as shown in Figure G-17.



Data Code			Description
Ø	Ø	Ø	4 word floating point numeric[1]
Ø	Ø	1	Single numeric (signed 15-bit)
Ø	1	Ø	String
Ø	1	1	Double numeric (signed 31-bit)
1	Ø	Ø	Array offset
1	Ø	1	Array offset and single numeric
1	1	Ø	Array offset and string
1	1	1	Array offset and double numeric

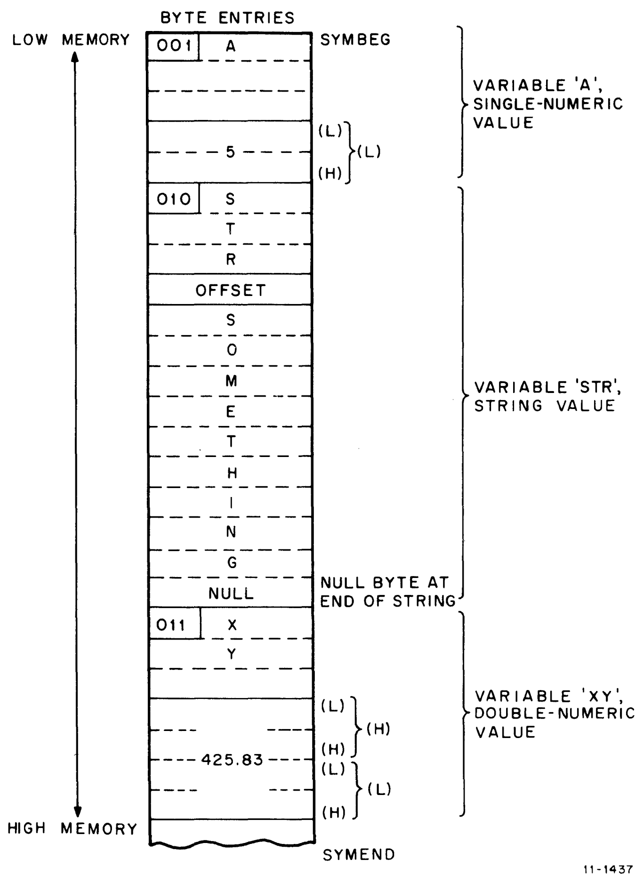
[1] Reported as $D()=8

Figure G-17 MUMPS-11 Symbol Table Data Codes

Double numeric data are stored in the symbol table in the following order:

high-order word	high-order byte
	low-order byte
low-order word	high-order byte
	low-order byte

Figures G-18, G-19, and G-20 show the formats of the various types of data entries in the Symbol Table.

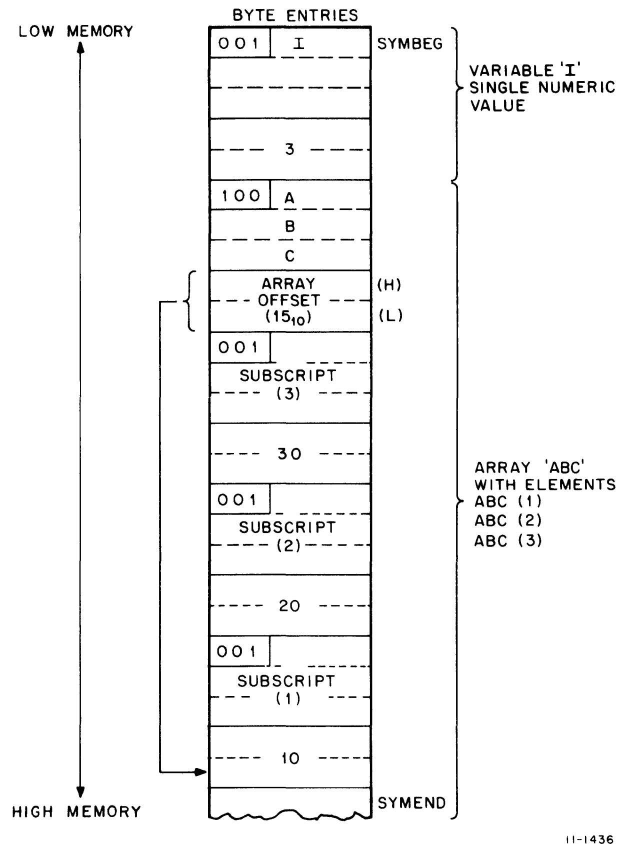


NOTE

Assuming an empty Symbol Table as an initial condition, the entries in the table above could have been made by the following command string:

>SET XY=425.83,STR="SOMETHING",A=5

Figure G-18 Symbol Table Containing Three (Simple) Variables



NOTES

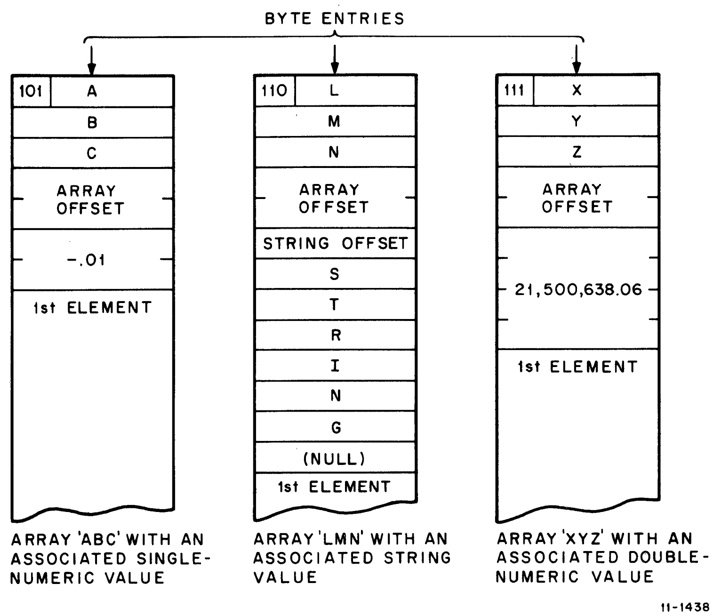
The above symbol table could have resulted from the following command string:

>FOR I=1:1:3 SET ABC(I)=10*I

1. *There is no datum associated with the array name.*

2. *The array offset is a count of the number of bytes contained in the array.*

Figure G-19 Symbol Table Containing a Simple Variable and 3-Element Array



NOTES

1) *The datum associated with the array name may have been defined (as a simple variable) either before or after the array was defined.*

2) *This datum (single-numeric, string, or double-numeric) immediately follows the array offset, preceding the first element in the array.*

3) *The array offset (byte count) includes the number of bytes of data associated with the array name.*

4) *If the array is KILLed, the datum associated with the array nan is also KILLed.*

Figure G-20 Array Entries Having a Datum Associated with the Array Name

APPENDIX H
CRC REFERENCE SUBROUTINE

```
CPUDSC  RSTS  MACRO VM02-10   26-OCT-75 00:17:06 PAGE 29
CPU SUBROUTINES AND UTILITIES

                       ;
                       ;****************
                       ;************************
                       ;
                       ;CRCHK
                       ;
                       ;UPDATES CURRENT CRC TOTAL
                       ;
                       ;
                       ;ENTRY:
                       ;        (R3)=CHARACTER
                       ;        R1 POINTS TO THE DDM
                       ;EXIT:
                       ;        R3 INCREMENTED BY ONE
                       ;
   03412               CRCHK:
   03412 010046                MOV     R0,-(SP)                ;SAVE R0
   03414 010546                MOV     R5,-(SP)                ;SAVE R5
   03416 112305                MOVB    (R3)+,R5        ;GET CHAR.
   03420 042705                BIC      #HIBYTE+PARITY,R5
         177600
   03424 016100                MOV     CRC(R1),R0      ;GET PREVIOUS CRC
         000016
   03430 004767                JSR     PC,XOR          ;PERFORM XOR WITH R5 AND R0
         000046
   03434 012746                MOV     #8.,-(SP)       ;SET UP COUNTER
         000010
   03440 000241 1$:            CLC                     ;SO BIT 15=0 ON ROR
   03442 006000                ROR     R0
   03444 103004                BCC     2$
   03446 012705                MOV     #120001,R5              ;SET UP SOURCE
         120001
   03452 004767                JSR     PC,XOR          ;XOR THEM
         000024
   03456 005316 2$             DEC     (SP)            ;LOOP THRU?
   03460 001367                BNE     1$
   03462 005726                TST     (SP)+           ;POP COUNTER
   03464 042700                BIC     #100200,R0
         100200
   03470 010061                MOV     R0,CRC(R1)      ;UPDATE CRC
         000016
   03474 012605                MOV     (SP)+,R5                ;RESTORE R5
   03476 012600                MOV     (SP)+,R0        ;RESTORE R0
   03500 000207                RTS     PC
                       ;*******
                       ;
                       ;SUBROUTINE TO PERFORM EXCLUSIVE-OR BETWEEN R5 AND R0:
                       ;
   03502               XOR:
   03502 010546                MOV     R5,-(SP)
   03504 040016                BIC     R0,(SP)
   03506 040500                BIC     R5,R0
```

```
03510 052600          BIS     (SP)+,R0
03512 000207          RTS     PC
```

APPENDIX I
WRITING AND INTERFACING I/O DEVICE DRIVERS

I.1 INTRODUCTION

The MUMPS-11 system provides the user with the option of expanding the system's configuration with up to four additional devices, which may be assigned to the reserved device numbers 51-54. However, if the card reader is to be installed as part of the system configuration, device number 54 will be assigned to the card reader. This will mean that device number 54 cannot be assigned to another user device. The user must write his own device drivers in the MACRO Assembly Language to handle the I/O operations associated with the additional devices.

A MUMPS-11 I/O device driver follows certain system naming conventions and uses several MUMPS system (global) routines to perform standard operations. The following paragraphs are intended to provide the application system programmer with the information necessary to write a device driver using these conventions and routines. A knowledge of MACRO-11 Assembly Language is assumed[1].

If more than one kind of device is to be added to the system, the user must write a separate driver for each particular device. Furthermore, he will also have to write the routines for associating the device number with the appropriate driver. Refer to paragraph I.4 for details.

Object modules of all user-written device drivers must be incorporated (linked) into the MUMPS system during system build. The *MUMPS-11 Operator's Guide* describes the linking procedure.

I.1.1 Device Driver Functions and Routines

The primary functions of an I/O device driver include:

- initializing the driver when the device is ASSIGNed
- handling any special device functions made with an ASSIGNment
- setting up buffers for the temporary storage of data during I/O transfers

---

[1]For details on the MACRO language, the user may refer to the *MACRO-11 Assembly Language Programmer's Manual* (DEC-11-OMACA-A-D). He may also find the appropriate *PDP-11 Processor Handbook* and the *PDP-11 Peripherals and Interfacing Handbook* useful.

- answering calls from the MUMPS-11 Language Interpreter to perform an I/O operation
- transferring data from a device's buffer to a user's program partition buffer and vice versa
- servicing device interrupts
- reporting error conditions
- cleaning up when the device is UNASSIGNed

These functions correspond, in general, to the types of routines which must be included in the driver.

In order to interface properly with the MUMPS system, the driver must include at least six routines:

- an initialization routine named RESASG
- a clean-up routine named RSUASG
- an input handling routine--named by system convention XXXIN (where XXX is a 1 to 3-character mnemonic for the device)
- an output handling routine--named by convention XXXOUT
- an error reporting routine--named by convention XXX$A
- device interrupt service routine(s)

All routines, except the interrupt service routines, are called by the MUMPS-11 Language Interpreter when processing the job currently in the Run Queue. The specific routine called depends on the I/O operation requested by the job.

The distinction between the routines called by the Interpreter and the interrupt service routines is important. If not written as distinct routines, a job might enter an indeterminate state if a process initiated from one set of routines is interrupted from the other before the first processing is finished. The user should note, therefore, that jobs may be "hung" only from Interpreter-called routines and "waked" only from interrupt processing routines (see paragraphs I.2.4 and I.3.2).

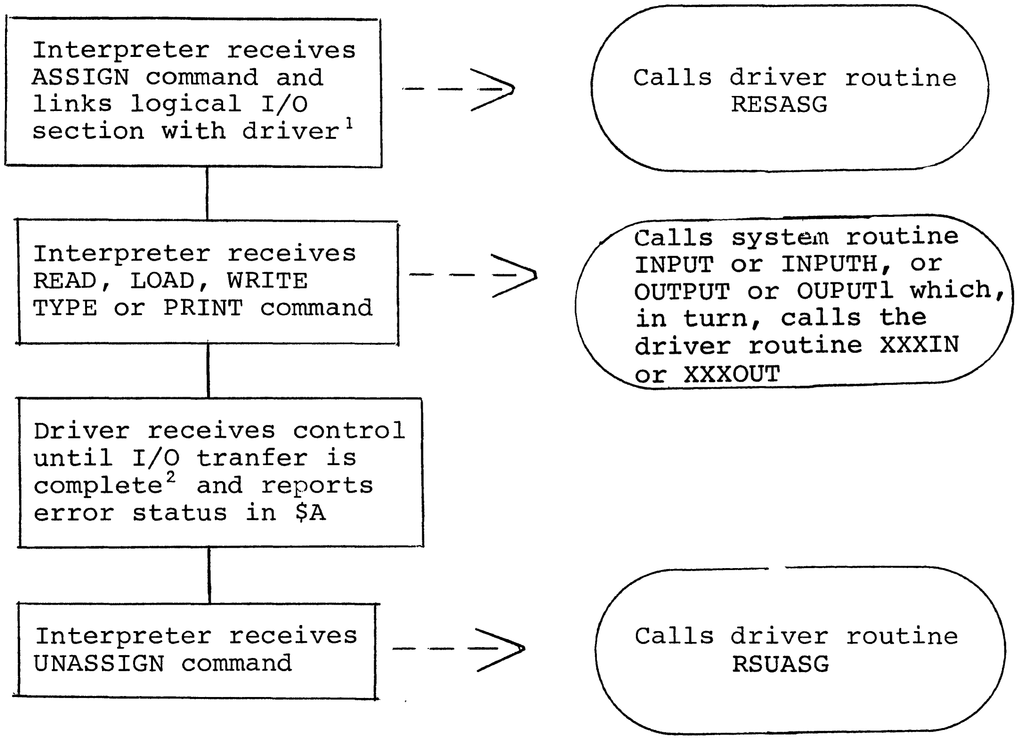


Figure I-1 Driver Operation

I.1.2 System Global Variables and Routines

The driver uses several global system routines and variables when handling standard operations. Table I-1 lists the global system routines and variables which may be used by either Interpreter-called routines or interrupt service routines. Table I-2 lists the globals which may only be used by Interpreter-called routines.

The global routine named QJOB which "wakes" a job may only be called by interrupt service routines (see I.3.2).

---

[1]The logical I/O section re-establishes the link each time the job which issued the ASSIGN command is swapped-in until the device is UNASSIGNed.

[2]On input, driver retains control until the entire line is packed into the partition's line buffer. Output may be submerged depending on the device and buffer structure.

Table I-1

Global Routines and Variables

NAME	FUNCTION
LBSIZE (byte)	The maximum number of characters that a driver may put in the current job's line buffer; the maximum size of a line input (132_{10}).
ILGINT (word)	The number of illegal interrupts which have occurred in the system. The driver should increment the location each time it detects an illegal interrupt.
INHIB	The interrupt priority level used to shut out interrupts. Only necessary for Interpreter-called routines to avoid race conditions (interrupt processing routines are normally entered at level 6). The driver may raise the priority level before making checks which may hang the job by issuing the instruction: MOVB #INHIB,PSW
DEVTAB	The device table (see Appendix F)
JOBTAB	The job table (see Appendix F)
SAVREG	Routine which saves registers R0-R5 (see I.3)
RESREG	Routine which restores registers R0-R5 (see I.3)
NUMAC (2 words)	The two-word numeric accumulator (see I.2.1.1)

Table I-2

System Global Variables and Routines Called by Interpreter-called Driver Routines

NAME	FUNCTION
LINPTR (word)	The partition overhead location (indexed by R4) which contains the address of the line buffer within the partition (see I.2.4 and I.2.5).
LBHDR (word)	The base address of the line buffer in the current job's partition (see I.2.4 and I.2.5).
$I (byte)	The partition overhead location (indexed by R4) which contains the number of currently ASSIGNed device.
JBSTAT (word)	The partition overhead location (indexed by R4) which contains the job's status ($J) (see I.2.4.2).
TMP1 (word)	The partition overhead location (indexed by R4) which contains the time interval on a timed READ.
$A (2 words)	The $A System Variable (see I.2.6).
CURRX (byte)	The $X System Variable. The driver can calculate $X on output if applicable to the device. It is handled like $A and should be in the XXX$A routine (I.2.6).
CURRY (byte)	The $Y System Variable; same as CURRX above.

LBOV	The terminal error message processed by the system for line buffer overflow. The driver may issue it as illustrated below, where R5 contains the character count:
	`        CMPB #LBSIZE,R5  ;too many characters?`
	`        BGE ERROR`
	`          .`
	`          .`
	`          .`
	`ERROR: LBOV              ;yes, terminal error`

ADREVL	Routine which checks for ASSIGN command arguments (see I.2.1.1).
DIV100	Routine which divides contents of NUMAC by 100 (see I.2.1.1).
RBREQ	Routine which requests a ring buffer from the system's buffer pool (see I.2.3.2).
RTNRB	Routine which returns a ring buffer to the pool (see I.2.3.2).

(continued on next page)

Table I-2 (cont.)

System Global Variables and Routines Called by Interpreter-called Driver Routines

NAME	FUNCTION
PACK	Routine which transfers one line from the job's output buffer to the device's buffer (see I.2.3.5).
UNPACK	Routine which transfers one line from the device's buffer to the job's line buffer (see I.2.3.4).
DQJOB	Routine which hangs a job (see I.2.4.1).
CLKIOR	Routine which hangs a job for a timed READ (see I.2.4.2).

SWPOUT	Routine which hangs a job for a time slice. No calling conditions other than it must be called at priority level 6. The programmer may not assume that any registers are intact on return. Calling sequence:
	`MOVB #INHIB,PSW`
	`JSR PC,SWPOUT`

RESDEV	First of 3 words in IOD module into which reserved device handler must enter the addresses (in stated order) of
	`RESDEV: XXX$A`
	`    +2: XXXIN`
	`    +4: XXXOUT`
	routines. These are the locations which tie the system routines 'INPUT and 'OUTPUT' to the appropriate routine in the user-created driver.

I.2 INTERPRETER-CALLED ROUTINES

I.2.1 Device ASSIGNment--Routine Name: RESASG

The system calls the driver when the Interpreter receives an ASSIGN command for the driver's device. This allows the driver to perform any initial operations that may be necessary. For example, the terminal driver initialization routine turns off the CTRL O feature if it was in effect and processes a right margin specification if it is present in the ASSIGN command. If there are several reserved device drivers, the routine would also direct a call to the appropriate driver for the ASSIGNed device (see I.4).

The routine to process the ASSIGN for a reserved device must be named RESASG and be defined by a .GLOBL statement in the device driver. RESASG must perform the operations necessary to initialize the driver and the device for the following conditions:

a. the device is not already 'owned' by the job
b. the device is already 'owned' by the job.

On entry to the routine, R5 contains the device number in its low byte, the exit is accomplished via a 'RTS PC' instruction sequence. No return conditions are necessary.

I.2.1.1 Handling Optional Syntax on the ASSIGN Command

The ASSIGN command may contain information that the driver must process. For example, an ASSIGN command for a DECTape device may contain the byte address for the I/O operation. The system's global routine ADREVL is used to look at the character immediately following the device number in the ASSIGN statement, and to set the appropriate condition codes. The command line is currently in the partition's line buffer.

Routine Call:

```
     JSR PC,ADREVL      ;no entry conditions
```

Return:

```
     +                  ;no colon argument present
     0                  ;the colon argument is False
     -                  ;the colon argument is True (returns
                         a non-zero value)
```

No register may be assumed to be intact on return.

If the driver needs the actual numeric value of the optional ASSIGN command argument for further processing, it must call the system's global routine DIV100 to divide the number by 100, since it being carried as a MUMPS integer. DIV100 places the result in the numeric accumulator.

Routine Call:

```
JSR PC,DIV100       ;no entry condition
```

Return:

```
NUMAC               ;contains high-order numeric value
NUMAC+2             ;contains low-order value
```

No registers may be assumed to be intact on return.

EXAMPLE:

The following code sequence sets the right margin on a terminal.
MUMPS command line: A 1:72
Driver ASSIGN Code.

```
.GLOBL ADREVL,DIV100,NUMAC
MOV R1,-(SP)                ;save address of device descriptor
                            ;buffer (DDB) for terminal
JSR PC,ADREVL               ;get optional argument
BGE DONE                    ;if there is none, quit
JSR PC,DIV100               ;otherwise divide argument by 100
MOV (SP)+,R1                ;get the DDB address saved
MOVB NUMAC+2,MARGIN (R1)    ;set user-specified margin
```

I.2.2 Device UNASSIGNment--Routine Name: RSUASG

The driver is called when the Interpreter detects an UNASSIGN command for that driver's device to allow the driver to perform any necessary clean-up operations. For example, when the terminal driver's RSUASG routine is called, it sets up a wait condition to allow any current operation to finish (since in this case, output may be submerged), returns the ring buffer to the system's buffer pool, and resets the right margin to zero.

Any device driver must have a global routine named RSUASG, even if it is only a return to the caller. On entry to the routine, R5 contains the device number in the low byte. Return is through RTS PC, and no special conditions are necessary.

I.2.3 Buffers

A programmer may provide temporary storage space for I/O data in one of three ways:

1. include the necessary buffers within the driver itself
2. request 64-character ring buffers from the system's pool[1]
3. use the 512-character buffers automatically provided by the system from the system's buffer pool

The system supplies the addresses of buffers obtained from the buffer pool in System Table entries SYSTAB+76 (device number 51), SYSTAB+78 (device number 52), SYSTAB+80 (device number 53), and SYSTAB+82 (device number 54).

I.2.3.1 Internal Buffers - If the programmer includes buffers within the driver and is not going to use buffers from the system buffer pool, the driver routine RESASG must pass the buffer address to the system when the device is ASSIGNed. The code is:

```
MOV #BUFADR,@2(SP)
```

When the device is UNASSIGNed, the RSUASG routine must remove the buffer address from the system as follows:

```
CLR@2(SP)
```

I.2.3.2 64-Character Buffers - If the programmer wishes to request a 64-character buffer from the system, the driver competes with terminal drivers for buffers. Each ring buffer is 64 bytes (32 words) long. The last 6 bits of a buffer's starting address contain zeroes. The last location in a buffer, therefore, is the starting address plus 77_8.

The driver must request a 64-character buffer and pass the buffer address onto the stack when the RESASG routine is called, and return the buffer to the pool when the RSUASG routine is called. The driver uses the global routine RBREQ to request a 64-character buffer. This routine returns the buffer address in register 5 (R5). This address must be placed on the stack. The calling sequence is:

```
.GLOBL RBREQ
JSR PC,RBREQ
MOV R5,@2(SP)
```

[1]Buffers obtained from the system's buffer pool must be allocated during system generation.

All registers are destroyed. If no buffer is available, the job is hung in the proper buffer resource bound state until one is available.

The driver uses the global routine RTNRB to return the buffer when the device is UNASSIGNed. The driver must pass the address of the buffer to RTNRB in R5. The calling sequence is:

```
.GLOBL RTNRB
JSR PC,RTNRB
CLR @2(SP)          ;R5 has been set to buffer address before this
```

I.2.3.3 512-Character Buffers - If the driver within RESASG did not allocate a buffer as described in I.2.3.1 or I.2.3.2, the system will then allocate a 512-character buffer to the driver when the device is ASSIGNed and will automatically return the buffer to the pool when the device is UNASSIGNed. The driver competes with magtape, DECtape Sequential Disk Processor, and VIEW (disk) command devices for buffers. Each buffer is 512 bytes (256 words) long and always starts on a word boundary.

I.2.3.4 Unpacking Buffers - The global routine UNPACK takes a line from the device buffer and puts it into the buffer to which R0 points. Lines are terminated by legal ASCII line terminators: LINE FEEDS (12_{10}), vertical TABs (13_{10}), and FORM feeds (14_{10}). Carriage RETURNS (15_{10}) are ignored. A null (0) causes a return with the N bit set. A single call can unpack a maximum of 132 characters.

To call UNPACK:

- the address of the line buffer into which the characters are to be inserted must be in R0
- a pointer to a 2-word block (in the driver) must be in R2;

  --word 1 contains a pointer to the device buffer from which characters are to be taken

  --word 2 contains a negative count of the number of bytes left in that device buffer.

Calling Sequence:

```
.GLOBL UNPACK
JSR PC,UNPACK               ;R0 and R2 already set up
```

On return, the driver may expect:

- a pointer in R0 to the next free location in the line buffer just packed
- the words pointed to by R2 updated:

  --word 1 points to the next character to unpack from the device buffer

  --word 2 contains the updated negative count (0 if the buffer is empty)
- the condition codes indicate the following:

  --Z bit set means that the exit was successful (i.e., a line terminator was encountered)

  --N bit set means line buffer overflow (i.e., no line terminator encountered)

  --no bits set means that the buffer is full
- R1, R3, R4, R5 are the same as on entry; R2 is destroyed

I.2.4 Interpreter Call for Input--Routine Name: XXXIN

When the Interpreter detects a READ or LOAD command in the current job, it does a JSR PC,INPUT to the logical I/O section of the system module IOD. The logical I/O section does a JSR PC,XXXIN to the I/O driver which was linked to the job when the ASSIGN command was issued.

The driver routine may expect the following conditions to exist when it is called:

- the base address of the partition in R4
- the address of the user partition line buffer in the globally-tagged location LBHDR in the MUMPS Executive
- two entries on the stack: the return address to the logical I/O section and the value for a timed READ (see below). The programmer should pop both of these entries off immediately to enable eventual return directly to the Interpreter and handle timed READs if necessary. If timed READs are to be processed, the second entry must be used as described in I.2.4.2.

MUMPS expects the following conditions when the driver returns to the Interpreter:

- the base address of the partition in R4 (same as entry)
- the base address of the user partition line buffer in the globally-tagged location LINPTR in the user partition overhead. The code is:

```
.GLOBL LBHDR,LINPTR
MOV LBHDR,LINPTR(R4)
```

- the complete line of input in the user partition line buffer (I.2.3.4). This line may contain not more than 132_{10} 7-bit ASCII characters (one per byte) terminated by a null (0) byte.
- the current error status in the $A System Variable (see I.2.6).

I.2.4.1 Hanging a Job on Input - In order to allow other MUMPS jobs to run, the driver should hang a job when the complete line of input is not in the driver's buffer and some device activity (e.g., tape rewind) is required to obtain the data. The driver accomplishes this by passing a word to DQJOB in R5 which contains a unique hang code for the device. The word should be in the form:

- bit 15 set to 1--indicates that the job is I/O hung
- bits 14 through 8 contain a unique code
- bits 7 through 0 set to 0--the system will put a job number in these bits

NOTE

The following codes for bits 14-8 are already assigned and must not be used:

140000_8 -- terminals

142000_8 -- magtapes

144000_8 -- DECtapes

The driver's interrupt service routines should then request the Executive to return a job to the run queue when the required device activity is completed (see I.3.2). The driver can therefore assume upon return from DQJOB that the buffer is full or that a line terminator has been received.

Example:

The following two instructions hang a job until terminal I/O is finished.

```
HANG:  .GLOBL DQJOB
       MOV #140000,R5      ;terminal hang code
       JSR PC,DQJOB        ;hang the job until I/O is done
       (BR MORE)           ;go get more characters
```

I.2.4.2 Handling Timed READs - Although the timed READ syntax is primarily intended for terminals, any device may use it. Upon entry, the driver can expect the second entry on the stack to contain the time interval specified with the READ; if there is none, the stack entry is 0.

If the job needs to be hung to wait for input, the driver must put the job in the clock queue rather than the I/O bound queue, by calling the CLKIOR global routine. By convention, the driver must return a null line to the job and set bit 4 in the $J System Variable (global tag JBSTAT) if a timed READ fails.

CLKIOR returns to the driver on one of three conditions:

- a full buffer was input from the device
- a line terminator was received from the device
- the specified time was up

The first two conditions must be detected by the interrupt service routine which must, in turn, signal the executive as shown in I.3.2.

The driver must distinguish between the first two conditions and the third.

Example:

Suppose that the driver set a word labelled "INPUT" to a non-zero value if some data was received from a device; the driver did not pop entries off the stack upon entry. The following code sequence will perform a timed READ.

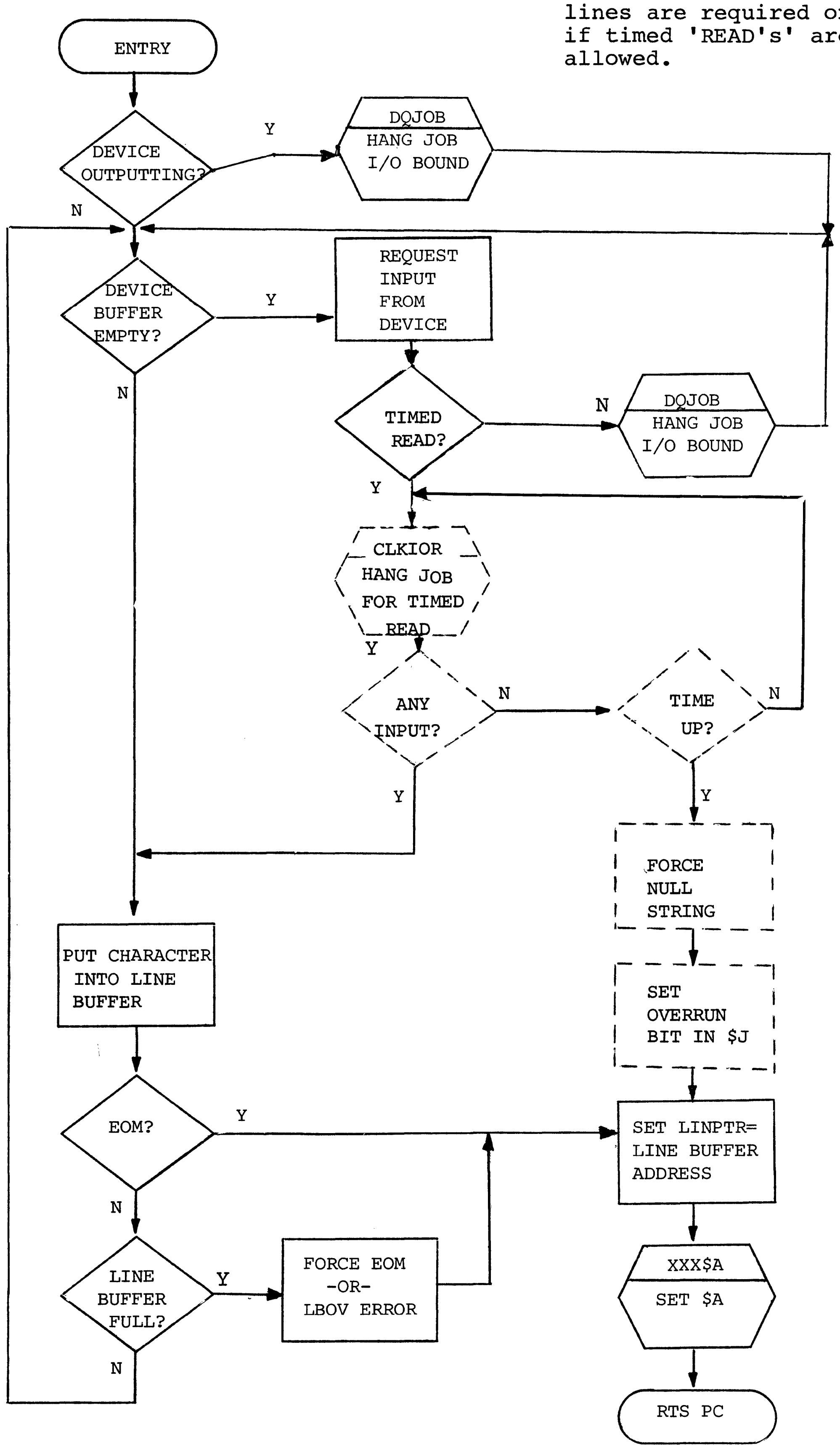
```
.GLOBL CLKIOR,JBSTAT,LBHDR
MOV 4(SP),R5                 ;non-zero time interval?
BEQ HANG                     ;then go to HANG (as above)
JSR PC,CLKIOR                ;else, hang job in clock queue
TST INPUT                    ;did any input occur?
BNE MORE                     ;yes--go process it
BIS #20,JBSTAT(R4)           ;no--set bit 4 in $J
MOV LBHDR,R0                 ;reset to beginning of line buffer
CLRB (R0)                    ;return null string
(go to normal exit)
```

Due to a time-sharing conflict, the system may wake a job prematurely that was hung for a timed READ. The time interval, stored in TMP1 in the partition's overhead, will not be zero. The following code expands the previous example to avoid a premature wake condition.

```
        .GLOBL CLKIOR,TMP1,JBSTAT,LBHDR
        MOV 4(SP),R5                  ;non-zero time interval?
        BEQ HANG
TIMHNG: JSR PC,CLKIOR                 ;hang job for time out
        TST INPUT                     ;did any input occur?
        BNE MORE                      ;yes, go process it
        MOV TMP1(R4),R5               ;did timed READ time out? (0)
        BNE TIMHNG                    ;go hang for remainder of
                                       time
        BIS #20,JBSTAT(R4)            ;time overrun, set bit 4
                                       in $J
        MOV LBHDR,R0
        CLRB (R0)                     ;return null string
        (go to normal exit)
```

I.2.4.3 Sample Input Routine Flowchart

Note: Portions shown in dashed lines are required only if timed 'READ's' are allowed.



I.2.5 Interpreter Call for Output--Routine Name: XXXOUT

The dispatch to the output routine is the same as that for input, except that it is called when the Interpreter detects a TYPE, PRINT, WRITE or output in a READ command; they all require the same driver activity.

The driver may expect the following conditions upon entry:

- R4 contains the base address of the partition
- R0 contains the address plus one of the line buffer that contains the line to be output
- R3 (low byte) contains the first character to be output
- the return to the logical I/O section on the stack (may be discarded immediately)

MUMPS expects the following conditions when the driver returns:

- R4 contains the base address of the partition (same as entry)
- the base address of the line buffer in the word labelled LINPTR in the partition overhead; thus:

```
.GLOBL LBHDR,LINPTR
MOV LBHDR,LINPTR(R4)
```

- the latest error status in the $A System Variable

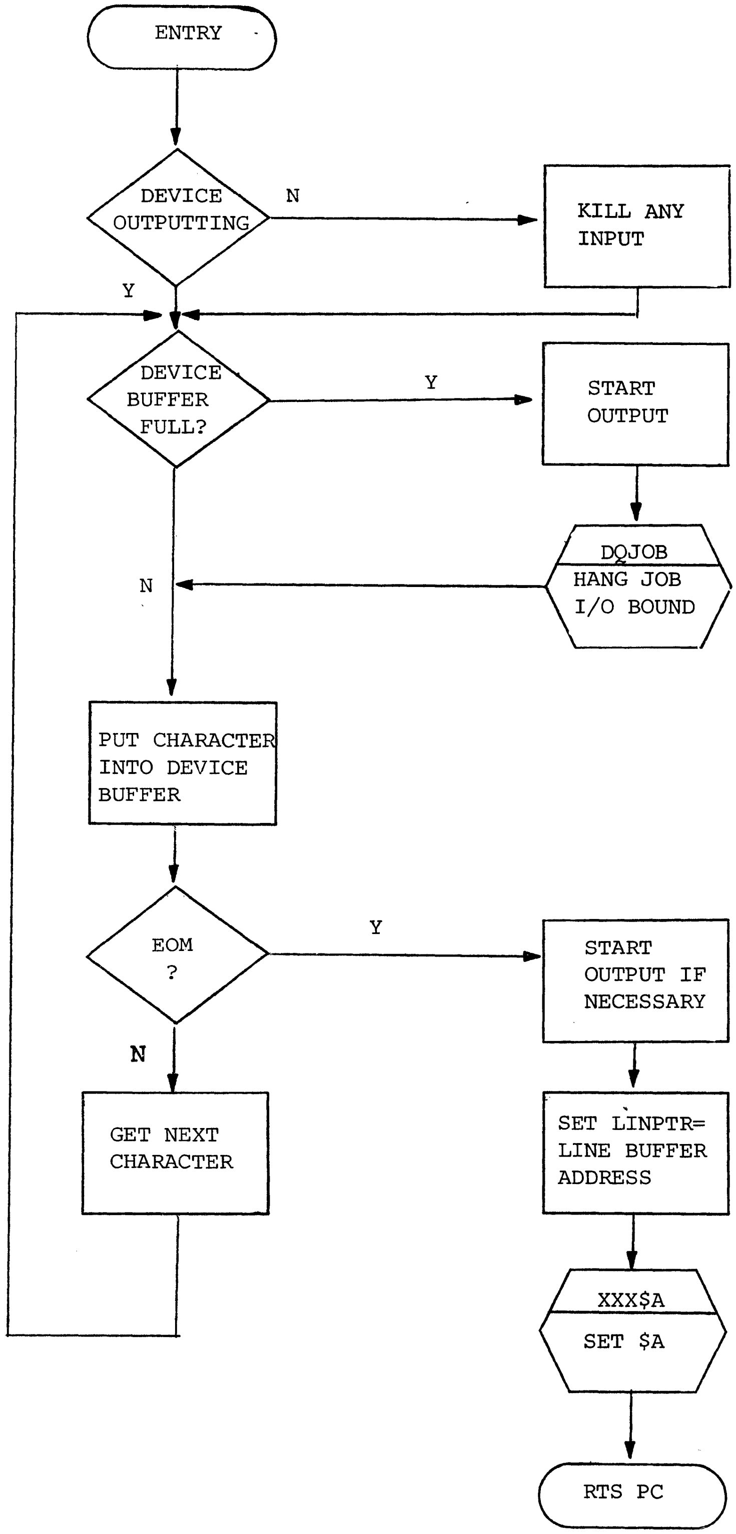
I.2.5.1 Hanging a Job on Output

The driver needs to hang a job on output only when:

- the device is currently output active and, therefore, the output buffer is busy, or
- there are more characters to go out than there is room in the output buffer (for example, the terminal driver hangs the job if there are more than 64 characters in the output message).

The driver may, therefore, submerge a significant part of the output. Whenever the message length exceeds the buffer length or previous output is unfinished, however, the job must hang to allow other MUMPS jobs to run by using the routine DQJOB, as described in paragraph 5.8.4.1.

I.2.5.2 Sample Output Routine Flowchart



I.2.6 Error Reporting--Routine Name: XXX$A

The driver must set the $A System Variable to the current error status at three different times:

- when MUMPS calls XXX$A when the device is ASSIGNED
- when the job is swapped in and the device is still current (device number in $I)
- when the driver finishes I/O processing in the XXXIN and XXXOUT routines

$A may contain two different kinds of information:

- error status
- byte or block address

The terminals and magtape use $A only as an error status register. DECtape uses $A to contain the current byte address. The sequential disk processor uses $A both as an error status register and to contain the current address, through not at the same time.

$A is a 2-word system variable which is treated as a 2-word numeric value. $A is a global tag; the variable itself is in the system module IOD.

$A -- high-order word
$A+2 -- low-order word

Since $A is a numeric value, it must fall within the range of legal MUMPS numbers, when multiplied by 100 (±21474836.47). The programmer should, therefore, take some care when putting information into the high-order word of $A. For example, it is not possible to put a device status register which uses bit 15 as an error bit in the word $A; it must go in $A+2 or be split between the two words.

On entry to the XXX$A routine from the system, the device number will be in the low-order byte of R1. Return is through RTS PC, and no return conditions are expected.

I.3 INTERRUPT SERVICE ROUTINES

When the system receives a device interrupt, it calls the driver's I/O interrupt service routine. What each routine must do is entirely dependent on the device. Two general considerations for both the input and output processing routines are: error checking and deciding when to inform the system that I/O is complete.

When the system calls the routine, it accesses the driver through the device's interrupt vector. The driver does not know the system status; in particular, it does not know whether or not the job owning the device is in the system's Run Queue.

It is suggested that the processor priority level be raised to 6 on any interrupts by setting the second word of the device vector to the appropriate psw value (e.g., 300_8). The Interrupt Service Routine should be brief because interrupts from other devices are shut out during interrupt processing.

The driver must first save the registers using the global routine SAVREG. The driver must restore the registers immediately before the exit RTI by using the global routine RESREG. The calling sequence is the same for both routines:

```
JSR R0,SAVING
    -and-
JSR R0,RESREG
```

I.3.1 Error Reporting

The driver reports error conditions in the $A System Variable. Interrupt processing routine, however, cannot put the error codes directly into $A because the requesting job may not be in the Run Queue. The routine should save the error code in a location unique to the device, for example, each terminal has a "Device Descriptor Buffer" (DDB) to carry this information. The XXX$A routine then puts the error code in $A when the job is waked. In general, an error should terminate any input or output and wake the job although the driver performs more sophisticated error-handling procedures, such as performing retries, etc., if the nature of the device warrants them.

I.3.2 Waking the Job

When I/O completes or an error occurs, the driver's interrupt service routine must request that the job be placed in the system's Run Queue. This operation is the complement of hanging a job in Interpreter-called routines.

The interrupt service routine calls the global routine QJOB to wake a job. QJOB expects the job's number in the low byte of R5 and the hang code (as specified to DQJOB) in the high byte of R5.

Given that the driver knows the device number, it may obtain the job number from the system table DEVTAB. The following example wakes the job owning the interrupting device, and assumes that R5 contains the device number.

```
          .GLOBL DEVTAB,QJOB
          MOVB DEVTAB (R5),R5        ;put job number in R5 (note
                                      that the high byte of R5 is
                                      always 0 because the job
                                      number is positive)
          BIS #HNGCOD,R5             ;put hang code in high byte
          JSR PC,QJOB                ;wake the job
```

There is no I/O hung code when a job is hung on a timed READ. The following example passes the correct information to QJOB when waking a job from a timed READ.

```
          .GLOBL DEVTAB,JOBTAB,QJOB
          MOVB DEVTAB(R5),R5         ;put the job number in R5
          TST JOBTAB(R5)             ;bit 15 is 0 if in timing state
          BPL WAKJOB
          BIS #HNGCOD,R5             ;put hang code in high byte
WAKJOB:   JSR PC,QJOB                ;wake the job
```

If the driver calls QJOB and the job is not hung, the system ignores the call. The interrupt processing routines may, therefore, wake the job at any time.

I.4 DRIVER INTERFACING

A driver is connected to the system software in three ways:

1) The RESASG and RSUASG routines are globally linked to the system's I/O driver module at system build time.

2) The XXXIN, XXXOUT, and XXX$A routines are connected to the system in the RESDV table in the IOD module, in one of two ways:

   a) automatically by 'once only' initialization code in the driver itself probably executed upon the first ASSIGN after system startup.

   b) manually by using Mini-ODT to patch memory after system generation.

3) The interrupt service routines are connected to appropriate interrupt vectors in core memory:

   a) automatically by absolutely defining the contents of the vector before assembly via the .ASECT pseudo-op, or

   b) manually by using Mini-ODT to patch the vector after system generation.

Although there are four reserved device numbers, 51-54, the system provides direct interfacing for only one driver. If more than one driver is necessary, the user must write dispatch routines to match the device number with the correct driver. This applies to all routines for each driver except the interrupt routines.

The device number specified in the ASSIGN command is either in R5 or in $I(R4) in the partition overhead. XXXIN, XXXOUT and XXX$A get the device number from $I. RESASG AND RSUASG get the device number from the low byte of R5.

For example, assume that a card reader has been assigned to device number 51 and a cassette to 52. User-written drivers for both have been linked into the system. The following code performs the correct dispatch from RESASG. CDASG is the ASSIGN routine for the card reader and is physically located in the card reader driver. CSASG is the ASSIGN routine for the cassette driver, and is located in that driver.

```
         .GLOBAL CDASG,RESASG
RESASG:  CMPB #51.,R5             ;compare card reader device number
                                   in R5
         BNE CSASG                ;if not equal, go to cassette
                                   driver routine
         JMP CDASG                ;otherwise, go to card reader routine
CSASG:   .                        ;ASSIGN routine for the cassette
         .
         .
```

The following example shows a dispatch for card reader and cassette input routines. The general input routine was linked into the system as RESIN. The card reader input routine is CDIN,; the cassette input routine is CSIN.

```
        .GLOBL CDIN,$I
RESIN:  CMPB #51.,$I(R4)
        BNE CSIN
        JMP CDIN
CSIN:   .
        .
        .
```

INDEX

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. Problems with software should be reported on a Software Performance Report (SPR) form. If you require a written reply and are eligible to receive one under SPR service, submit your comments on an 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 ________

Please cut along this line.

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 Documentation
146 Main Street ML5-5/E39
Maynard, Massachusetts 01754

digital

digital equipment corporation

Printed in U.S.A.