# PDP-10
# TIME-SHARING MONITORS:
# MULTIPROGRAMMING DISK MONITOR (10/40)
# MULTIPROGRAMMING NONDISK MONITOR (10/40)
# SWAPPING MONITOR (10/50)
# PROGRAMMER'S REFERENCE MANUAL

For additional copies order No. DEC-T9-MTZA-D from Program Library,
Digital Equipment Corporation, Maynard, Massachusetts Price $3.00

DIGITAL EQUIPMENT CORPORATION • MAYNARD, MASSACHUSETTS

Original Printing April 1967
Reprinted July 1967
Revised November 1967
Reprinted March 1968
Revised May 1968
Revised October 1968
Revised August 1969



Instruction times, operating speeds and the like are included in this manual for reference only; they are not to be taken as specifications.

CONTENTS


| | | Page |
|---|---|---|
| CHAPTER 1 | INTRODUCTION | |
| 1.1 | Monitor Functions | 1-1 |
| 1.2 | User Facilities | 1-2 |
| 1.3 | Operating Technique | 1-2 |
| CHAPTER 2 | MONITOR COMMANDS | |
| 2.1 | Console Control | 2-1 |
| 2.2 | Command Interpreter and Command Format | 2-1 |
| 2.2.1 | Command Names | 2-1 |
| 2.2.2 | Arguments | 2-1 |
| 2.2.3 | Login Check (10/50 Monitor) | 2-2 |
| 2.2.4 | Job Number Check (10/40 Monitor) | 2-4 |
| 2.2.5 | Core Storage Check | 2-4 |
| 2.2.6 | Delayed Command Execution | 2-4 |
| 2.2.7 | Completion-of-Command Signal | 2-4 |
| 2.2.8 | Program Searching | 2-4 |
| 2.3 | System Access Control Commands (10/50 Monitor System Only) | 2-5 |
| 2.3.1 | LOGIN Command (Swapping or Multiprogramming Disk Monitor) | 2-5 |
| 2.3.2 | SYSTAT Command (10/50 Monitor) | 2-5 |
| 2.4 | Facility Allocation Commands | 2-6 |
| 2.4.1 | Device Descriptors | 2-6 |
| 2.4.2 | ASSIGN dev ldev° | 2-6 |
| 2.4.3 | DEASSIGN ldev° | 2-7 |
| 2.4.4 | REASSIGN dev job | 2-8 |
| 2.4.5 | FINISH ldev | 2-8 |
| 2.4.6 | TALK tty | 2-8 |
| 2.4.7 | CORE core° | 2-8 |
| 2.4.8 | RESOURCES | 2-9 |
| 2.5 | Run Control Commands | 2-9 |
| 2.5.1 | File Descriptors | 2-9 |
| 2.5.2 | RUN ldev file ext° [p,p]° core° | 2-10 |
| 2.5.3 | R file.ext° core° | 2-10 |

CONTENTS (Cont)


| | | Page |
|---|---|---|
| 2.5.4 | GET, START, HALT (↑C), and CONT Commands | 2-10 |
| 2.5.5 | DDT, REENTER, E and D | 2-11 |
| 2.5.6 | SAVE ldev file ext° core° | 2-12 |
| 2.6 | Background Job Control | 2-13 |
| 2.6.1 | PJOB | 2-13 |
| 2.6.2 | CSTART and CCONT | 2-13 |
| 2.6.3 | DETACH | 2-14 |
| 2.6.4 | ATTACH job [p,p] | 2-14 |
| 2.7 | Job Termination | 2-15 |
| 2.7.1 | KJOB | 2-15 |
| 2.8 | System Timing | 2-15 |
| 2.8.1 | DAYTIME | 2-15 |
| 2.8.2 | TIME job° | 2-15 |
| 2.9 | Comment Entries (;) | 2-16 |

CHAPTER 3 LOADING USER PROGRAMS

| | | |
|---|---|---|
| 3.1 | Memory Protection and Relocation | 3-1 |
| 3.2 | User's Core Storage | 3-1 |
| 3.2.1 | Job Data Area | 3-1 |
| 3.2.2 | Loading Relocatable Binary Files | 3-4 |

CHAPTER 4 USER PROGRAMMING

| | | |
|---|---|---|
| 4.1 | User Mode | 4-1 |
| 4.2 | Programmed Operators (UUO's) | 4-1 |
| 4.2.1 | Operation Codes 001-037 (User UUO's) | 4-2 |
| 4.2.2 | Operation Codes 040-077, and 0 (Monitor UUO's) | 4-2 |
| 4.2.3 | Operation Codes 100-127 (Unimplemented Op Codes) | 4-2 |
| 4.2.4 | Illegal Operation Codes | 4-2 |
| 4.3 | Program Control | 4-3 |
| 4.3.1 | Starting | 4-3 |
| 4.3.2 | Stopping | 4-3 |
| 4.3.3 | Trapping | 4-7 |

CONTENTS (Cont)


| | | Page |
|---|---|---|
| 4.3.4 | Timing Control | 4-7 |
| 4.3.5 | Identification | 4-8 |
| 4.3.6 | Direct User I/O | 4-12 |
| 4.4 | Input/Output Programming | 4-13 |
| 4.4.1 | File | 4-13 |
| 4.4.2 | Initialization | 4-19 |
| 4.4.3 | Data Transmission | 4-27 |
| 4.4.4 | Status Checking and Setting | 4-31 |
| 4.4.5 | Terminating A File (CLOSE) | 4-31 |
| 4.4.6 | Synchronization of Buffered I/O (CALL D, [SIXBIT/WAIT]) | 4-32 |
| 4.4.7 | Relinquishing A Device (RELEASE) | 4-32 |
| 4.5 | Core Control (CALL AC, [SIXBIT/CORE/]) | 4-33 |
| CHAPTER 5 | DEVICE DEPENDENT FUNCTIONS | 5-1 |
| 5.1 | Teletype | 5-2 |
| 5.1.1 | Data Modes | 5-3 |
| 5.1.2 | DDT Submode | 5-5 |
| 5.1.3 | Special Programmed Operator Service | 5-6 |
| 5.1.4 | Special Status Bits (Full Duplex Software only) | 5-9 |
| 5.1.5 | Paper Tape Input from the Teletype (Full Duplex Software only) | 5-9 |
| 5.2 | Paper Tape Reader | 5-9 |
| 5.2.1 | Data Modes (Input Only) | 5-9 |
| 5.3 | Paper Tape Punch | 5-10 |
| 5.3.1 | Data Modes | 5-10 |
| 5.3.2 | Special Programmed Operator Service | 5-11 |
| 5.4 | Line Printer | 5-11 |
| 5.4.1 | Data Modes | 5-11 |
| 5.4.2 | Special Programmed Operator Service | 5-11 |
| 5.5 | Card Reader | 5-12 |
| 5.5.1 | Data Modes | 5-12 |
| 5.6 | Card Punch | 5-13 |
| 5.6.1 | Data Modes | 5-13 |

CONTENTS (Cont)


| | | Page |
|---|---|---|
| 5.6.2 | Special Programmed Operator Service | 5-14 |
| 5.7 | DECtape | 5-14 |
| 5.7.1 | Data Modes | 5-14 |
| 5.7.2 | DECtape Block Format | 5-15 |
| 5.7.3 | DECtape Directory Format | 5-16 |
| 5.7.4 | DECtape File Format | 5-16 |
| 5.7.5 | Special Programmed Operators Service | 5-17 |
| 5.7.6 | Special Status Bits | 5-19 |
| 5.7.7 | Important Considerations | 5-19 |
| 5.8 | Magnetic Tape | 5-20 |
| 5.8.1 | Data Modes | 5-20 |
| 5.8.2 | Magnetic Tape Format | 5-21 |
| 5.8.3 | Special Programmed Operator Service | 5-21 |
| 5.8.4 | 9-Channel Magtape | 5-23 |
| 5.8.5 | Special Status Bits | 5-25 |
| 5.9 | Disk | 5-25 |
| 5.9.1 | Data Modes | 5-25 |
| 5.9.2 | Structure of Files on Disk | 5-26 |
| 5.9.3 | User Programming for the Disk | 5-30 |
| 5.10 | Incremental Plotter | 5-34 |
| 5.10.1 | Character Decoding | 5-34 |
| 5.10.2 | Data Modes | 5-34 |
| 5.10.3 | Pen-up Commands | 5-35 |
| 5.11 | Display with Light Pen (Type 30 and Type 340) | 5-35 |
| 5.11.1 | Data Words | 5-35 |
| 5.11.2 | Background | 5-35 |
| 5.11.3 | Display UUO's | 5-35 |
| 5.12 | CALL AC, [SIXBIT/DEVCHR/] or CALLI AC, 4 | 5-37 |

APPENDIX 1

APPENDIX 2

CONTENTS (Cont)

Page

APPENDIX 3

ILLUSTRATIONS

| | | |
|---|---|---|
| 2-1 | Console Teletype Modes | 2-2 |
| 3-1 | User's Core Area | 3-2 |
| 3-2 | Loading User Core Area | 3-4 |
| 4-1 | User's Ring of Buffers | 4-17 |
| 4-2 | Detailed Diagram of Individual Buffer | 4-17 |
| 4-3 | File Protection Key | 4-25 |

TABLES

| | | |
|---|---|---|
| 2-1 | Monitor Commands | 2-3 |
| 3-1 | Job Data Area Locations | 3-2 |
| 4-1 | Monitor Operation Codes | 4-4 |
| 4-2 | CALL and CALLI Monitor Operations | 4-5 |
| 4-3 | Data Modes | 4-14 |
| 4-4 | File Status | 4-18 |
| 5-1 | Device Summary | 5-1 |
| 5-2 | PDP-10 Card Codes | 5-13 |
| 5-3 | DECtape Programmed Operators | 5-17 |
| 5-4 | MTAPE Functions | 5-22 |
| 5-5 | Magnetic Tape Special Status Bits | 5-25 |

# FOREWORD

This manual covers the use of the Time Sharing Monitors, which include the Multiprogramming non-disk Monitor and the Multiprogramming disk Monitor (formerly known as 10/40) and the Swapping Monitor (formerly known as 10/50).

The Single-User Monitor (formerly known as 10/20, 10/30) is covered in the manual Single User Monitor Systems.

The two most recent software releases, the Concise Command Language for the Monitor and the 4-Series Re-Entrant User Capability, have been included as Addendum I and Addendum II in the back of this manual. The addenda are written from the point of view of someone already familar with the body of the manual. It is suggested that the user read the addenda after becoming acquainted with the body of the manual. This material will be incorporated into the next revised edition of this manual.

# CHAPTER 1
# INTRODUCTION

This manual covers commands, program loading and programming of the PDP-10 Time-Sharing Monitors -- three multiprogramming, time-sharing systems designed to allow many independent user programs to share the facilities of the computing system. Such users can access the computer at the same time from consoles located at the computer site, at nearby offices and laboratories, or even at remote consoles connected by telephone lines.

Operating concurrently under Monitor control, these diverse users may access available I/O device and system software to compile, assemble, and execute their programs, or perform this sequence automatically for many jobs by using the batch control processor (Batch). Real-time jobs can operate either as independent user programs or as fully integrated Monitor subroutines.

The Multiprogramming non-disk (10/40) Monitor is a multiprogramming, time-sharing system which includes an I/O controller, run-time selection of I/O devices, job-to-job transition, job save and restore features, and memory dump facilities. All of these features are incorporated with concurrent realtime processing, batch processing, and time sharing. The Multiprogramming disk (10/40) Monitor adds a comprehensive file system with both sequential and random access of shared, named files to the Multiprogramming non-disk system. The Swapping (10/50) Monitor incorporates all of the features of the Multiprogramming disk system and, in addition, swaps programs between high speed disk and core.

## 1.1 MONITOR FUNCTIONS

All Monitors schedule multiple-user time sharing of the system, allocate available facilities to user programs, accept input from and direct output to all system I/O devices, and relocate and protect user programs in core memory.

The Monitors utilize the PDP-10 hardware features of memory protection, memory relocation, executive/user mode, and real-time clock to provide an advanced, third-generation, multiprogramming, time-sharing environment. System facilities start with a minimum configuration of 16K core and two DECtapes, and can accommodate magnetic tapes, disks, communication line controllers, card readers, paper tape readers and punches, line printers, displays, plotters, and user Teletype consoles. Other special devices, including real-time digitizers and analog converters, easily interface with the system.

Several programs are loaded into core. The Monitors allow each program to run for a certain length of time, based on a scheduling algorithm which permits the most efficient use of system facilities. The Monitors process input/output commands from the programs, making them device independent, and perform I/O operations concurrently with computation for high system efficiency.

## 1.2 USER FACILITIES

Users gain access to the system from a console at the facility or remotely located at any point with telephone facilities. Three levels of communication are available at the consoles. Initially, the console communicates with the Monitor Command Interpreter, which provides the system with access protection (LOGIN); allocates and protects memory (CORE) and peripherals (ASSIGN, REASSIGN) requested by the user; provides communication to the operator (TALK) for mounting of special tapes; provides the user with run control (RUN, GET, START, HALT, CONT) over programs stored in the system; allows the user to initiate background jobs (CSTART, CCONT, DETACH, ATTACH); provides the user with job monitoring and debugging (E, D, DDT, REENTER) facilities, and returns facilities to the system (KJOB, DEASSIGN) when the job is finished.

With this set of Monitor commands, the user at his console has access to the system file, which contains programs such as TECO, EDITOR and PIP, for creating and editing program source files, assembling or compiling (MACRO, FORTRAN) program source files, and loading relocatable binary files. The core image of a loaded relocatable binary file may be stored on a retrievable storage device (SAVE) and thereafter be available through the Monitor Command Interpreter. Many other programs are available in the system file to facilitate file management and translation.

## 1.3 OPERATING TECHNIQUE

When a user starts a program, his console serves as an input/output device, which provides a control and data path to his private program. The console is switched back to the Monitor Command Interpreter by either the program (HALT, EXIT) or by the user striking both the CTRL and C keys ( ↑ C) at the console. The user can exercise another dimension of control over his program by loading it with the powerful Dynamic Debugging Technique (DDT) available in the system file. Entry to DDT is through the Monitor Command Interpreter or by break points from the program. While program control is in DDT, the console permits examining intermediate results and modifying the program (symbolically).

The user's program communicates with the Monitors by means of the PDP-10 operation codes 040 through 077. With these calls, the Monitors provide the program with complete device-independent input/output services, which relieves the programmer of the arduous task of I/O programming, as well as freeing him from dependence on the availability of particular devices at run time. In addition, the user's program may exercise control over central processor trapping (overflow, underflow, pushdown overflow, clock), modify its memory allocation (CORE), and monitor its own running time. Provision exists for inter-job communication and control, reentrant user programs, and, in selected cases, direct user I/O control.

# CHAPTER 2
# MONITOR COMMANDS

## 2.1 CONSOLE CONTROL

From the user's point of view, his time-sharing console is in one of three modes: the Monitor mode, the user mode, or the detached mode. In the Monitor mode, characters typed in are presented to the Monitor Command Interpreter. In the user mode, the console acts as an ordinary input/output device under control of the user's program (the DDT submode, a special user mode, is used when running under control of the Dynamic Debugging Technique program). The console is in the detached mode if nothing has been typed on it since the Monitor was started or if the DETACH command is typed. The ATTACH command places it back in Monitor mode.

If the console is in the detached mode and a character is typed in, the console either enters the Monitor mode or immediately responds with "X" or "JOB CAPACITY EXCEEDED," both indicating that the system is at maximum job capacity. It remains in the detached mode. Once in the Monitor mode, each line of text typed in is sent to the Monitor Command Interpreter for processing. If the command is not understood by the Monitor Command Interpreter, an error message is typed out and the console mode is unchanged. Figure 2-1 indicates the console mode at the successful completion of each command.

## 2.2 COMMAND INTERPRETER AND COMMAND FORMAT

Table 2-1 lists the commands and their characteristics. Each command is a line of ASCII characters in upper and/or lower case. Spaces and nonprinting characters preceding the command name are ignored. The Monitor Command Interpreter ignores a line preceded by a semicolon.

### 2.2.1 Command Names

Command names are strings of from one to six letters. Characters after the sixth are ignored. Only enough characters to uniquely identify the command need be typed.

### 2.2.2 Arguments

Arguments follow the command name, separated from it by a space or any printing character that is not a letter or a numeral. Argument formats are described under the associated commands.

If the Monitor Command Interpreter recognizes the command name, but a necessary argument is missing, the Monitor responds with

TOO FEW ARGUMENTS

Extra arguments are ignored.

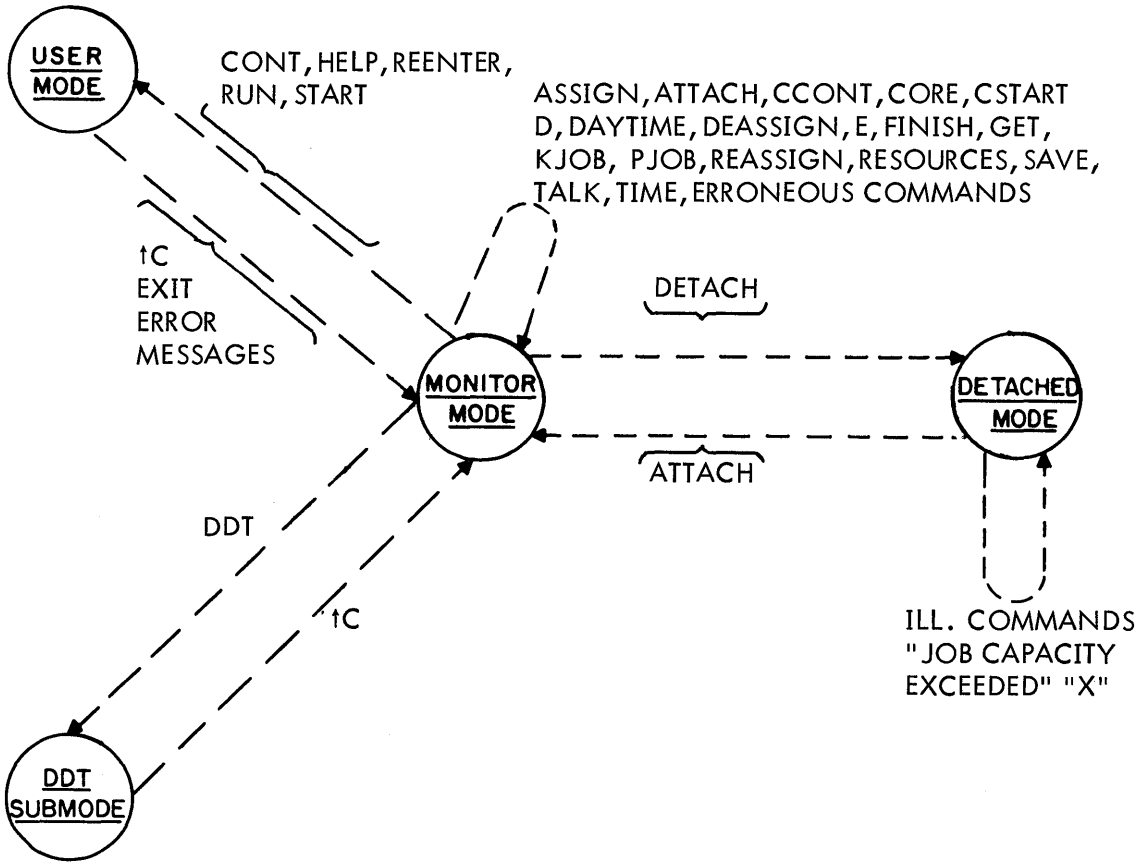



Figure 2-1 Console Teletype Modes

## 2.2.3 Login Check (10/50 Monitor)

If a user who has not logged in (see "LOGIN Command") types a command requiring the user to be logged in, the 10/50 Monitor responds with

LOGIN PLEASE

The user's command is not executed. Login is not required by the 10/40 Monitor.

Table 2-1

Monitor Commands (See Addendum I and II)

| Name | Abbre-viation | Arguments 1 | 2 | 3 | 4 | 5 | Console Mode | Characteristics |
|---|---|---|---|---|---|---|---|---|
| ASSIGN | AS | dev | ldev° | | | | m | L,J |
| ATTACH | AT | job | [p,p] | | | | m | |
| ATTACH | AT | dev | | | | | m | L |
| CCONT | CC | | | | | | m | L,J,C,I |
| CONT | CON | | | | | | u | L,J,C,I |
| ↑C | - | | | | | | m | |
| CORE | COR | core° | | | | | m | J,A,I |
| CSTART | CS | addr° | | | | | m | L,J,C,I |
| D | D | lh | rh | addr | | | m | L,J,C,I |
| DAYTIME | DA | | | | | | m | |
| DDT | DD | | | | | | u (DDT) | L,J,C,I |
| DEASSIGN | DEA | ldev° | | | | | m | L |
| DETACH | DET | dev° | | | | | d | L |
| E | E | addr° | | | | | m | L,J,C,I |
| FINISH | F | ldev | | | | | m | L,J,C,A,I |
| GET | G | ldev | file | ext° | [p,p]° | core° | m | L,J,A |
| HA | H | | | | | | m | |
| KJOB | K | | | | | | m | A |
| LOG | L | | | | | | u | I |
| PJOB | P | | | | | | m | L,J |
| R | R | file | ext° | core° | | | u | L |
| REASSIGN | REA | ldev | job | | | | m | L,J,I |
| REENTER | REE | | | | | | u | L,J,C,I |
| RESOURCES | RES | | | | | | m | L |
| RUN | RU | ldev | file | ext° | [p,p]° | core° | u | L |
| SAVE | SA | ldev | file | ext° | [p,p]° | core° | m | L,J,C,A,I |
| START | ST | addr° | | | | | u | L,J,C,A,I |
| SYSTAT | SYS | | | | | | u | |
| TALK | TA | tty | | | | | m | |
| TIME | TI | job° | | | | | m | |

| | | | |
|---|---|---|---|
| ° | optional argument | [p,p] | [project number, programmer number] (see "LOGIN Command") |
| addr | octal address | | |
| core | decimal number of 1K blocks | tty | CTY,OPR,TTY0,...,TTYn |
| dev | CDR,CTY,DIS,DSK,DTA0,...,DTA7 LPT,MTA0,...,MTA7,OPR,PTP,PTR PTY0,...PTYn,SYS,TTY0,...,TTYn | d | detached |
| | | m | Monitor |
| | | u | user |
| ldev | dev or a logical device name. | L | LOGIN required (10/50 Monitor) |
| ext | filename extension, 1 to 3 characters, must be preceded by a point (.) | A | no active devices |
| | | C | core required |
| file | filename, 6 characters or less | I | must be in core |
| job | job number assigned by Monitor | J | requires job number (10/40 Monitor) |
| lh rh | octal value of left and right half words. | | |

### 2.2.4 Job Number Check (10/40 Monitor)

If the 10/40 Monitor recognizes a command name which requires a job number and no job number is assigned, the Monitor assigns a job number, n, and responds with,

JOB n

together with a line identifying the Monitor version, and proceeds to execute the command.

### 2.2.5 Core Storage Check

If the Monitor Command Interpreter recognizes a command name which requires core storage to have been allocated to the job and the job has no core, the Monitor responds with

NO CORE ASSIGNED

The user's command is not executed.

### 2.2.6 Delayed Command Execution

If the Monitor Command Interpreter recognizes the command name and the job has devices actively transmitting data to or from its core area and the command requires that all devices be inactive, or if the job is swapped out to the disk and the command requires core residence, the Monitor delays execution of the command until the devices are inactive or the job is in core. If another command is typed while a command is waiting, the first command is ignored.

### 2.2.7 Completion-of-Command Signal

Most commands are processed instantly. The completion of each command is signaled by the output of a carriage return, line feed. If the console is left in Monitor mode, a period follows the carriage return, line feed. If the console is left in user mode, any response other than the carriage return, line feed must come from the user's program.

### 2.2.8 Program Searching

If the Monitor Command Interpreter does not recognize the command name, the Monitor assumes that it is the name of a program in the system file. If the Monitor cannot find the program in the system file, it responds with the name, followed by

NOT FOUND

If the program is found, the Monitor loads the program into core and starts it with the console in user mode.

## 2.3 SYSTEM ACCESS CONTROL COMMANDS (10/50 Monitor System Only)

Access to the system is limited to authorized personnel. The system administrator provides each user with a project number, a programmer number, and a password. The project and programmer numbers are octal numbers up to nine digits each. The password is a sequence of from one to five ASCII characters, which must match the password stored in the system accounting file to LOGIN successfully.

### 2.3.1 LOGIN Command (Swapping or Multiprogramming Disk Monitor)

LOGIN waits for the user to type in the project and programmer numbers on a line, separated by a comma (,) and terminated by a carriage return. LOGIN then responds with the word PASSWORD: and turns off the Teletype printer. The user types in his password which is not printed on the paper. If the typed-in project-programmer number and password match a project-programmer number and password stored in the system accounting file (ACCT. SYS [1,1]), LOGIN admits the user to the system and responds with the time of day, date, Teletype number, and message of the day, if any (file NOTICE. TXT [1,1]), and finally, ↑C and a period, leaving the console in Monitor mode ready to accept commands. If the password does not match or the project-programmer number does not exist, LOGIN responds with the error message

"?INVALID ENTRY-TRY AGAIN"

and waits for the project-programmer number, password combination to be typed again.

Example:

| | |
|---|---|
| .LOGIN ↵ | User issues LOGIN command. |
| JOB n | Monitor responds with job number assigned, followed by Monitor name and version number. |
| # | System types out a number sign to indicate user should type his project-programmer number. |
| proj,prog ↵ | User types in his project-programmer number (each number can contain up to nine octal digits). |
| PASSWORD: | System requests user to type in his password. The password will not be printed on the paper. |
| 1045 01-AUG-69 TTY7<br>↑C<br>. | If the user entries are correct, the Monitor responds with the time of day, date, Teletype number, message of the day (if any), Control C, and a period. |

### 2.3.2 SYSTAT Command (10/50 Monitor)

SYSTAT prints a summary of the current system status on the user's console.

## 2.4 FACILITY ALLOCATION COMMANDS

One of the functions of the Monitor is to allocate peripheral devices and core memory to users upon request, and to protect allocated facilities from interference by other users. To this end, the Monitor maintains a pool of available facilities from which a user can draw and restore by request.

A user should never abandon a time-sharing console without returning allocated facilities to the pool.

### 2.4.1 Device Descriptors

The devices controllable by the system are listed in Table 5-1. Associated with each device is a physical name, made up of three letters and zero to three numerals to specify unit (transport) number. All references to devices in the Monitor are made by these physical names or by assigned logical names.

### 2.4.2 ASSIGN dev ldev°

ASSIGN has one required argument, dev (device), and one optional argument, ldev. Dev must be a physical device name, or DTA or MTA. If dev is DTA or MTA, the Monitor searches the device pool for a free unit. Monitor responses are:

| | |
|---|---|
| DEVICE dev ASSIGNED | (physical device dev was free and has been assigned to the user) |
| NO SUCH DEVICE | (all units are in use) |
| ALREADY ASSIGNED TO JOB n | (dev is allocated to another job, n) |

2.4.2.1 Logical Device Names (ldev) - The second argument, ldev, is optional. It represents a logical device name of one to six alphanumeric characters of the user's choice, usable synonymously with dev in all references to the device. Logical device names take precedence over physical device names. Thus, a user may write programs to use arbitrarily named devices which he assigns to the most convenient physical devices at run time.

If the user has the name ldev assigned to another device, the Monitor responds with

LOGICAL NAME ALREADY IN USE
DEVICE dev ASSIGNED

2.4.2.2 Examples

| | | |
|---|---|---|
| User types | ASSIGN DTA, ABC | |
| Monitor responds | DEVICE DTA6 ASSIGNED | (successful) |

| | | |
|---|---|---|
| User then types | ASSIGN DTA,DEF | (find another unit) |
| Monitor responds | NO SUCH DEVICE | (all in use) |
| User then types | ASSIGN PTP, ABC | (reserve paper tape punch) |
| Monitor responds | LOGICAL NAME ALREADY IN USE<br>DEVICE PTP ASSIGNED | (paper tape punch is reserved, but ABC still refers to DTA6 only) |
| User then types | ASSIGN DTA1, DEF | |
| Monitor responds | ALREADY ASSIGNED TO JOB 2 | (another user has it) |

2.4.2.3 ASSIGN SYS: dev - This command is used to change the systems device (SYS:) from its currently allocated device to some other device (dev). In order to issue this command, the user must be logged in under either [1,1] or [1,2].

2.4.2.4 Device Protection - When a device is assigned to a job, it is removed from the Monitor's pool of available devices. Any attempt by another job to reference the device fails. The device is returned to the pool when the user deassigns it or kills the job.

2.4.2.5 Special Functions - The ASSIGN command applied to DECtapes clears the copy of the directory currently in core, forcing any directory references to read a new copy from the tape. This is especially important when changing reels. (See Chapter 5 for further details.)

## 2.4.3 DEASSIGN ldev°

This command cancels device reservations made via the ASSIGN command and returns the device(s) to the Monitor pool. The command may be typed alone or with one argument, ldev. When an argument is typed, it must be the logical or physical name of some device previously reserved by the ASSIGN command. If no argument is typed, all devices currently reserved by the user via the ASSIGN command are affected. The DEASSIGN command may be typed, even though the user's program continues to use the devices affected.

Monitor error responses are:

NO SUCH DEVICE

DEVICE WASN'T ASSIGNED

2.4.3.1 Special Functions - The DEASSIGN command applied to DECtapes performs the same special function as ASSIGN, section 2.4.2.5.

### 2.4.4 REASSIGN dev job

REASSIGN allows one job to pass a device to a second job without going through the Monitor pool. Two arguments are required: the physical device name, dev, and the job number of the second job. Dev is deassigned from the current job and assigned to the second job. All devices except user consoles can be reassigned.

Monitor error responses are:

DEVICE dev WASN'T ASSIGNED
JOB NEVER WAS INITIATED
NO SUCH DEVICE
DEVICE CAN'T BE REASSIGNED

### 2.4.5 FINISH ldev

FINISH terminates any input or output currently in progress on device ldev and relinquishes it (see RELEASE).

Monitor error response is:

NO SUCH DEVICE

### 2.4.6 TALK tty

The TALK command allows a user to type directly on another user's console, and the latter to type back. If device tty is in the detached mode or in Monitor mode and at the left margin, the user's console is inserted into a talk "ring" with tty. Otherwise the Monitor responds with BUSY. Any number of consoles can be in the same talk ring. Each character typed on any console in the ring is printed on all other consoles in the ring. Any console is removed from the ring by typing ↑C. The required argument, tty, can be any of the physical device names CTY, TTY0,...,TTYn or the special device name OPR.

2.4.6.1 Operator's Console - When the Monitor is started, one console, usually CTY, is designated as the operator's console and given the name OPR. All requests for local operations such as mounting and unmounting tapes, etc., can be performed with TALK OPR.

### 2.4.7 CORE core°

The CORE command has one optional argument, core. Without the argument, the Monitor responds with the decimal number of 1024-word blocks of unallocated core in its pool if 10/40 system and with the maximum size of user's core if 10/50 system. The optional argument, core, is the total

decimal number of 1024-word blocks of core memory allocated to the job upon successful completion of the command. If it is smaller than the current allocation, the difference is removed from the top of the user's core area, and returned to the Monitor pool. If it is larger than the current allocation, the difference, if available, is removed from the pool and appended to the top of the user's core area. In the 10/40 (nonswapping) system, if the difference is not available, the user's current core area is unchanged, and the Monitor responds with the decimal number of 1024-word blocks in the pool. In the 10/50 (swapping) system, if the difference is not available, the user program is swapped out and brought back a short time later when it can fit. The user need not know if his program is swapped out or not.

### 2.4.8 RESOURCES

This command causes the typeout of all available devices (except Teletypes) and the number of free blocks on the disk.

## 2.5 RUN CONTROL COMMANDS

Core image files located on retrievable storage devices such as disk, DECtape, and magnetic tape can be retrieved and controlled from the user's console. The process of creating such files is described in Chapter 3. Files stored on disk and DECtape are addressable by name. Files on magnetic tape require prepositioning the tape by the user.

### 2.5.1 File Descriptors

2.5.1.1 Filenames - Filenames are from one to six letters or digits. All letters and digits after the sixth are ignored. A filename is terminated by any character that is not a letter or digit.

2.5.1.2 Filename Extension - If the filename is terminated by a period, a filename extension is assumed to follow. A filename extension is from one to three letters or digits. It is generally used to indicate file format. The filename extension is terminated by any character not a letter or a digit. If a filename extension is not specified with the RUN, GET, and SAVE commands, an extension of SAV is assumed.

2.5.1.3 Project-Programmer Numbers - If a user wants to perform a RUN or GET command on a disk file belonging to another user, he must specify the user's project-programmer numbers. The format is

[project-number, programmer-number]

2.5.2 RUN ldev file ext° [p,p] ° core°

The RUN command loads a core image from a retrievable storage device (DECtape, disk, and magnetic tape), ldev, and starts it at a location specified within the file (see JOBSA, "Job-Data Area", Chapter 3). The arguments file, ext, and [p,p] are used to select the file. The minimum amount of core required to load the file is allocated. After the file is loaded, core is reallocated if the optional fifth argument, core, is specified or if the file was saved with a core argument. If both were specified, the RUN command core argument takes precedence. The optional argument is ignored if it is less than the size of the file. If ldev is a magnetic tape, the fifth argument must be specified, and be at least as large as the core image file to assure proper loading.

Monitor error responses are:

| | |
|---|---|
| ldev NOT AVAILABLE | (ldev is allocated to another job) |
| NO SUCH DEVICE | (ldev is undefined) |
| nK OF CORE NEEDED | (where n is a decimal number of 1024-word blocks, if there is insufficient free core to load the file or to satisfy the optional core argument on the reallocation) - 10/40 Monitor only. |
| NOT A DUMP FILE | (the selected file is not a core image file) |
| TRANSMISSION ERROR | (a parity or device error occurred during data transmission) |

2.5.3 R file.ext° core° - The R command is equivalent to the command

RUN SYS file.ext° core°

and is provided as a convenience for the user. In other words, the R command is the usual command for running one of the CUSPs (Commonly Used System Programs) in the system library. Note that R is not an abbreviation for RUN; if the program is on a device other than SYS, the user must use the RUN command (abbreviated RU).

2.5.4 GET, START, HALT (↑C), and CONT Commands

The GET, START, HALT, and CONT commands permit the user to control the running of his program from the console.

2.5.4.1 GET ldev file ext° [p,p] ° core° - The GET command is the same as the RUN command, except that the Monitor responds with

JOB SETUP

instead of starting the program. The assignment of core is also similar to that of the RUN command.

2.5.4.2 START addr° - The START command begins execution of the user's program. If the optional argument, addr, is not specified, the starting address is found in the core area (right half of JOBSA as set up by the LOADER from an END statement in the source program, see Chapter 3). The optional argument, addr, is an octal number and, if specified, the program is started at that location. Monitor error responses are:

NO CORE ASSIGNED

NO START ADR (if the content of JOBSA is 0).

The user must supply a starting address on his END statement.

2.5.4.3 HALT and ↑C - Typing a ↑C (hold down the CTRL key and strike "C") on the console puts the console in Monitor mode and transmits a HALT command to the Monitor Command Decoder. The HALT command stops the job and stores the program counter in the job's core area (JOBPC, "Job-Data Area," Chapter 3).

2.5.4.4 CONT - The CONT command starts the program at the location specified by the contents of the saved program counter in the job's core area (JOBPC, see "Job-Data Area," Chapter 3), and puts the console in user mode. If the CONT command is given to a job which was stopped as a result of a Monitor-detected error, the Monitor responds with

CAN'T CONTINUE

The CONT command is applicable only if the job was stopped by the HALT (↑C) command or the HALT instruction.

2.5.5 DDT, REENTER, E and D

The DDT, REENTER, E, and D commands are used primarily for program debugging and exception handling. The DDT and REENTER commands provide alternate program entry points. E and D provide a means of examining and modifying locations in the user's core area from the console.

2.5.5.1 DDT - The DDT command copies the saved value of the user's program counter (JOBPC) into a second location in his core area (JOBOPC, see "Job Data Area," Chapter 3), and starts his program at an alternate entry point specified by another location (JOBDDT, see "Starting Addresses," Chapter 3) in his core area. This alternate entry point is set to the beginning address of DDT by the loader, if the program was loaded with DDT. Alternately, the user may set this address to any desired location. To resume computation following the DDT command interruption, execute a ↑C and START (JRST 2, @ JOBOPC). The Monitor error response is:

NO START ADR (if the content of JOBSA is 0).

The user must supply a starting address on his END statement.

2.5.5.2 REENTER - The REENTER command is similar to the DDT command. The alternate entry point is specified by a different location (JOBREN, see "Job Data Area," Chapter 3) in the job core area, and must be set by the user or his program. The Monitor error response is:

NO START ADR (if the content of JOBSA is 0).

The user must supply a starting address on his END statement.

A typical use of this command is interrupting a long computation to examine intermediate results. The user types ↑C, and then REENTER, which transfers to his routine to print intermediate results. This routine should preserve the state of his main program, and return to the interrupted computation by executing a JRST 2, @ JOBOPC.

2.5.5.3 E addr° - The E command allows the user to examine locations in his core area. If the optional argument, addr, which is an octal number, is specified, the octal contents of the left and right halves of location addr are typed. Leading zeros in the half words are suppressed. The half-word values are separated by a space, and the right half value is followed by a horizontal tab. If the optional argument, addr, is not specified, the contents of the next location are typed. If the location to be examined lies outside the user's allocated core area, the Monitor responds with

OUT OF BOUNDS

2.5.5.4 D lh rh addr° - The D command allows a user to deposit into his core area. The required arguments lh and rh are the octal values of the left and right half words to be deposited. If the optional argument, addr, which is an octal number, is specified, the word is deposited at location addr. If it is not specified, the word is deposited at the location following the last location examined or deposited. If the location is above the user's core area, or in the protected part of the job data area (see Table 3-1) above user AC 17, the Monitor responds with

OUT OF BOUNDS

2.5.6 SAVE ldev file ext° core°

The SAVE command writes a core image file of the user's core area. The Monitor saves space by compressing core and eliminating words of zeroes before writing. It expands core back again after the output operation has completed. If DDT is loaded, i.e., if JOBDDT is nonzero (see Chapter 3), the entire core area, except the user's ACS, is written. Otherwise, the area starting from JOBDDT and extending up through the program break (as specified by the contents of JOBFF, see Chapter 3) is

written. If the optional argument, ext, is not specified, the filename extension is SAV. The optional argument, core, specifies the minimum number of 1024-word blocks in which the program is to be run. This parameter is stored in the job's core area (JOBCOR, see Table 3-1), and is used by the RUN and GET commands. The state of the users accumulators and input/output devices are not saved.

After the output is completed, the Monitor responds with:

JOB SAVED

Monitor error responses are:

| | |
|---|---|
| n 1K BLOCKS OF CORE NEEDED | (where n is the contents of JOBFF modulo 1024, if the user's current core allocation is less than the contents of JOBFF) |
| DEVICE NOT AVAILABLE | (device ldev is allocated to another user) |
| TRANSMISSION ERROR | (an error was detected while writing) |
| DIRECTORY FULL | (the maximum number of files already exists for device ldev) |

## 2.6 BACKGROUND JOB CONTROL

A job is a "background" job if it is not under control of a user console. Any console can initiate any number of background jobs. Input/output to the console while a job is running in background mode causes the job to stop until a console is attached.

### 2.6.1 PJOB

The PJOB command responds with the job number to which the user's console is attached. If the console is not attached to a job, the 10/40 Monitor assigns a job number, and responds with the job number and a line identifying the Monitor version; the 10/50 Monitor responds with the message

LOGIN PLEASE

The job number is a necessary argument for the ATTACH command.

### 2.6.2 CSTART and CCONT

The RUN, START, and CONT commands always leave the user console in user mode. ↑C switches the console to Monitor mode, but also stops the job. The CSTART and CCONT commands are identical to the START and CONT commands, respectively, with the exception that the console is left in Monitor mode.

In general, to start a job running with the console in Monitor mode, it is necessary to begin with the console in user mode; type control information to the program; type ↑C, which stops the job with console in Monitor mode; and, finally, type the CCONT command, which allows the job to con-

tinue running with the console in Monitor mode. Further commands may now be executed while the job is running.

### 2.6.3 DETACH

The DETACH command disconnects the user's console from the job, placing the console in the detached mode without affecting the status of the job. For instance, if the job was running, it remains running in the background mode. The user console is now free to control another job, either by creating a new job or ATTACHing to a background job.

2.6.3.1 DETACH dev - This command causes the assignment of device dev to JOB 0, thus making it unavailable to the system. In order to issue this command, the user must be logged in under [1,1].

### 2.6.4 ATTACH job [p,p]

The ATTACH command allows a user to connect a console to a background job. Two arguments are required. The first argument, job, is the job number of the job to which the user desires to attach. In Multiprogramming disk systems and Swapping systems, the second argument, [p,p], is the project-programmer number pair of the originator of the desired job. Following the ATTACH command, the console is always left in the Monitor mode. If the job happens to be running, typing CONT places the console in the user mode without affecting the operation of the job. It is not necessary to execute the DETACH command before the ATTACH command, in order to switch the console between two jobs, since the current job is automatically DETACHed.

If [p,p] is omitted, the user's project-programmer number will be assumed. This speeds up the usual case when the user has LOGGED in twice under the same project-programmer number. The operator (device OPR) may always attach to a job even though another console is attached, provided he specifies the proper project-programmer number. This gives the operator complete control of the system in case of mishap in a particular job.

If an error condition occurs, the console is left attached to the job to which it was connected before the ATTACH was typed.

Monitor error responses are:

| | |
|---|---|
| TTYn ALREADY ATTACHED | (either the job number typed is erroneous and by coincidence is attached to another console, or another user is attached to the job number specified) |
| NOT A JOB | (the specified job number is not assigned to any job) |
| CAN'T ATTACH TO JOB | (the second argument, [p,p] is not the project-programmer pair of the job originator) |

2.6.4.1 ATTACH dev - This command returns a detached device to the Monitor pool and makes it available to the system. In order to issue this command, the user must be logged in under [1,1].

## 2.7 JOB TERMINATION

When a user leaves the system, all facilities allocated to his jobs must be returned to the Monitor facility pool, thereby making them available to other users.

### 2.7.1 KJOB

The KJOB command performs the following functions on the job to which the console is attached:

a. Stops all allocated input/output devices and returns them to the Monitor pool;

b. Returns all allocated core to the Monitor pool;

c. Returns the job number to the Monitor pool;

d. Performs a TIME command; and

e. Leaves the console in Monitor mode.

## 2.8 SYSTEM TIMING

All system times are kept in increments of one 60th (or 50th) of a second. The DAYTIME and TIME commands print time in the format

hhmm:ss.ss

where hhmm is a 4-digit representation of hours and minutes and ss.ss is seconds to the nearest hundreth.

### 2.8.1 DAYTIME

The DAYTIME command prints the data followed by the time of day.

### 2.8.2 TIME job°

The TIME command prints the incremental running time, i.e., the running time since the last TIME command, followed by the total running time used by the job. Interrupt level and job scheduling times are charged to the job running when the interrupt or rescheduling occurred. If the optional argument, job, is not specified, the job to which the console is attached is used. If the optional argument is zero, the Monitor prints 5 quantities about system utilization as

HH:MM:SS:HH

SHFL - Time spent in BLT shuffling core

ZCOR - Time spent in BLT zeroing core

LOST - Time spent in NULL job when other jobs wanted to run but could not because they were swapped out, on the way in or out; they were stopped, waiting to be shuffled; or they were being swapped because of expanding core.

NULL - Total time in NULL job (including LOST)

UP - Total time since system was loaded

## 2.9 COMMENT ENTRIES (;)

The operator may type a line of comments on the Teletype by preceding the line with a semicolon. This line will not be interpreted or executed by the Monitor.

# CHAPTER 3
# LOADING USER PROGRAMS

## 3.1 MEMORY PROTECTION AND RELOCATION

A user's program runs while the computer is in a special mode known as the user mode. In this mode, the contents of the memory relocation register in the central processor are automatically added to each memory address before the address is sent to the memory system. The address, before this addition takes place, is called the relative address; after the addition, the address is called the absolute address. The contents of the memory protection register are compared with the eight high-order bits of each relative address. If the relative address exceeds the contents of the memory protection register, the memory violation flag is set in the central processor and control traps to the Monitor.

Thus, the contents of the memory protection and relocation registers define a contiguous area of core with the following properties:

a. All memory references from within the region are relative to the beginning of the region.

b. It is impossible to address a location outside the region from within the region.

When the Monitor schedules a user's program to run, it sets the memory protection and relocation registers to the bounds of the user's allocated core area and switches the central processor to the user mode.

In this manual, all addresses in the user's area are relative addresses.

To take advantage of the fast accumulators, memory addresses 0 through 17 are not relocated. Thus, relative locations 0 through 17 cannot be referenced by the user's program. The Monitor saves the user's accumulators in this area when the user's program is not running and while the Monitor is servicing a program call from the user.

## 3.2 USER'S CORE STORAGE

A user's core storage consists of a single contiguous block of memory whose size is an integral number of 1024 words (see Figure 3-1). There are two methods available to the user for loading his core area. The simplest way is to load a core image stored on a retrievable device (see RUN and GET, Chapter 2); the second is to use the relocatable binary loader to link-load binary files. The user may then write the core image on a retrievable device for future usage (see SAVE, Chapter 2).

### 3.2.1 Job Data Area

The first $140_8$ locations of the user's core area comprise the job data area reserved for storing specific information concerning the job, such as the starting address of the user's program (JOBSA),

highest legal address (JOBREL), etc. Locations in this area have been given mnemonic assignments whose first three characters are JOB, e.g., JOBSA, JOBFF, JOBDDT, etc (see Table 3-1). As a consequence all mnemonics in this manual with a JOB prefix refer to locations in the job data area.

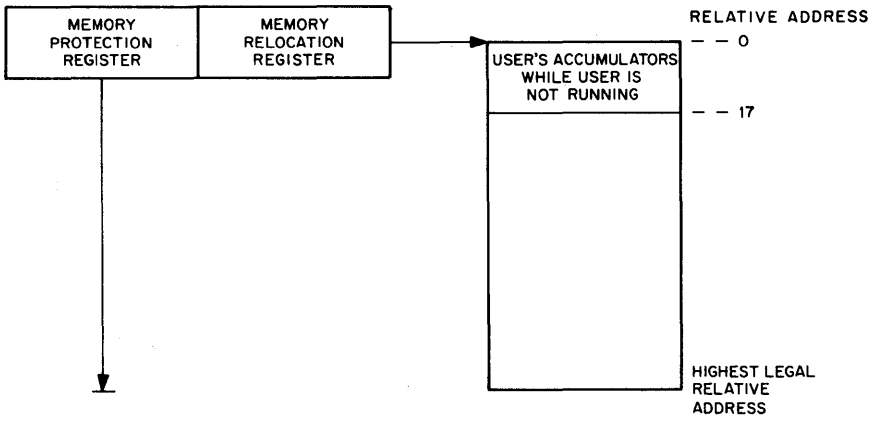



Figure 3-1 User's Core Area

Table 3-1
Job Data Area Locations (See Addendum II)

| Name | Relative Location(s) Octal | Description |
|---|---|---|
| JOBUUO | 40 | User's location $40_8$. Used for processing user UUO's (001 through 037). |
| JOB41 | 41 | User's location $41_8$. Contains the beginning address of the user's programmed operator service routine. |
| JOBREL | 44 | Left half: 0<br>Right half: The highest relative core location available to the user (i.e., the contents of the memory protection register when this user is running). |
| JOBDDT | 74 | Contains the starting address of DDT. If contents are 0, DDT has not been loaded. |
| JOBPFI | 114 | Highest location in the job data area protected from I/O, that is, the Monitor will not perform I/O into or out of locations 0 through JOBPFI. |
| JOBSYM | 116 | Contains a pointer to the symbol table created by Linking Loader.<br>Left half: Negative count of the length of the symbol table.<br>Right half: Lowest register used. |
| JOBSA | 120 | Left half: First free location in user area (set by Loader).<br>Right half: Starting address of user's program. |

Table 3-1 (Cont)
Job Data Area Locations

| Name | Relative Location(s) Octal | Description |
|---|---|---|
| JOBFF | 121 | Left half: 0<br>Right half: Address of the first free location following the user's program. Set to C(JOBSA)$_{LH}$ by RESET UUO. |
| JOBREN | 124 | Set by user and used by REENTER command as an alternate entry point. |
| JOBAPR | 125 | Contains user location to be trapped to when APR trap occurs (see APRENB UUO, Section 4.3.3.1). |
| JOBCNI | 126 | Set by CONI APR when an APR trap occurs to user program so that it can see APR flags (see APRENB UUO). |
| JOBTPC | 127 | APR trap PC stored here on APR trap to user program so that execution can be continued (see APRENB UUO). |
| JOBOPC | 130 | The previous (old) contents of the user's program counter are stored here by Monitor upon execution of a DDT, REENTER START, or CSTART command. |
| JOBCHN | 131 | Left half: 0<br>Right half: Address of first location after first FORTRAN IV Block Data. |
| JOBCOR | 133 | Left half: Unused<br>Right half: Highest core address for SAVE, GET, and RUN (i.e., user's 3rd argument). |

NOTE: Only those JOBDAT locations of significant importance to the user are given in this table. JOBDAT locations not listed include those which are used by the Monitor and those which are unused at the present time.

Some locations in the job data area, such as JOBSA and JOBDDT, are set by the user's program for use by the Monitor. Others, such as JOBREL, are set by the Monitor for use by the user's program. In particular, the right half of JOBREL contains the highest legal address set by the Monitor whenever the user's core allocation changes.

User programs must reference locations in the job data area with the assigned mnemonics, which must be declared as EXTERNAL references to the assembler. The values are assigned when the loader performs an automatic library search for undefined global references. The specific library subfile, in which these symbols are defined, is called JOBDAT.

### 3.2.2 Loading Relocatable Binary Files

The relocatable binary loader (LOADER) resides in the system file, and is started by the command

R LOADER core° Example: R LOADER 5

The PDP-10 Systems User's Guide contains a description of the loader command string. Figure 3-2 shows the user's core area with the loader resident.

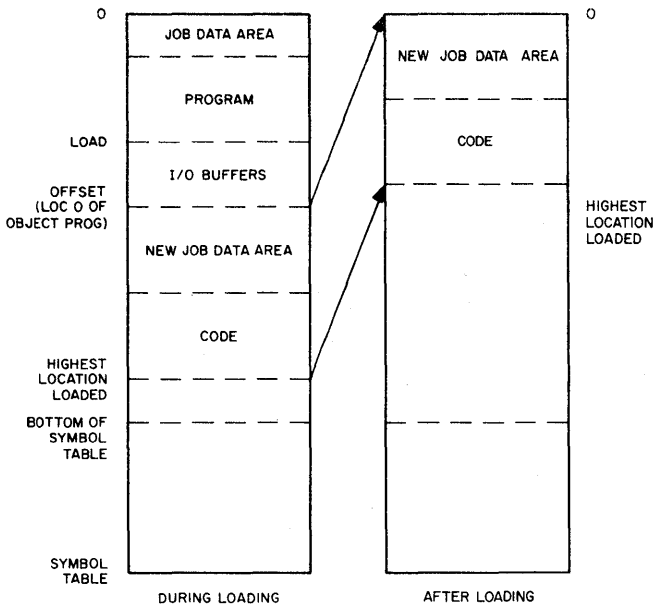



Figure 3-2 Loading User Core Area

3.2.2.1 Program Origin - The new program code is loaded upward from an offset above the resident loader. The program origin (i.e., the first location loaded) is $140_8$, unless the user changes it by the assembler LOC pseudo-instruction. The symbol table is built down from the top of the allocated core area. If the code and symbol table overlap, the core area is expanded by 1024 words and the symbol table is moved up to the top of the expanded area. Upon completion of loading, the loader stores some values in the new job data area, and moves the area from the offset to the highest location loaded (top of new code) down to zero. The symbol table remains at the top of the allocated core.

3.2.2.2 Program Break - After loading, the address of the first location above the new code area (i.e., the program break) lies in the left half of location JOBSA and in the right half of JOBFF. The left half of JOBFF contains 0.

3.2.2.3 Starting Addresses - The right half of JOBSA contains the program starting address. The value is the last nonzero address field of the assembler END pseudo-instruction to be loaded, or 0. This is the address used by the RUN and START commands.

If DDT was loaded by means of the D switch in the loader command string, the right half of JOBDDT is set by DDT to the starting address of DDT; the left half is 0; otherwise, the contents of JOBDDT are zero, the DDT command uses this address as the starting address. Location JOBREN may be set by the user's program for use with the REENTER command (see Chapter 2).

3.2.2.4 Symbol Table - JOBSYM contains a pointer to the bottom of the symbol table. The left half is the negative word length of the table, and the right half is the address of the lowest location used. The top of the symbol table is the top of the user's allocated core area, pointed to by the contents of the right half of JOBREL. DDT uses this symbol table for printing and interpreting symbolic values.

The right half of JOBUSY is the beginning address of the list of undefined global symbols. If some symbols are undefined after loading is complete, DDT may be used to define their values. These values are automatically substituted by DDT in all locations referencing them.

# CHAPTER 4
# USER PROGRAMMING

The central processor operates in one of three modes: executive mode, user I/O mode, or user mode. The Monitor operates in executive mode, which is characterized both by the lack of memory protection and relocation (see Chapter 3) and by normal execution of all defined operation codes. The user I/O mode is a special mode, wherein memory protection and relocation are in effect, as well as the normal execution of all defined operation codes. (This mode is not used by the Monitor, and is not normally available (see TRPSET) to the time-sharing user.) User programs are run in user mode, to guarantee the integrity of both the Monitor and each user program.

## 4.1 USER MODE

The user mode of the central processor is characterized by the following features:

a. Automatic memory protection and relocation (see Chapter 3)

b. Trap to absolute location 40 on

   (1) Operation codes 40 through 77 and 0;

   (2) Input/output instructions (DATAI, DATAO, BLKI, BLKO, CONI, CONO, CONSZ, and CONSO);

   (3) HALT (i.e., JRST 4,); or

   (4) Any JRST instruction that attempts to enter executive mode or user I/O mode.

c. Trap to relative location 40 on execution of operation codes 001 through 037.

Since user programs run in user mode, the Monitor must perform all input/output operations for the user, as well as any other operations required by the user not available in the user mode.

## 4.2 PROGRAMMED OPERATORS (UUO's)

Operation codes 000 through 077 are programmed operators (sometimes referred to as UUO's - Unimplemented User Operators); some trap to the Monitor and the rest trap to the user program.

After the effective address calculation is complete, the contents of the instruction register are stored in user or Monitor location 40, along with the effective address, and the instruction in user or Monitor location 41 is executed out of normal sequence. Location 41 must contain a JSR instruction to a routine to interpret the contents of location 40.

### 4.2.1 Operation Codes 001-037 (User UUO's)

Operation codes 001 through 037 do not effect the mode of the central processor. Thus, when executed in user mode, they trap to user location 40, which allows the user complete freedom in the use of these programmed operators.

### 4.2.2 Operation Codes 040-077, and 0 (Monitor UUO's)

Operation codes 040 through 077 and 0 trap to absolute location 40, with the central processor in executive mode. These programmed operators are interpreted by the Monitor to perform input/output operations and other control functions for the user's program.

Table 4-1 lists the operation codes and their mnemonics.

### 4.2.3 Operation Codes 100-127 (Unimplemented Op Codes)

| | |
|---|---|
| Op code 100-UJEN | Dismisses realtime interrupt from user mode (see 4.3.6.2). |
| Op codes 101-127 | Monitor prints ILL INST AT USER n and stops job. |

4.2.3.1 CALL and CALLI - Operation codes 040 through 077 limit the Monitor to $40_8$ operations. The CALL operation extends this set by specifying the name of the operation by the contents of the location specified by the effective address, e.g., CALL [SIXBIT/EXIT/]. This provides for indefinite extendability of the Monitor operations, at the overhead cost to the Monitor of a table lookup.

The CALLI operation eliminates the table lookup of the CALL operation by having the programmer perform the lookup once, and specifying an index to the operation in the effective address of the CALLI. Table 4-2 lists the Monitor operations specified by the CALL and CALLI operations

The customer is allowed to add his own CALL and CALLI calls to the Monitor. A negative CALLI effective address (starting with -2) should be used to specify such customer added operations.

### 4.2.4 Illegal Operation Codes

The eight input/output instructions (DATAI, etc.) and JRST instructions attempting to enter executive or user I/O mode from the user mode are interpreted by the Monitor as illegal instructions. The job is stopped and the following error message is printed on the user's console.

ERROR IN JOB n
ILL INST AT USER LOC addr

## 4.3 PROGRAM CONTROL

### 4.3.1 Starting

All program starting is accomplished by the Monitor commands RUN, START, CSTART, CONT, CCONT, DDT, and REENTER (see Chapter 2). The starting address is either an argument of the command or stored in the user's job data area (see Chapter 3).

4.3.1.1 CALL AC, [SIXBIT/SETDDT/] or CALLI AC, 2 - This UUO causes the contents of the AC to replace the DDT starting address, which is stored in the protected job data area location, JOBDDT. This starting address is used by the Monitor command, DDT.

### 4.3.2 Stopping

Any one of the following procedures can stop a running program:

a. One ↑C from user console if user program is in a Teletype input wait; otherwise, two ↑C's from user console (See Chapter 2);

b. A Monitor detected error; or

c. Program execution of HALT, CALL [SIXBIT/EXIT/], or CALL [SIXBIT/LOGOUT/].

4.3.2.1 Illegal Instructions (700-777, JRST 10, JRST 14) and Unimplemented Op codes (101-127) - Illegal instructions trap to the Monitor, stop the job, and print

ERROR IN JOB
ILL. INST. AT USER n

Note that the program cannot be continued by typing the CONT or CCONT commands.

4.3.2.2 HALT or JRST 4, - The HALT instruction is an exception to the illegal instructions; it traps to the Monitor, stops the job, and prints

ERROR IN JOB
HALT AT USER n

However, the CONT and CCONT commands are still valid and, if typed, will continue the program at the effective address of the HALT instruction. HALT is useful for impossible error returns such as INIT on TTY.

Table 4-1
Monitor Operation Codes

| Operation Code | Mnemonic | Function |
|---|---|---|
| 040 | CALL | Operation code extension (See 4.2.3.1) |
| 041 | INIT | Initialize I/O device (See 4.4.2.2) |
| 042 | | No operation — Reserved for installation-dependent calls |
| 043 | | No operation |
| 044 | | No operation |
| 045 | | No operation |
| 046 | | No operation |
| 047 | CALLI | Operation code extension (See 4.2.3.1) |
| 050 | OPEN | Open file (See 4.4.2.2) |
| 051 | TT CALL | Special Teletype Operations (See 5.1.3) |
| 052 | | No operation — Reserved for future expansion |
| 053 | | No operation |
| 054 | | No operation |
| 055 | RENAME | Rename or delete a file (See 4.4.2.5) |
| 056 | IN | Input and Skip (See 4.4.3) |
| 057 | OUT | Output and Skip (See 4.4.3) |
| 060 | GETSTS | Set file status (See 4.4.4) |
| 061 | STATO | Skip on file status one (See 4.4.4) |
| 062 | STATUS | Read file status (See 4.4.4) |
| 063 | STATZ | Skip on file status zero (See 4.4.4) |
| 064 | INBUF | Set up input buffer ring (See 4.4.2.3) |
| 065 | OUTBUF | Set up output buffer ring (See 4.4.2.3) |
| 066 | INPUT | Read (See 4.4.3) |
| 067 | OUTPUT | Write (See 4.4.3) |
| 070 | CLOSE | Close file (See 4.4.5) |
| 071 | RELEASE | Release device (See 4.4.7) |
| 072 | MTAPE | Position tape (See 5.8.2 and 5.7.5) |
| 073 | UGETF | Get next free block number (See 5.7.5) |
| 074 | USETI | Set next input block number (See 5.7.5) |
| 075 | USETO | Set next output block number (See 5.7.5) |
| 076 | LOOKUP | Select file (See 4.4.2.4) |
| 077 | ENTER | Create file (See 4.4.2.4) |
| 100 | UJEN | Dismiss real-time interrupt (See 4.3.6.2) |

Table 4-2
CALL and CALLI Monitor Operations

| CALLI AC, x | CALL AC, [SIXBIT/y/] | Function |
|---|---|---|
| x = -2,...,-n | Customer defined | Reserved for definition by each customer installation. |
| -1 | DATAO | Displays AC in console lights. |
| 0 | y = RESET | Reset I/O devices (See 4.4.2.1) |
| 1 | DDTIN | DDT mode console input (See 5.1.2) |
| 2 | SETDDT | Set protected DDT starting address (See 4.3.1.1) |
| 3 | DDTOUT | DDT mode console output (See 5.1.2) |
| 4 | DEVCHR | Get device characteristics (See 5.11) |
| 5 | (DDTGT) | No operation |
| 6 | (GETCHR) | Same as DEVCHR(4) |
| 7 | (DDTRL) | No operation |
| 10 | WAIT | Wait until device inactive (See 4.4.6) |
| 11 | CORE | Allocate core (See 4.5) |
| 12 | EXIT | Release devices, stop job (See 4.3.2.3) |
| 13 | UTPCLR | Clear directory (See Table 5-2) |
| 14 | DATE | Return data (See 4.3.4.1) |
| 15 | LOGIN | Special operation for LOGIN (See 4.3.5.3) |
| 16 | APRENB | Enable central processor traps (See 4.3.3.1) |
| 17 | LOGOUT | Kill job (See 4.3.2.4) |
| 20 | SWITCH | Read processor console switches (See 4.3.6.3) |
| 21 | REASSI | Reassign device (See 2.4.4) |
| 22 | TIMER | Read clock in ticks (See 4.3.4.2) |
| 23 | MSTIME | Read clock in milliseconds (See 4.3.4.3) |
| 24 | GETPPN | Read project-programmer pair (See 4.3.5.2) |
| 25 | TRPSET | Set trap for user I/O mode (See 4.3.6.1) |
| 26 | TRPJEN | Illegal UUO |
| 27 | RUNTIM | Return job running time (See 4.3.4.4) |
| 30 | PJOB | Return job number (See 4.3.5.1) |
| 31 | SLEEP | Stop job for specified time (See 4.3.4.5) |
| 32 | (SETPOV) | Set pushdown overflow trap (this command has been superceded by APRENB (16). |
| 33 | PEEK | Return specified Monitor location (See 4.3.5.4) |

Table 4-2 (Cont)
CALL and CALLI Monitor Operations

| CALLI AC, x | CALL AC, [SIXBIT/y/] | Function |
|---|---|---|
| 34 | GETLIN | Return physical name of attached Teletype console. (See 4.3.5.5) |
| 35 | RUN | Call new program (both high and low) (See Addendum III) |
| 36 | SETUWP | Set user's mode write protect (See Addendum III) |
| 37 | REMAP | Remap top of low segment into high segment (See Addendum III) |
| 40 | GETSEG | Replace high segment only (See Addendum III) |
| 41 | GETTAB | Examine contents of specified Monitor location (See 4.3.5.6) |
| 42 | SPY | Make physical core be high segment for efficient looking at Monitor (See Addendum III) |
| 43 | SETNAM | Set program name (See 4.3.6.4) |

Note: Other CALLI UUOs will be implemented from time to time and will be documented in Software Manual Updates and in revised editions of this manual. Execution of a CALLI UUO with an address higher than the last implemented operator will result in an ILLEGAL UUO message.

4.3.2.3 CALL [SIXBIT/EXIT/] or CALLI 12 - All input/output devices are RELEASed (see Section 4.4.7), and the job is stopped.

EXIT
↑C

is printed on the user's console, which is left in Monitor mode. The CONT or CCONT commands cannot continue the program.

4.3.2.4 CALL N, [SIXBIT/EXIT/] or CALLI N, 14 - When N = 1, the job is stopped but devices are not released. The carriage return-linefeed operation will be performed and

is printed on the user's console and the CONT command will return after the UUO instead of printing CAN'T CONTINUE.

4.3.2.5 CALL [SIXBIT/LOGOUT/] or CALLI 17 - All input/output devices are RELEASed (see Section 4.4.7), and returned to the Monitor pool, along with the allocated core and the job number. The ac-

cumulated running time of the job is printed on the user's console, which is left in the detached mode. This UUO is not available to user programmers. It is only for use of the LOGOUT CUSP. If a user program executes a LOGOUT UUO, the Monitor will treat it like EXIT (See 4.3.2.3).

### 4.3.3 Trapping

4.3.3.1 CALL AC, [SIXBIT/APRENB/] or CALLI AC, 16 - APR trapping allows a user to handle any and all traps that occur on the central processor, including illegal memory references, nonexistent memory references, pushdown list overflow, arithmetic overflow, floating point overflow, and clock flag. To enable for trapping a CALL AC, [SIXBIT/APRENB/] or CALLI AC, 16 is executed, where the AC contains the central processor flags to be tested on interrupts, as defined below:

| AC Bit | | Trap On |
|---|---|---|
| 19 | 200000 | pushdown overflow* |
| 22 | 20000 | memory protection violation * |
| 23 | 10000 | nonexistent memory flag* |
| 26 | 1000 | clock flag* |
| 29 | 100 | floating point overflow |
| 32 | 10 | arithmetic overflow |

When one of the specified conditions occurs while the central processor is in user mode, the state of the central processor is Conditioned Into (CONI) location JOBCNI, and the PC is stored in location JOBTPC in the job data area (see Table 3-1). Then control is transferred to the user trap-answering routine specified by the contents of the right half of JOBAPR, after the arithmetic overflow flag has been cleared. The user program must set up location JOBAPR before executing the CALL AC [SIXBIT/APRENB/] or CALLI AC, 16. To return control to his interrupted program, the user's trap answering routine must execute a JRST 2, @ JOBTPC to restore the state of the processor.

4.3.3.2 Console-Initiated Traps - Program control can be changed from the user's console by use of the ↑C, START, DDT, and REENTER commands (see Chapter 2).

### 4.3.4 Timing Control

The central processor clock, which generates interrupts at the power-source frequency (60 Hz in North America, 50 Hz in most other countries), keeps time in the Monitor. Each clock interrupt (tick) corresponds to 1/60th (or 1/50th) of a second of elapsed real time. The clock is set initially to the current time of day by console input when the system is started, as is the current date. When the clock reaches midnight, it is reset to zero, and the date is advanced.

*The Monitor is always enabled for these.

4.3.4.1 CALL AC, [SIXBIT/DATE/] or CALLI AC, 14 - A 12-bit binary integer computed by the formula

$$date=( (year-1964)x12+(month-1) )x31+day-1$$

represents the date.

This integer representation is returned right-justified in accumulator AC.

4.3.4.2 CALL AC, [SIXBIT/TIMER/] or CALLI AC, 22 - These return the time of day, in clock ticks (jiffies), right-justified in accumulator AC.

4.3.4.3 CALL AC, [SIXBIT/MSTIME/] or CALLI AC, 23 - These return the time of day, in milliseconds right-justified in accumulator AC.

4.3.4.4 CALL AC, [SIXBIT/RUNTIM/] or CALLI AC, 27 - The accumulated running time, in milliseconds, of the job whose number is in accumulator AC, is returned right-justified in accumulator AC. If the job number in AC is zero, the running time of the currently running job is returned. If the job whose number is in AC does not exist, zero is returned.

4.3.4.5 CALL AC, [SIXBIT/SLEEP/] or CALLI AC, 31 - These stop the job, and continue automatically after an elapsed real time of

$$[c(AC)x\text{clock frequency}] \text{ modulo } 2^{12} \text{ jiffies.}$$

The contents of the AC are thus interpreted as the number of seconds the job wishes to sleep; however, there is an implied maximum of approximately 68 seconds or one minute.

4.3.5 Identification

4.3.5.1 CALL AC, [SIXBIT/PJOB/] or CALLI AC, 30 - These return the job number right-justified in accumulator AC.

4.3.5.2 CALL AC, [SIXBIT/GETPPN/] or CALLI AC, 24 - These return in AC the project-programmer pair of the job. The project number is a binary number in the left half of AC, and the programmer number is a binary number in the right half of AC. If the program being run is LOGIN or LOGOUT from the system device, the current project-programmer number is changed to 1,2 so that all files are accessible for reading and writing, and a skip return is given if the old project-programmer number is also logged in on another job.

4.3.5.3 CALL AC, [SIXBIT/LOGIN/] or CALLI AC, 15 - This programmed operator is intended for use with the LOGIN command only. Accumulator AC contains XWD -n, TABLE, where TABLE is the first location of n words to be stored in the Monitor's job tables for this user. The first table is project-programmer number (PRJPRG); the second is the job privilege bits (JBTPRV). If LH is less than -2, the extra words are ignored. If LH is -1, only the first table is set.

4.3.5.4 CALL AC, [SIXBIT/PEEK/] or CALLI AC, 33 - This UUO allows a user program to examine any location in the Monitor. Some customers may want to restrict the use of this UUO to project 1.

```
The call is:    MOVEI AC, exec address      ;TAKEN MODULO 16K
                CALL AC, [SIXBIT/PEEK/]     ;OR CALLI AC, 33
```

This call returns with the contents of the Monitor location in AC. It is used by SYSTAT and could be used for on-line Monitor debugging.

4.3.5.5 CALL AC, [SIXBIT/GETLIN/] or CALLI AC, 34 - This UUO returns the SIXBIT physical name of the Teletype console that the program is attached to.

```
The call is:    CALL AC, [SIXBIT/GETLIN/]   ;OR CALLI AC, 34
```

The name is returned left justified in the AC.

```
Example:                        CTY or TTY3 or TTY30
```

This UUO is used by the LOGIN program to print the TTY name.

4.3.5.6 CALL AC, [SIXBIT/GETTAB/] or CALLI AC, 41 - This UUO provides a mechanism for user programs to examine the contents of certain Monitor locations in a way which will not vary from Monitor to Monitor.

```
The call is:    CALL AC, [SIXBIT/GETTAB/]   ;OR CALLI AC, 41
```

The left half of AC contains a job number or an index to a table. Some job numbers may refer to high segments of programs by using arguments greater than JOBN for the current Monitor. A negative LH means the current job number. The right half of AC contains a table number from the following list of Monitor data tables and parameters. The entries in these tables are all globals in the Monitor subroutine COMMON. The actual values of the core addresses of these locations are subject to change and can be found in the LOADER storage map for the Monitor.

The complete descriptions of these globals will be found in the listings of COMMON and S. The list of entries is as follows, with brief description.

Entries in CNFTBL (Configuration Table)- Table 11

| ITEM | LOCATION | USE |
|---|---|---|
| 0 | CONFIG | Name of systen in ASCIZ |
| - | | |
| 4 | CONFIG+4 | |
| 5 | SYSDAT | Date of system in ASCIZ |
| 6 | SYSDAT+1 | |
| 7 | SYSTAP | Name of the system device (SIXBIT) |
| 10 | TIME | Time of day in jiffies |
| 11 | THSDAT | Todays date (12-bit format) |
| 12 | SYSSIZ | Highest location in the monitor + 1 |
| 13 | DEVOPR | Name of the OPR TTY console |
| 14 | DEVLST | LH is start of DDB chain |
| 15 | SEGPTR | LH=-# of high segments, RH=+# of JOBS (counting NULL job) |
| 16 | TWOREG | Non-zero if system has two-register hardware and software |
| 17 | STATES | Location describing feature switches of this system in LH, and current state in RH.<br><br>Assembled according to MONGEN dialog and S.MAC:<br><br>Bit 0=1 If disk system (FTDISK)<br>Bit 1=1 If swap system (FTSWAP)<br>Bit 2=1 If LOGIN system (FTLOGIN)<br>Bit 3=1 If full duplex software (FTTTYSER)<br>Bit 4=1 If privilege feature (FTPRV)<br>Bit 5=1 If assembled for choice of reentrant or non-reentrant software at monitor load twice (FT2REL)<br>Bit 6=1 If clock is 50 cycle instead of 60 cycle<br><br>Deposited by operator any time:<br><br>Bit 34=1 Means no remote LOGINS<br>Bit 35=1 Means no more LOGINS |
| 20 | SERIAL | Serial number of PDP processor<br>Set by MONGEN dialog |

Entries in ODPTBL (once only disk parameters) - Table 15

| ITEM | LOCATION | USE |
|---|---|---|
| 0 | SWPHGH | Highest logical block # in the swapping space |
| 1 | K4SWAP | K of disk words set aside for swapping |
| 2 | PROT | In-core protect time multiplies size of job in K-1 |
| 3 | PROT0 | In-core protect time added to above result after multiply |

Entries in NSWTBL (non-swapping data) - Table 12

| ITEM | LOCATION | USE |
|---|---|---|
| 0 | CORTAB | Map of physical core |
| - | | 1 bit for each K of core |
| 7 | CORTAB+7 | |

| | | |
|---|---|---|
| 10 | CORMAX | Size in words of largest legal user job (low seg+high seg) |
| 11 | CORLST | Byte pointer to last free block in CORTAB |
| 12 | CORTAL | Total free+dormant+idle K physical core left |
| 13 | SHFWAT | Job no. shuffler has stopped |
| 14 | HOLEF | Abs. adr. of job above lowest hole, 0 if no job |
| 15 | UPTIME | Time system has been up in jiffies |
| 16 | SHFWRD | Tot. no. of words shuffled by system |
| 17 | STUSER | Number of job using sys if not a disk |
| 20 | HIGHJB | Highest job number currently assigned |
| 21 | CLRWRD | Total no. of words cleared by CLRCOR |
| 22 | LSTWRD | Total no. of clock ticks when null job ran and other jobs wanted to but couldn't, because:<br>1. Swapped out or on way in or out<br>2. Monitor waiting for IO to stop so can shuffle or swap<br>3. Job being swapped out because expanding core |

Entries in SWPTBL (swapping data) - Table 13

| ITEM | LOCATION | USE |
|---|---|---|
| 0 | BIGHOL | No. of K in biggest hole in core |
| 1 | FINISH | +Job no. of job being swapped out<br>-Job no. of job being swapped in |
| 2 | FORCE | Job being forced to swap out |
| 3 | FIT | Job waiting to be fit into core |
| 4 | VIRTAL | Amount of virtual core left in system in K (initially set to No. of K of swapping space) |
| 5 | SWPERC | LH=no. of swap read or write errors<br>RH=error bits (bits 18-21 same as status bits)+no. of K discarded |

Table Numbers (RH of AC)

00-JBTSTS-Index by job or segment number
01-JBTADR-Index by job or segment number
02-PRJPRG-Index by job or segment number
03-JBTPRG-Index by job or segment number
04-TTIME-Index by job number
05-JBTKCT-Index by job number
06-JBTPRV-Index by job number
07-JBTSWP-Index by job or segment number
10-TTYTAB-Index by line number
11-CNFTBL-Index by item number (see above)
12-NSWTBL-Index by item number (see above)
13-SWPTBL-Index by item number (see above)
14-JBTSGN-Index by job number
15-ODPTBL-Index by item number (see above)

An error return leaves the AC unchanged. This means that the job number or index number in the left half of AC was too high, or the table number in the right half of AC was too high, or that the user does not have the privilege of accessing that table.

A skip return supplies the contents of the requested table in AC, or a zero if the table is not defined in the current Monitor.

The SYSTAT CUSP makes heavy use of this UUO.

### 4.3.6 Direct User I/O

The user I/O mode (bits 5 and 6 of PC word = 11) of the central processor allows running privileged user programs with automatic protection and relocation in effect. This mode provides some protection against partially debugged Monitor routines, and permits running infrequently used device service routines as a user job. Direct control by the user program of special devices is particularly important in realtime applications.

To utilize this mode, the job number must be 1. CALL [SIXBIT/RESET/] or CALLI 0 terminates user I/O mode.

4.3.6.1 CALL AC, [SIXBIT/TRPSET/] or CALLI AC, 25 - This UUO is a privileged UUO which temporarily stops time sharing and allows the user program to gain control of the interrupt locations. This UUO is temporary until some "knave-proof" realtime UUOs are implemented which will not stop time sharing and which cannot crash the system. If the user is not job 1, or if AC contains either zero or the left half is not in the range 40 through 57, control returns to the next location after the CALL. Otherwise, all other jobs are stopped and, if AC contains zero, the central processor is placed in user I/O mode and control returns to the second location following the CALL. If the left half of AC contains a number between 40 and 57 inclusive, the contents of the relative location specified in the right half of AC are fetched; the job relocation address is added to the address field, and the result is stored in the absolute location (40-57) specified in the left half of AC; the central processor is placed in the user I/O mode; and control is returned to the second location following the CALL. Thus, the user can set up a priority interrupt trap into his relocated core area.

The call is:

```
MOVE AC, XWD N, ADR
CALL AC, [SIXBIT/TRPSET/]
ERROR RETURN
NORMAL RETURN
```

The Monitor assumes that user location ADR contains either a JSR U or BLKI U, where U is a user address. Consequently, the Monitor will add the job's relocation to the contents of location U to make it an absolute IOWD. Therefore, a user should reset the contents of U before every TRPSET call.

4.3.6.2 UJEN (Op code 100) - This unimplemented op code dismisses a user I/O mode interrupt if one is in progress. If the interrupt is from user mode, a JRST 12, instruction can dismiss the interrupt.

If the interrupt was from executive mode, however, this operator must be used to dismiss the interrupt. The program must restore all accumulators, and execute

UJEN U

where user location U contains the program counter as stored by a JSR instruction when the interrupt occurred.

4.3.6.3 CALL AC, [SIXBIT/SWITCH/] or CALLI AC, 20 - These return the contents of the central processor data switches in AC. Caution must be exercised in using the data switches since they are not an allocated device and are always available to all users.

4.3.6.4 CALL AC, [SIXBIT/SETNAM/] or CALLI AC, 43 - The contents of AC contain a left justified SIXBIT program name, which is stored in a Monitor job table. This UUO is used by the LOADER. The information in the table is used by the SYSTAT CUSP (See GETTAB UUO 4.3.5.6).

## 4.4 INPUT/OUTPUT PROGRAMMING

All user input/output operations are controlled by the use of Monitor programmed operators. These are device independent, in the sense that if an operator is not pertinent to a given device, the operator is treated as a no-operation code. For example, a rewind directed to a line printer does nothing. Devices are referenced by logical names or physical names (see ASSIGN command, Chapter 2), and the characteristics of a device can be obtained from the Monitor. Properly used, these systems characteristics permit the programmer to delay the device specification for his program from program-generation until program-run time. I/O is accomplished by associating a device, a file, and a ring buffer or command list with one of a user's I/O channels.

### 4.4.1 File

A file is an ordered set of data on a peripheral device. Its extent on input is determined by an end-of-file condition dependent on the device. For instance, a file is terminated by reading an end-of-file gap from magnetic tape, by an end-of-file card from a card reader, or by depressing the end-of-file switch on a card reader (see Chapter 5). The extent of a file on output is determined by the amount of information written by the OUT or OUTPUT programmed operators up through and including the next CLOSE or RELEASE operator.

4.4.1.1 Device - To specify a file, it is necessary to specify the device from which the file is to be read or onto which the file is to be written. This specification is made by an argument of the INIT or

OPEN programmed operators. Devices are separated into two categories--those with no filename directory, and those with one or more filename directories.

a. Nondirectory Devices - For nondirectory devices, e.g., card reader, line printer, paper tape reader and punch, and user console, the only file specification required is the device name. All other file specifiers, if given, are ignored by the Monitor. Magnetic tape, which is also a nondirectory device, requires, in addition to the name, that the tape be properly positioned. Even though LOOKUP is not required to read and ENTER is not required to write, it is advisable to always use them so that a directory device may be substituted for a nondirectory device at run time (using the Monitor command, ASSIGN). Only in this way can user programs be truly device independent.

b. Directory Devices - For directory devices, e.g., DECtape and disk, files are addressable by name. If the device has a single file directory, e.g., DECtape, the device name and filename are sufficient information to determine a file. If the device has multiple file directories, e.g., disk, the name of the file directory must also be specified. These names are specified as arguments to the LOOKUP, ENTER, and RENAME programmed operators.

4.4.1.2 Data Modes - Data transmissions are either unbuffered (dump) or buffered. The mode of transmission is specified by a 4-bit argument to the INIT, OPEN, or SETSTS programmed operators. Table 4-3 summarizes the data modes.

Table 4-3
Data Modes

| Octal Code Buffered Modes | Mnemonic | Meaning |
|---|---|---|
| 0 | A | ASCII. 7-bit characters packed left justified, five characters per word. |
| 1 | AL | ASCII line. Same as 0, except that the buffer is terminated by a FORM, VT (vertical tab), LINE-FEED or ALTMODE character. |
| 2-7 | | Unused. |
| 10 | I | Image. A device dependent mode. The buffer is filled with data exactly as supplied by the device. |
| 11-12 | | Unused. |
| 13 | IB | Image binary. 36-bit bytes. This mode is similar to binary mode, except that no automatic formatting or checksumming is done by the Monitor. |
| 14 | B | Binary. 36-bit byte. This is a blocked format consisting of a word count, n (the right half of the first data word of the buffer), followed by n 36-bit data words. Checksumming is done for cards and paper tape. |

Table 4-3 (Cont)
Data Modes

| Octal Code Unbuffered Modes | Mnemonic | Meaning |
|---|---|---|
| 15 | ID | Image Dump. A device dependent dump mode. |
| 16 | DR | Dump as records without core buffering. Data is transmitted between any contiguous blocks of core and one or more standard length records on the device for each command word in the command list. |
| 17 | D | Dump one record without core buffering. Data is transmitted between any contiguous block of core and exactly one record of arbitrary length on the device for each command word in the command list. |

a. Unbuffered Data Modes - Data modes 15, 16 and 17 utilize a command list to specify areas in the user's allocated core to be read or written. The effective address of the IN, INPUT, OUT, and OUTPUT programmed operators points to the first word of the command list. Three types of entries may occur in the command list.

(1) IOWD n, loc - Causes n words from loc through loc+n-1 to be transmitted. The next command is obtained from the next location following the IOWD. The assembler pseudo-op IOWD generates XWD -n, loc-1.

(2) XWD 0, y - Causes the next command to be taken from location y. Referred to as a GOTO word.

(3) 0 - Terminates the command list.

The Monitor does not return program control to the user until the command list has been completely processed. If an illegal address is encountered while processing the list, an APR trap occurs if the user has enabled the central processor for "illegal memory" references; otherwise, the job is stopped and the Monitor prints

ADDRESS CHECK AT USER LOC addr

on the user's console, leaving the console in Monitor mode.

b. Buffered Data Modes - Data modes 0, 1, 10, 13, and 14 utilize a ring of buffers in the user area and the priority interrupt system to permit the user to overlap computation with his data transmission. Core memory in the user's area serves as an intermediate buffer between the user's program and the device. A ring of buffers consists of a 3-word header block for bookkeeping and a data storage area subdivided into one or more individual buffers linked together to form a ring. During input operations, the Monitor fills a buffer, makes the buffer available to the user's program, advances to the next buffer

in the ring and fills it if it is free. The user's program follows along behind, emptying the next buffer if it is full, or waiting for the next buffer to fill. During output operations, the user's program and the Monitor exchange roles, the user filling the buffers and the Monitor emptying them.

(1) Buffer Structure - A ring of buffers consists of a 3-word header block and a data storage area subdivided into one or more individual buffers linked together to form a ring. The ring buffer layout is shown in Figure 4-1, and explained in the paragraphs which follow.

(a) Buffer Header Block - The location of the 3-word buffer header block is specified by an argument of the INIT and OPEN operators. Information is stored in the header by the Monitor in response to user execution of Monitor programmed operators. The user's program finds all the information required to fill and empty buffers in the header. Bit position 0 of the first word of the header is a flag which, if 1, means that no input or output has occurred for this ring of buffers. The right half of the first word is the address of the second word of the buffer currently in use by the user's program. The second word of the header contains a byte pointer to the current byte in the current buffer. The byte size is determined by the data mode. The third word of the header contains the number of bytes remaining in the buffer.

(b) Buffer Data Storage Area - The buffer data storage area is established by the INBUF and OUTBUF operators, or, if none exists when the first IN, INPUT, OUT, or OUTPUT operator is executed, a 2-buffer ring is set up. The effective address of the INBUF and OUTBUF operators specifies the number of buffers in the ring. The location of the buffer storage area is specified by the contents of the right half of JOBFF in the user's job data area. The Monitor updates JOBFF to point to the first location past the storage area.

All buffers in the ring are identical in structure. As Figure 4-2 shows, the right half of the first word contains the file status at the time that the Monitor advanced to the next buffer in the ring. Bit 0 of the second word of a buffer, called the use bit, is a flag that indicates whether the buffer contains active data. This bit is set to 1 by the Monitor when the buffer is full on input or being emptied on output, and set to 0 when the buffer is empty on output or is being filled on input. The use bit prevents the Monitor and the user's program from interfering with each other by attempting to use the same buffer simultaneously. Bits 1 through 17 of the second word of the buffer contain the size of the data area of the buffer which immediately follows the second word. The size of this data area depends on the device.

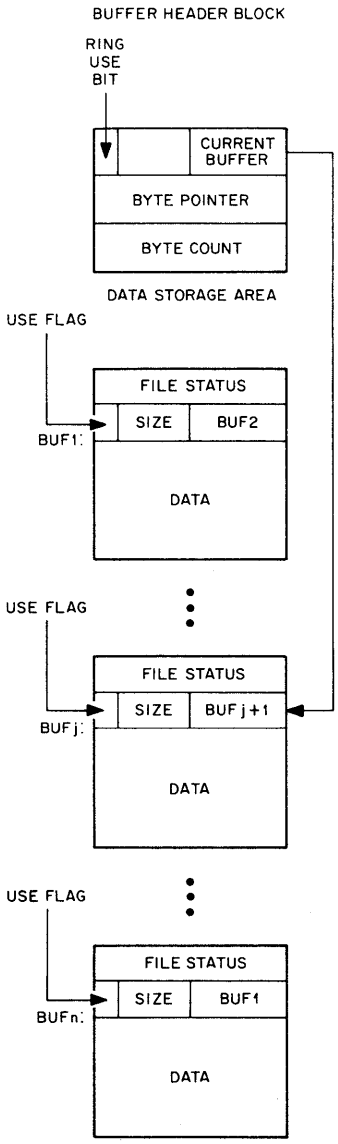



Figure 4-1 User's Ring of Buffers

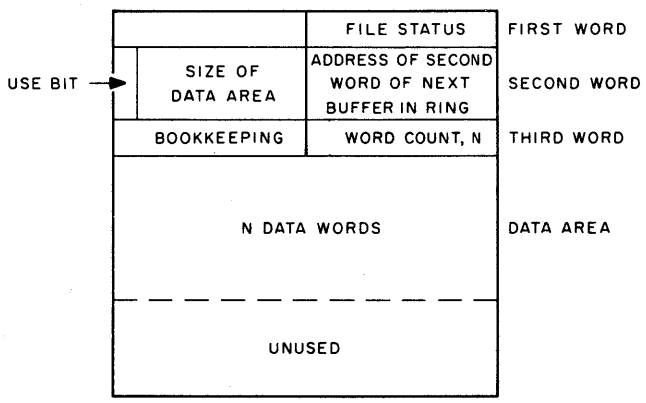



Figure 4-2 Detailed Diagram of Individual Buffer

The right half of the first word of the data area of the buffer, i.e., the third word of the buffer, is reserved for a count of the number of words (excluding itself) that actually contain data. The left half of this word is reserved for other bookkeeping purposes, depending on the particular device and the data mode.

4.4.1.3 File Status - The file status is a set of 18 bits (right half word), which reflects the current state of a file transmission. The initial status is a parameter of the INIT and OPEN operators. Thereafter, bits are set by the Monitor, and may be tested and reset by the user via Monitor programmed operators. Table 4-4 defines the file status bits. All bits, except the end-of-file bit, are set immediately by the Monitor as the conditions occur, rather than being associated with the buffer that the user is currently working on. However, the file status is stored with each buffer so that the user can determine which bufferful produced an error. A more thorough description of bits 18 through 29 is given in Chapter 5.

Table 4-4
File Status

| Bit | Meaning |
|---|---|
| 18 | Improper mode, e.g., attempt to write on a write-locked tape. |
| 19 | Device detected error, other than hardware checksum or parity. Checksum, and/or parity error detected by hardware and/or software. |
| 20 | Data error, e.g., a computed checksum failed or invalid data was received. |
| 21 | Block too large. A block of data from a device is too large to fit in a buffer, or a block number is too large. |
| 22 | End of file. |
| 23 | Device is actively transmitting or receiving data. |
| 24-29 | Device dependent parameters. (See Chapter 5.) |
| 30 | Synchronous input. Stop the device after each buffer is filled. |
| 31 | Forces the Monitor to use the word count in the first data word of the buffer (output only). The Monitor normally computes the word count from the byte pointer in the buffer header. |
| 32-35 | Data mode. See Table 4-3. |

### 4.4.2 Initialization

4.4.2.1 Job Initialization - The Monitor programmed operator

CALL [SIXBIT/RESET/] or CALLI 0

should be the first instruction in each program. It immediately stops all input/output transmissions on all devices without waiting for the devices to become inactive. All device allocations made by the INIT and OPEN operators are cleared, and, unless the devices have been assigned by the ASSIGN command (see Chapter 2), the devices are returned to the Monitor facilities pool. The content of the left half of JOBSA (program break) is stored in the right half of JOBFF so that the user buffer area is reclaimed if the program is starting over. The left half of JOBFF is cleared. Any files which have not been closed will be deleted on disk. Any older version having the same filename will remain.

4.4.2.2 Device Initialization

```
OPEN D,SPEC
error return
normal return
      :
      .
SPEC:EXP STATUS
     SIXBIT/ldev/
     XWD OBUF,IBUF
```

```
INIT D,STATUS
SIXBIT/ldev/
XWD OBUF,IBUF
error return
normal return
```

The OPEN (operation code 050) and INIT (operation code 041) programmed operators initialize a file by specifying a device, ldev, and initial file status, STATUS, and the location of the input and output buffer headers.

a. Data Channel - OPEN and INIT establish a correspondence between the device, ldev, and a 4-bit data channel number, D. Most of the other input/output operators require this channel number as an argument. If a device is already assigned to channel D, it is released. (See RELEASE in this chapter.) The device name, ldev, is either a logical or physical name, with logical names taking precedence over physical names. (See ASSIGN command, Chapter 2.) If the device, ldev, is not the system device, SYS, and is allocated to another job or does not exist, the error return is taken. If the device is the system device, SYS, the job is stopped in a system device wait queue, and will continue running when SYS becomes available.

b. Initial File Status - The file status, including the data mode, is set to the value of the symbol STATUS. If the data mode is not legal (see Chapter 5) for the specified device, the job is stopped and the Monitor prints

ILL DEVICE DATA MODE FOR DEVICE dev AT USER addr,

where dev is the physical name of the device and addr is the location of the OPEN or INIT operator, on the user's console and leaves the console in Monitor mode.

c. Buffer Header - Symbols OBUF and IBUF, if nonzero, specify the location of the first word of the 3-word buffer header for output and input respectively. Only those headers which are to be used need to be specified. For instance, the output header need not be specified, if only input is to be done. Also, modes 15, 16, and 17 require no header. If either of the buffer headers of the 3-word block starting at location SPEC lies outside the user's allocated core area[1], an illegal memory violation occurs. If the user has enabled the central processor for illegal memory traps (see APRENB in this chapter), the trap occurs. Otherwise, the job is stopped and the Monitor prints

ADDRESS CHECK FOR DEVICE dev AT USER LOC addr

where addr is the address of the OPEN or INIT operator, on the user's console and leaves the console in Monitor mode.

The first and third words of the buffer header are set to zero. The left half of the second word is set up with the byte pointer size field in bits 6 through 11 for the selected device-data mode combination.

4.4.2.3 Buffer Initialization - Buffer data storage areas may be established by the INBUF and OUTBUF programmed operators, or by the first IN, INPUT, OUT, or OUTPUT operator, if none exists at that time, or the user may set up his own buffer data storage area.

a. Monitor Generated Buffers - Each device has associated with it a standard buffer size (see Chapter 5). The Monitor programmed operators INBUF D, n (operation code 064) and OUTBUF D, n (operation code 065) set up a ring of n standard size buffers associated with the input and output buffer headers, respectively, specified by the last OPEN or INIT operator on data channel D. If no OPEN or INIT operator has been performed on channel D, the Monitor stops the job and prints.

I/O TO UNASSIGNED CHANNEL AT USER LOC addr

where addr is the location of the INBUF or OUTBUF operator, on the user's console leaving the console in Monitor mode.

The storage space for the ring is taken from successive locations, beginning with the location specified in the right half of JOBFF. This is set to the program break, which is the first free location above the program area, by RESET. If there is insufficient space to set up the ring, an "illegal memory" violation occurs, which will cause a trap, if the user has enabled for it (see APRENB in this chapter), or the Monitor will stop the job and print

ADDRESS CHECK FOR DEVICE ldev AT USER LOC addr

[1] Buffer headers may not be in the user's AC's. However, they may be in locations above JOBPFI.

where ldev is the physical name of the device associated with channel D and addr is the location of the INBUF or OUTBUF operator, on the user's console and leaves the console in Monitor mode.

The ring is set up by setting the second word of each buffer with a zero use bit, the appropriate data area size, and the link to the next buffer. The first word of the buffer header is set with a 1 in the ring use bit, and the right half contains the address of the second word of the first buffer.

b. User Generated Buffers - The following code illustrates an alternative to the use of the INBUF programmed operator. Analogous code may replace OUTBUF. This user code operates similarly to INBUF. SIZE must be set equal to the greatest number of data words expected in one physical record.

```
GO:     INIT 1, 0                          ;INITIALIZE ASCII MODE
        SIXBIT/MTA0/                       ;MAGNETIC TAPE UNIT 0
        XWD 0, MAGBUF                      ;INPUT ONLY
        JRST NOTAVL
        MOVE 0, [XWD 400000,BUF1+1] ;THE 400000 IN THE LEFT HALF MEANS THE
                                           ;BUFFER WAS NEVER REFERENCED.
        MOVEM 0, MAGBUF
        MOVE 0, [POINT BYTSIZ,0,35] ;SET UP NONSTANDARD BYTE SIZE
        MOVEM 0, MAGBUF+1
        JRST CONTIN                        ;GO BACK TO MAIN SEQUENCE
MAGBUF: BLOCK 3                            ;SPACE FOR BUFFER HEADER
BUF1:   0                                  ;BUFFER 1, 1ST WORD UNUSED
        XWD SIZE+2,BUF2+1                  ;LEFT HALF CONTAINS BUFFER SIZE,
                                           ;RIGHT HALF HAS ADDRESS OF NEXT BUFFER
        BLOCK SIZE+1                       ;SPACE FOR DATA, 1ST WORD RECEIVES
                                           ;WORD-COUNT. THUS ONE MORE WORD
                                           ;IS RESERVED THAN IS REQUIRED
                                           ;FOR DATA ALONE
BUF2:   0                                  ;SECOND BUFFER
        XWD SIZE+2,BUF3+1
        BLOCK SIZE+1
BUF3:   0                                  ;THIRD BUFFER
        XWD SIZE +2,BUF1+1                 ;RIGHT HALF CLOSES THE RING
        BLOCK SIZE+1
```

4.4.2.4 File Selection - The LOOKUP (operation code 076) and ENTER (operation code 077) programmed operators select a file for input and output respectively. Although these operators are not necessary for nondirectory devices, it is good programming practice to always use them so that directory devices may be substituted at run time. (See ASSIGN, Chapter 2.)

a.
```
        LOOKUP D, E
        error return
        normal return
        :
        :
E:      SIXBIT/file/                       ;filename, 1 to 6 characters.
        SIXBIT/ext/                        ;filename extension, 0 to 3 characters.
        0
        XWD project number, programmer number,
```

LOOKUP selects a file for input on channel D. If no device is associated with channel D, 7 is stored in bits 33 through 35 of location E+1, and the error return is taken. If the input side of channel D is not closed (see CLOSE, in this chapter), it is now closed. The output side of channel D is not affected. If the device associated with channel D does not have a directory, the normal return is now taken. If the device has multiple directories, e.g., disk, the Monitor searches the master file directory of the device for the user's file directory whose number is in location E+3 and whose extension is UFD. If E+3 contains zero, the project-programmer pair of the current job is used as the name of the user's file directory. If this file is not found in the master file directory, 1 is stored in bits 33 through 35 of location E+1 and the error return is taken.

The user's file directory or the device directory in the case of a single-directory device (e.g., DECtape) is searched for the file whose name is in location E and whose extension is in the left half of location E+1. If the file is not found, 0 is stored in the right half of E+1 and the error return is taken. If the device is a multiple-directory device (e.g., disk) and the file is found, but is read protected (see File Protection in this chapter), 2 is stored in the right half of location E+1 and the error return is taken. Otherwise, location E+1 through E+3 are filled by the Monitor with the following data concerning the file, and the normal return is taken.

(1) The left half of location E+1 is set to the filename extension.

(2) If the device is a multiple-directory device, bits 24 through 35 of location E+1 are set to the date (in the format of DAYTIME programmed operator) that the file was last referenced.

If the device is a single-directory device, the right half of location E+1 is set to the device block number of the first block of the file.

(3) If the device is a multiple-directory device, bits 0 through 8 of location E+2 are set to the file protection. (See "File Protection," this chapter.)

(4) Bits 9 through 12 of location E+2 are set to the data mode in which the file was written.

(5) Bits 13 through 23 of location E+2 are set to the time, in minutes, and bits 24 through 35 of location E+2 are set to the date (in the format of the DAYTIME programmed operator) of the file's creation, i.e., of the last ENTER or RENAME programmed operator.

(6) If the device is a multiple-directory device, the left half of location E+3 is set to the negative of the number of words in the file, and the right half is unchanged . If the file contains more than $2^{17}$ words, then the left half contains the positive number of 128-word blocks in the file.

If the device is a single-directory device, location E+3 is used only for SAVed files (see Chapter 3), and contains the IOWD of the core image, i.e., the left half is the negative word length of the file and the right half is the core address of the first word minus 1.

b. ENTER D,E

```
        error return
        normal return
        :
        :
E:      SIXBIT/file/                ;filename, 1 through 6 characters.
        SIXBIT/ext/                 ;filename, extension, 0 through 3 characters.
        EXP<TIME>B23+DATE
        XWD project number, programmer number.
```

ENTER selects a file for output on channel D. If no device is associated with channel D, 7 is stored in bits 33 through 35 of location E+1 and the error return is taken. If the output side of channel D is not closed (see CLOSE in this chapter), it is now closed. The input side of channel D is not affected. If the device does not have a directory, the normal return is now taken.

If the device has multiple directories, e.g., disk, the Monitor searches the master file directory of the device for the user's file directory whose name is in location E+3 and whose extension is UFD. If E+3 contains 0, the project-programmer pair of the current job is used as the name of the user's file directory. If this file is not found in the master file directory, 1 is stored in bits 33 through 35 of location E+1, and the error return is taken. If the filename in location E is 0, 0 is stored in bits 33 through 35 of location E+1, and the error return is taken. The user's file directory, or the device file directory in the case of a single-directory device, such as DECtape, is searched for the file whose name is in location E and whose extension is in the left half of location E+1.

If the device is a multiple-directory device and the file is found but is being written or renamed, 3 is stored in bits 33 through 35 of location E+1, and the error return is taken. If the file is write protected (See "File Protection", this chapter), 2 is stored in bits 33 through 35 of location E+1, and the error return is taken.

If the file is found, and is not being written or renamed and is not write protected, then the file is deleted, or marked for later deletion after all read references are completed, and the storage space on the device is recovered.

The Monitor then makes the file entry by recording the following information concerning the file and takes the normal return.

(1) The filename is taken from location E.

(2) The filename extension is taken from the left half of location E+1.

(3) If the device is a multiple-directory device, then

(a) the current date is taken as the date of last reference;

(b) the file protection key is set to 055 (see "File Protection," this chapter);

(c) the current data mode is taken as the mode in which the file is to be written;

(d) the project number of the current job is taken as the file owner's project number; and

(e) if bits 13 through 35 of location E+2 are nonzero, bits 13 through 23 are taken as the time of creation, in minutes, and bits 24 through 35 are taken as the date of creation (in the format of the DAYTIME programmed operator) of the file. Otherwise, the current time and date are used.

If the device is a single-directory device, then, if bits 24 through 35 of location E+2 are nonzero, they are taken as the date of creation; otherwise, the current date is used.

4.4.2.5 File Protection - File protection on nondirectory and single-directory devices is obtained by use of the ASSIGN command (see Chapter 2). Multiple-directory devices have a master file directory for the device which contains entries for each user's file directory. File selection (see LOOKUP and ENTER in this chapter) requires specification ~~of the name of a user's file directory~~ of a user to locate his file directory and a filename within ~~that~~ his directory. This permits each user to access all files on the device, and necessitates a file protection scheme to prevent unauthorized references. For this purpose users are divided into three categories:

a. The file owner is the user whose project-programmer pair is the same as that in the NAME field of the user's file directory in which the file is entered.

b. Project members are users whose project number is the same as that of the file owner.

c. All other users.

There are three types of protection against each of the three categories of users.

a. Protection-protection - the protection cannot be altered

b. Read protection - the file may not be read.

c. Write protection - the file may not be rewritten, RENAMEd, or deleted.

The file protection key, shown in the following figure, is a set of nine bits which specify the three types of protection for each of the categories of users. (Also see Section 5.8.2.4, "Protection".)

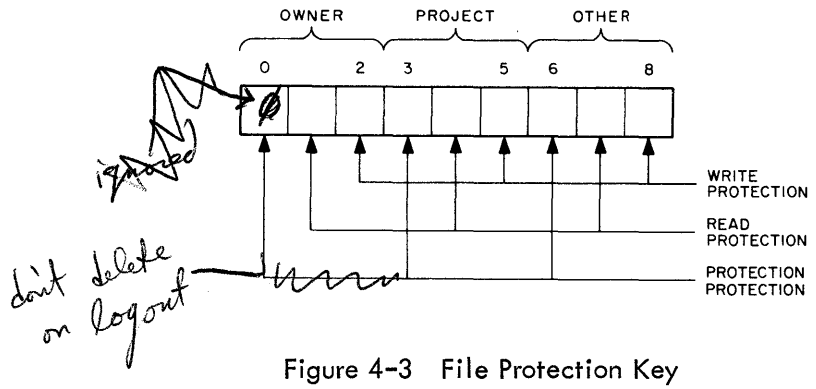



Figure 4-3 File Protection Key

When a file is created by an ENTER programmed operator, the file protection key is set to 055, indicating that the file is protection-protected and write-protected against all users except the owner. The protection key is returned by the LOOKUP D,E programmed operator in bits 0 through 8 of location E+2. It can be changed by the RENAME programmed operator. The owner's protection-protection and read-protection bits are ignored by the Monitor, thereby preventing a file from becoming inaccessible to everyone. However, the LOGIN CUSP sets the protection-protection bit if a user indicates he wishes to selectively protect his file for future logouts. This feature is handled completely by the LOGOUT CUSP.

a. RENAME D,E

```
        error return
        normal return
            .
            .
            .
E:      SIXBIT/file/                    ;filename, 1 through 6 characters.
        SIXBIT/ext/                     ;filename extension, 0 through 3 characters.
        EXP <PROT>B8+<TIME>B23+DATE
        XWD project number, programmer number.
```

The RENAME programmed operator (operation code 055) is used to alter the filename, the filename extension, and the file protection key, or to delete a file associated with channel D on a directory device.

If no device is associated with channel D, 7 is stored in bits 33 through 35 of location E+1, and the error return is taken. If the device is a nondirectory device, the normal return is taken. If no file is currently selected on channel D, 5 is stored in bits 33 through 35 of location E+1, and the error return is taken.

If the device has multiple directories, e.g., disks, the Monitor searches the master file directory of the device for the user's file directory whose name is in location E+3 and whose extension is UFD. If E+3 contains 0, the project-programmer pair of the current job is used as the name of the user's file directory. If this file is not found in the master file directory, 1 is stored in bits 33 through 35 of location E+1, and the error return is taken. The user's file directory, or the device file directory in the case of a single-directory device, is searched for the file currently selected on channel D. If the file is not found, 0 is stored in bits 33 through 35 of location E+1, and the error return is taken.

If the device is a multiple-directory device and the file is found, but is being written or renamed, 3 is stored in bits 33 through 35 of location E+1, and the error return is taken. If the file is owner write-protected or if the protection key is being modified, i.e., bits 0 through 8 of location E+2 differ from the current protection key, and the file is owner protection-protected, 2 is stored in bits 33 through 35 of location E+1, and the error return is taken.

If the new filename in location E is 0, the file is deleted, or marked for deletion, after all read references are completed, and the normal return is taken. If the filename in location E and the filename extension in the left half of location E+1 are the same as the current filename and filename extension, respectively, the protection key is set to the contents of bits 0 through 8 of location E+2, and the normal return is taken.

If the new filename in location E and/or the filename extension in the left half of location E+1 differ from the current filename and/or filename extension, the user's file directory (or the device directory) is searched for the new filename and extension, as in LOOKUP. If a match is found, 4 is stored in bits 33 through 35 of location E+1, and the error return is taken. If no match is found, the file is changed to the new name in location E, the filename extension is changed to the new filename extension in the left half of location E+1, the protection key is set to the contents of bits 0 through 8 of location E+2, the access date is set to the current date, and the normal return is taken.

#### 4.4.2.6 Examples

General Device Initialization

```
INIDEV:     0                       ;JSR HERE
            INIT 3, 14              ;BINARY MODE, CHANNEL 3
            SIXBIT/DTA5/            ;DEVICE DECTAPE UNIT 5
            XWD OBUF, IBUF          ;BOTH INPUT AND OUTPUT
            JRST NOTAVL             ;WHERE TO GO IF DTA5 IS BUSY
;FROM HERE DOWN IS OPTIONAL DEPENDING ON THE DEVICE AND PROGRAM
;REQUIREMENTS
            MOVE 0, JOBFF
            MOVEM 0, SV JBFF        ;SAVE THE FIRST ADDRESS OF THE BUFFER
                                    ;RING IN CASE THE SPACE MUST BE
                                    ;RECLAIMED.
```

```
        INBUF 3,4           ;SET UP 4 INPUT BUFFERS
        OUTBUF 3,1          ;SET UP 1 OUTPUT BUFFER
        LOOKUP 3, INNAM     ;INITIALIZE AN INPUT FILE
        JRST NOTFND         ;WHERE TO GO IF THE INPUT FILE NAME IS
                            ;NOT IN THE DIRECTORY
        ENTER 3, OUTNAME    ;INITIALIZE AN OUTPUT FILE
        JRST NOROOM         ;WHERE TO GO IF THERE IS NO ROOM IN
                            ;THE DIRECTORY FOR A NEW FILE NAME.
        JRST@INIDEV         ;RETURN TO MAIN SEQUENCE
OBUF:   BLOCK 3             ;SPACE FOR OUTPUT BUFFER HEADER
IBUF:   BLOCK 3             ;SPACE FOR INPUT BUFFER HEADER
INNAM:  SIXBIT/NAME/        ;FILE NAME
        SIXBIT/EXT/         ;FILE NAME EXTENSION (OPTIONALLY 0),
                            ;RIGHT HALF WORD RECEIVES THE
                            ;FIRST BLOCK NUMBER
        0                   ;RECEIVES THE DATE
        0                   ;UNUSED FOR NONDUMP I/O
OUTNAM: SIXBIT/NAME/        ;SAME INFORMATION AS IN INNAME
        SIXBIT/EXT/
        0
        0
```

### 4.4.3 Data Transmission

The programmed operators

| | | |
|---|---|---|
| INPUT D,E | and | IN D,E<br>normal return<br>error return |

transmit data from the file selected on channel D to the user's core area. The programmed operators

| | | |
|---|---|---|
| OUTPUT D,E | and | OUT D,E<br>normal return<br>error return |

transmit data from the user's core area to the file selected on channel D.

If no OPEN or INIT operator has been performed on channel D, the Monitor stops the job and prints

I/O TO UNASSIGNED CHANNEL AT USER LOC addr

where addr is the location of the IN, INPUT, OUT, or OUTPUT programmed operator, on the user's console leaving the console in Monitor mode. If the device is a multiple-directory device and no file is selected on channel D, bit 18 of the file status is set to 1, and control returns to the user's program. Control always returns to the location immediately following an INPUT (operation code 066) and an OUTPUT (operation code 067). A check of the file status for end-of-file and error conditions must then be made by another programmed operator. Control returns to the location immediately following an IN (operation code 056) and an OUT (operation code 057), if no end-of-file or error condition

exists, i.e., if bits 18 through 22 of the file status are all 0. Otherwise, control returns to the second location following the IN or OUT. Note that IN and OUT UUOs are the only ones in which the error return is a skip and the normal return is not a skip.

4.4.3.1 Unbuffered (Dump) Modes - In data modes 15, 16, and 17, the effective address E of the INPUT, IN, OUTPUT, and OUT programmed operators is the address of the first word of a command list (see Section 4.4.1). Control does not return to the program until the transmission is terminated or an error is detected.

Example -

Dump Output

Dump input is similar to dump output. This routine outputs fixed-length records.

```
DMPINI:   0                       ;JSR HERE TO INITIALIZE A FILE
          INIT 0, 16              ;CHANNEL 0, DUMP MODE
          SIXBIT/MTA2/            ;MAGNETIC TAPE UNIT 2
          0                       ;NO RING BUFFERS
          JRST NOTAVL             ;WHERE TO GO IF UNIT 2 IS BUSY
          JRST@ DMPINI            ;RETURN
DMPOUT:   0                       ;JSR HERE TO OUTPUT THE OUTPUT AREA
          OUTPUT 0,OUTLST         ;SPECIFIES DUMP OUTPUT ACCORDING
                                  ;TO THE LIST AT OUTLIST
          STATZ 0, 740000         ;CHECK ERROR BITS
          CALL [SIXBIT/EXIT/]     ;QUIT IF AN ERROR OCCURS
          JRST @DMPOUT            ;RETURN
DMPDON:   0                       ;JSR HERE TO WRITE AN END OF FILE
          CLOSE 0,                ;WRITE THE END OF FILE
          STATZ 0, 740000         ;CHECK FOR ERROR DURING WRITE
                                  ;END OF FILE OPERATION
          CALL [SIXBIT/EXIT/]     ;QUIT IF ERROR OCCURS
          RELEAS 0,               ;RELINQUISH THE DEVICE
          JRST@DMPDON             ;RETURN
OUTLST:   IOWD BUFSIZ,BUFFER      ;SPECIFIES DUMPING A NUMBER OF
                                  ;WORDS EQUAL TO BUFSIZ, STARTING
                                  ;AT LOCATION BUFFER
          0                       ;SPECIFIES THE END OF THE COMMAND
                                  ;LIST
BUFFER:   BLOCK BUFSIZ            ;OUTPUT BUFFER, MUST BE CLEARED
                                  ;AND FILLED BY THE MAIN PROGRAM
```

4.4.3.2 Buffered Modes - In data modes 0, 1, 10, 13, and 14 the effective address E of the INPUT, IN, OUTPUT, and OUT programmed operators may be used to alter the normal sequence of buffer reference. If E is 0, the address of the next buffer is obtained from the right half of the second word of the current buffer. If E is nonzero, it is the address of the second word of the next buffer to be referenced. The buffer pointed to by E can be in an entirely separate ring from the present buffer. Once a new buffer location is established, the following buffers are taken from the ring started at E.

a. Input - If no input buffer ring is established when the first INPUT or IN is executed, a 2-buffer ring is set up. (See INBUF, Section 4.4.2.3.)

Buffered input may be performed synchronously or asynchronously at the option of the user. If bit 30 of the file status is 1, each INPUT and IN programmed operator

(1) Clears the use bit in the second word of the buffer whose address is in the right half of the first word of the buffer header, thereby making it available for refilling by the Monitor;

(2) Advances to the next buffer by moving the contents of the second word of the current buffer to the right half of the first word of the 3-word buffer header;

(3) If an end-of-file or an error condition exists, control is returned to the user's program. Otherwise, the Monitor starts the device which fills the buffer and stops transmission;

(4) Computes the number of bytes in the buffer from the number of words in the buffer (right half of the first data word of the buffer) and the data mode, and stores the result in the third word of the buffer header;

(5) Sets the position and address fields of the byte pointer in the second word of the buffer header, so that the first data byte is obtained by an ILDB instruction; and

(6) Returns control to the user's program.

Thus, in synchronous mode, the position of a device, such as magnetic tape, relative to the current data is easily determined. The asynchronous input mode differs in that once a device is started, successive buffers in the ring are filled at the interrupt level without stopping transmission until a buffer whose use bit is 1 is encountered. Control returns to the user's program after the first buffer is filled. The position of the device relative to the data currently being processed by the user's program depends on the number of buffers in the ring and when the device was last stopped.

Example -

General Subroutine to Input One Character

```
GETCHR:   0                       ;JSR HERE AND STORE PC
GETCNT:   SOSG   IBUF+2           ;DECREMENT THE BYTE COUNT
          JRST   GETBUF           ;BUFFER IS EMPTY (OR FIRST CALL AFTER
                                  ;INIT

GETNXT:   ILDB AC, IBUF+1         ;GET NEXT CHAR FROM BUFFER
          JMPN AC, @GETCHR        ;RETURN TO CALLER IF NOT NULL CHAR[1]
          JRST   GETCNT           ;IGNORE NULL AND GET NEXT CHAR
```

[1]For some devices in ASCII mode, the item count provided will always be a multiple of five characters. Since the last word of a buffer may be partially full, user programs which rely upon the item count should always ignore null characters.

```
GETBUF:   IN      3,          ;CALL MONITOR TO REFILL THIS BUFFER
          JRST    GETNXT      ;RETURN HERE WHEN NEXT BUFFER IS
                              ;FULL (PROBABLY IMMEDIATELY)
          JRST    ENDTST      ;RETURN HERE ONLY IF ERROR OR EOF

ENDTST:   STATZ 3, 740000     ;CHECK FOUR ERROR BITS FIRST
          JRST    INERR       ;WHERE TO GO ON AN ERROR
          JRST    ENDFIL      ;WHERE TO GO ON AN END OF FILE
```

b. Output - If no output buffer ring has been established, i.e., if the first word of the buffer header is 0, when the first OUT or OUTPUT is executed, a 2-buffer ring is set up (see OUTBUF, this chapter). If the ring use bit (bit 0 of the first word of the buffer header) is 1, it is set to 0, the current buffer is cleared to all 0s, and the position and address fields of the buffer byte pointer (the second word of the buffer header) are set so that the first byte is properly stored by an IDPB instruction. The byte count (the third word of the buffer header) is set to the maximum of bytes that may be stored in the buffer, and control is returned to the user's program. Thus, the first OUT or OUTPUT initializes the buffer header and the first buffer, but does not result in data transmission.

If the ring use bit is 0 and bit 31 of the file status is 0, the number of words in the buffer is computed from the address field of the buffer byte pointer (the second word of the buffer header) and the buffer pointer (the first word of the buffer header), and the result is stored in the right half of the first data word of the buffer. If bit 31 of the file status is 1, it is assumed that the user has already set the word count in the right half of the first data word. The buffer use bit (bit 0 of the second word of the buffer) is set to 1, indicating that the buffer contains data to be transmitted to the device. If the device is not currently active, i.e., not receiving data, it is started. The buffer header is advanced to the next buffer by setting the buffer pointer in the first word of the buffer header. If the buffer use bit of the new buffer is 1, the job is put into a wait state until the buffer is emptied at the interrupt level. The buffer is then cleared to all 0s, the buffer byte pointer and byte count are initialized in the buffer header, and control is returned to the user's program.

Example -

General Subroutine to Output One Character

```
PUTCHR:   0                   ;JSR HERE AND STORE PC
          SOSG    OBUF+2      ;INCREMENT BYTE COUNT
          JRST    PUTBUF      ;NO MORE ROOM (OR FIRST CALL AFTER INIT)

PUTNXT:   IDPB AC, OBUF+1     ;STORE THIS CHARACTER
          JRST    @PUTCHR     ;AND RETURN TO CALLER

PUTBUF:   OUT     3,          ;CALL MONITOR TO EMPTY THIS BUFFER
          JRST    PUTNXT      ;RETURN HERE WHEN NEXT BUFFER IS
                              ;EMPTY (PROBABLY IMMEDIATELY)
          JRST    OUTERR      ;RETURN HERE ONLY IF OUTPUT ERROR

OUTERR:   GETSTS  3,AC        ;GET THE ERROR STATUS TO LOOK AT
            .
            .
            .
```

### 4.4.4 Status Checking and Setting

The file status (see Table 4-4) is manipulated by the GETSTS (operation code 062), STATZ (operation code 063), STATO (operation code 061) and SETSTS (operation code 060) programmed operators. In each case the accumulator field of the instruction selects a data channel. If no device is associated with the specified data channel, the Monitor stops the job and prints,

I/O TO UNASSIGNED CHANNEL AT USER LOC addr

where addr is the location of the GETSTS, STATZ, STATO, or SETSTS programmed operator, on the user's console leaving the console in Monitor mode.

GETSTS D,E stores the file status of data channel D in the right half and 0 in the left half of location E.

STATZ D,E skips, if all file status bits selected by the effective address E are 0.

STATO D,E skips, if any file status bit selected by the effective address E is 1.

SETSTS D,E waits until the device on channel D stops transmitting data and replaces the current file status, except bit 23, with the effective address E. If the new data mode, indicated in the right four bits of E, is not legal for the device, the job is stopped and the Monitor prints

ILL DEVICE DATA MODE FOR DEVICE dev AT USER addr

where dev is the physical name of the device and addr is the location of the SETSTS operator, leaving the console in Monitor mode. If the user program changes the data mode, it must also change the byte size for the byte pointer in the input buffer header (if any) and the byte size and item count in the output buffer header (if any). Changing the output item count should be done using the count already placed there by the Monitor and dividing or multipling by the appropriate conversion factor, rather than assuming the length of a buffer.

### 4.4.5 Terminating A File (CLOSE)

File transmission is terminated by the CLOSE D,N (operation code 070) programmed operator. If no device is associated with channel D or if bits 34 and 35 of the instruction are both 1, control returns to the user's program immediately.

If bit 34 is 0 and the input side of data channel D is open, it is now closed. In data modes 15, 16, and 17, the effect is to execute a device dependent function and clear the end-of-file flag, bit 22 of the file status. Data modes 0, 1, 10, 13, and 14 have the additional effect, if an input buffer ring exists, of setting the ring use bit (bit 0 of the first word of the buffer header) to 1, setting the buffer byte count (the third word of the buffer header) to 0 and setting the buffer use bit (bit 0 of the second word of the buffer) of each buffer to 0.

If bit 35 of the instruction is 0 and the output side of channel D is open, it is now closed. In data modes 15, 16, and 17, the effect is to execute a device dependent function. In data modes 0, 1, 10, 13, and 14, if a buffer ring exists, all buffers that have not yet been transmitted to the device

are now written, device dependent functions performed, the ring use bit is set to 1, the buffer byte count is set to 0, and control returns to the user after transmission is complete.

Example:

Terminating A File

```
DROPDV:   0                    ;JSR HERE
          CLOSE 3,             ;WRITE END OF FILE AND TERMINATE
                               ;INPUT
          STATZ 3, 740000      ;RECHECK FINAL ERROR BITS
          JRST OUTERR          ;ERROR DURING CLOSE
          RELEAS 3,            ;RELINQUISH THE USE OF THE
                               ;DEVICE, WRITE OUT THE DIRECTORY
          MOVE 0, SVJBFF
          MOVEM 0, JOBFF       ;RECLAIM THE BUFFER SPACE
          JRST @ DROPDV        ;RETURN TO MAIN SEQUENCE
```

### 4.4.6 Synchronization of Buffered I/O (CALL D, [SIXBIT/WAIT/]

In some instances, such as recovery from transmission errors, it is desirable to delay until a device completes its input/output activities. The programmed operators,

CALL D, [SIXBIT/WAIT/] and CALLI D,10

return control to the user's program when all data transfers on channel D have finished. This UUO does not wait for a Magtape spacing operation, since no data transfer is in progress. An MTAPE D, 0 (see Section 5.7.2) should be used to wait for spacing and I/O activity to finish on Magtape. If no device is associated with data channel D, control returns immediately. After the device is stopped, the position of the device relative to the data currently being processed by the user's program can be determined by the buffer use bits.

### 4.4.7 Relinquishing A Device (RELEASE)

When all transmission between the user's program and a device is finished, the program must relinquish the device by performing a

RELEASE D,

RELEASE (operation code 071) returns control immediately, if no device is associated with data channel D. Otherwise, both input and output sides of data channel D are CLOSEd and the correspondence between channel D and the device, which was established by the INIT or OPEN programmed operators, is terminated. If the device is neither associated with another data channel nor assigned (see ASSIGN, Chapter 2) by command, it is returned to the Monitor's pool of available facilities. Control is returned to the user's program.

## 4.5 CORE CONTROL (CALL AC, [SIXBIT/CORE/])

| CALL AC, [SIXBIT/CORE/] | CALLI AC,11 |
|---|---|
| error return | error return |
| normal return | normal return |

These programmed operators provide a user program with the ability to expand and contract its core size as its memory requirements change. Accumulator AC should contain the desired highest relative address. The Monitor will set JOBREL to this new value before returning to the user, provided that the request can be satisfied. If AC contains 0, the number of free 1024-word blocks is returned right-justified in AC, and the error return is taken. If core is being increased, the error return is taken, and the current allocation remains in effect if the request cannot be satisfied. Otherwise, core is appended to or removed from the top of the user's current core area, and the normal return is taken. In all cases the number of free 1024-word blocks is returned right-justified in AC.

# CHAPTER 5
# DEVICE DEPENDENT FUNCTIONS

This chapter explains the unique features of each standard I/O device.[1] All devices accept the programmed operators explained in Chapter 4 unless otherwise indicated. Buffer sizes are given in octal and include two bookkeeping words. Table 5-1 is a summary of the characteristics of all devices.

Table 5-1
Device Summary

| Physical Name | Name | Hardware Type Number | Prog. Op. | Data Modes | Buffer[2] Size (octal) |
|---|---|---|---|---|---|
| CTY | Console Teletype | 626 Models 33,35, 37 | INPUT, IN<br>OUTPUT, OUT | A, AL | 23 |
| TTY0, TTY1 ..., TTY77 | Teletype | 630,680,or DC10 | INPUT, IN<br>OUTPUT, OUT, TTCALL | A, AL | 23 |
| PTY | Pseudo-Tele-type | None | INPUT, IN, OUTPUT, OUT | A, AL | 23 |
| PTR | Paper Tape Reader | 760 | INPUT, IN | A, AL, IB, B, I | 43 |
| PTP | Paper Tape Punch | 761 | OUTPUT, OUT | A, AL, IB, B, I | 43 |
| PLT | Plotter | XY 10 | OUTPUT, OUT | A, AL, I, B, IB | 46 |
| LPT or LPT0, ..., LPT7 | Line Printer | 646, LP10 | OUTPUT, OUT | A, AL | 34 |
| CDR | Card Reader | 461, CR10 | INPUT, IN | A, AL, B, I | 36 |
| CDP | Card Punch | CP 10 | OUTPUT, OUT | A, AL, IB, B | 34 |
| DTA0, DTA1, ..., DTA7 | DECtape | 551/555, TD10/TD55 | INPUT, IN<br>OUTPUT, OUT<br>LOOKUP<br>ENTER<br>MTAPE<br>USETO<br>USETI<br>UGETF<br>CALL [SIXBIT/UTPCLR/] | A, AL, IB, B, I, DR, D | 202 |

[1]The user may determine the physical characteristics associated with a logical device name by executing a DEVCHR UUO. See 5.11.

[2]Buffer sizes are subject to change and should be calculated rather than assumed by user programs. A dummy INBUF or OUTBUF may be employed for this purpose.

Table 5-1 (Cont)
Device Summary

| Physical Name | Name | Hardware Type Number | Prog. Op. | Data Modes | Buffer Size (octal) |
|---|---|---|---|---|---|
| MTA0, MTA1, ..., MTA7 | Magnetic Tape | 516, TM10, TU20, TU79 | INPUT, IN<br>OUTPUT, OUT<br>MTAPE | A, AL, IB, B, I, DR, D | 203 |
| DSK | Disk | RC10 | INPUT, IN<br>OUTPUT, OUT<br>LOOKUP<br>ENTER<br>RENAME<br>USETO | A, AL, I, IB, B, DR, D | 203 |
| DIS | Display | 30, 340 | INPUT<br>OUTPUT | ID | Dump only |

## 5.1 TELETYPE

Device Name - TTY0, TTY1, ..., TTY76, TTY77, CTY

Line number n of the Type 630 Data Communications System, Data Line Scanner DC10, PDP-8 680 System, or PDP-8 68i System is referred to as TTYn. The console Teletype is CTY. The Time-Sharing Monitor automatically gives the logical name, TTY, to the user's console whenever a job is initialized.

Teletype device names are assigned dynamically. For interconsole communication by program, it is necessary for one of the two users to type DEASSIGN TTY in order to make his Teletype available to the other user's program as an output or input device. Typing ASSIGN TTYn is the only way to re-assign a Teletype that has been deassigned. Also see TALK command, Section 2.4.6.

Buffer Size - $25_8$ words.

Two choices of Teletype routines are provided, a newer, full duplex software routine and an older, half duplex software routine. Use of the full duplex software is encouraged.

With a full duplex Teletype service, the two functions of a console, typein and typeout, are handled independently and need not be handled in the strict sense of output first and then input. For example, if two operations are desired from PIP, the request for the second operation can be typed before receiving the asterisk after the completion of the first. The echo of characters typed in will disappear since the keyboard and the printing operations are independent. To stop output that is not wanted, a "Control O" is typed. Also, the command "Control C" will not stop a program instantly. Rather, the Control C will be delayed until the program requests input from the keyboard, and then the program will be stopped. When a program must be stopped instantly, as when it gets into a loop, Control C typed twice will stop the program.

Programs waiting for Teletype output will be awakened eight characters before the output buffer is empty, causing them to be swapped in sooner and preventing pauses in typing. Programs waiting for Teletype input will be awakened ten characters before the input buffer is filled, thus reducing the probability of lost typein.

### 5.1.1 Data Modes

#### 5.1.1.1 Full-Duplex Software A(ASCII) and AL(ASCII Line)

The input handling of all control characters is as follows:
(All are passed to program except as noted below).

| | | |
|---|---|---|
| 000 | NULL | Ignored on input, suppressed on output. |
| 001 | ↑A | Echoes as ↑A. Passed to program. |
| 002 | ↑B | Complements switch controlling echoing, not passed to program. Used on local-copy dataphones and TWX's. |
| 003 | ↑C | The Teletype mode is switched to Monitor mode the next time input is requested by the program. Two successive ↑C's cause the mode to be switched to Monitor mode immediately. |
| 004 | ↑D (EOT) | 004 Passed to program. Not echoed, so typing in a "Control D" (EOT) will not cause a full duplex dataphone to hang up. |
| 005 | ↑E (WRU) | No special action. |
| 006 | ↑F | Complements switch controlling translation of lower case letters to upper case. Used when lower case input is desired to programs. Not sent to program, but program can sense the state of this switch by the TTCALL UUO. |
| 007 | ↑G (Bell) | 007 Passed to program, and is a break character. |
| 010 | ↑H (Back-space) | Acts as a RUBOUT, unless either DDT mode or full character set mode is true, or the ↑F switch is on. In these cases, 010 is sent to the program. |
| 011 | ↑I (TAB) | 011 Passed to program. Echoed as spaces if Teletype is a model 33 (determined by ↑P switch). Spaces are not passed to program. |
| 012 | ↑J (Line-feed) | Is a break character. No other special action. |
| 013 | ↑K (Vertical Tab) | 013 Passed to program. Echoes as four linefeeds, if a model 33. Is a break character. Linefeeds are not passed to program. |
| 014 | ↑L (Form) | 014 Passed to program. Echoes as 8 linefeeds on a 33. Is a break character. Linefeeds are not passed to program. |
| 015 | ↑M (Carriage Return) | If Teletype is in paper-tape input mode, 015 is simply passed to program. Otherwise, supplies a linefeed echo, and is passed to program as a CR and LF, and is a break character (due to LF). |

| | | |
|---|---|---|
| 016 | ↑N | No special action. |
| 017 | ↑O | Suppresses output until an INPUT, or an INIT or OPEN UUO occurs. Not passed to program. Typed by as ↑O followed by carriage return-linefeed. |
| 020 | ↑P | Does not appear in the input buffer. Some Teletype units (usually Models 35 and 37) have horizontal tab, vertical tab, and form feed mechanisms while other units (usually Model 33s) do not. The Monitor assumes that all Teletype units in the system either do or do not have these mechanisms depending upon how the system was built (System Builder). If the user finds that his particular Teletype unit is different from the Monitor's assumption, he should type ↑P. Otherwise, tabs will not be printed at all or spaces will be substituted for a tab depending upon the Monitor's assumption. Alternate uses of ↑P simulate hardware tabs with multiple spaces on and off. |
| 021 | ↑Q (XON) | Starts paper-tape-mode, as described above. Passed to program. |
| 022 | ↑R (TAPE) | No special action. |
| 023 | ↑S (XOFF) | Ends paper-tape mode, as described above. 023 is passed to program. |
| 024 | ↑T (NO TAPE) | No special action. |
| 025 | ↑U | Deletes input line back to last break character. Typed back as ↑U followed by carriage return-linefeed. |
| 026 | ↑V | No special action. |
| 027 | ↑W | No special action. |
| 030 | ↑X | No special action. |
| 031 | ↑Y | No special action. |
| 032 | ↑Z | Acts as end-of-file on Teletype input. Echoes as ↑Z followed by carriage-return, linefeed. Is a break character. Appears in buffer as 032. |
| 033 | ↑[ (ESC) | This is the ASCII altmode these days, but is translated to 175 before being passed to the program, unless in full character set mode (bit 29 in INIT). 1/5 is the 1963 altmode. Echoes as a dollar sign. Always, is a break character. |
| 034 | ↑\ | No special action. |
| 035 | ↑] | No special action. |
| 036 | ↑↑ | No special action. |
| 037 | ↑← | No special action. |
| 040-137 | | Printing characters, no special action. |
| 140-174 | | "Lower case" ASCII. Translated to upper case, unless ↑F switch is set. Echoes as upper case if translated to upper case. |
| 175 and 176 | | Old versions of altmode. See description of "ESC" (033). |
| 177 | | RUBOUT or DELETE:<br>A) Completely ignored if in papertape mode (XON).<br>B) Is a break character, passed to program if either DDTmode or fullcharacter-set mode is true.<br>C) Otherwise (ordinary case) causes a character to be deleted for each rubout types. All the characters deleted are echoed between a single pair of backslashes. If no characters remain to be deleted, echoes as a carriage-return, linefeed. |

On output, all characters are typed just as they appear in the output buffer with the exceptions, TAB, VT, and FORM, which are processed the same as on type in.

5.1.1.2 Half-Duplex Software A(ASCII) - If, during output operations, an echo-check failure occurs (the transmitted character was not the same as the intended character), the I/O routine suspends output until the user types the next character. If that character is ↑C, the console is placed in Monitor mode immediately. If it is ↑O, all Teletype output buffers that are currently full are ignored, thus cutting the output short. All other characters cause the service routines to continue output. The user may cause a deliberate echo check by typing in while typeout is in progress. For example, to return to Monitor control mode while typeout is in progress, the user must type any character ("X", for example) until an echo check occurs and output is suspended; then and only then he types ↑C.

The buffer is terminated when it fills up or when the user types ↑Z.

5.1.1.3 Half-Duplex Software AL(ASCII Line) - Same as ASCII mode (usually preferred) with the addition that the input buffer is terminated by a CR/LF pair, FF, VT, or ALTMODE.

5.1.2 DDT Submode

To allow a user's program and the DDT debugging program to use the same Teletype without interfering with one another, the Teletype service routine provides the DDT submode. This mode does not affect the Teletype status if it is initialized with the INIT operator. It is not necessary to use INIT in order to do I/O in the DDT submode. I/O in DDT mode is always to the user's Teletype and not to any other device.

In the DDT submode, the user's program is responsible for its own buffering. Input is usually one character at a time, but if the typist types characters faster than they are processed, the Teletype service routine supplies bufferfuls of characters at a time.

To input characters in DDT mode, use the sequence

```
MOVEI AC,BUF
CALL AC, [SIXBIT/DDTIN/]
```

BUF is the first address of a 21-word block in the user's area. The DDTIN operator delays, if necessary, until one character is typed in. Then all characters (in 7-bit packed format) typed in since the previous occurrence of DDTIN are moved to the user's area in locations BUF, BUF+1, etc. The character string is always terminated by a null character (000). RUBOUTs are not processed by the service routine but are passed on to the user. The special control characters ↑O and ↑U have no effect. Other characters are processed as in ASCII mode.

To perform output in DDT mode, use the sequence

```
MOVEI AC,BUF
CALL AC, [SIXBIT/DDTOUT/]
```

BUF is the first address of a string of packed 7-bit characters terminated by a null (000) character. The Teletype service routine delays until the previous DDTOUT operation is complete, then moves the entire character string into the Monitor, begins to output the string, and restarts the user's program. Character processing is the same as for ASCII mode output.

### 5.1.3 Special Programmed Operator Service

TTCALL UUO is (and will always be) implemented only in the "full duplex scanner service", SCNSRF.

The general form of this UUO is as follows:

```
OPDEF   TTCALL   [51B8]
TTCALL  AC, ADR
```

The AC field describes the particular function desired, and the argument (if any) is contained in ADR. ADR may be an AC or any address in low segment above JOB AREA (137). It may be in high segment for AC fields 1 and 3. The functions are:

| AC Field | Mnemonic | Action |
|---|---|---|
| 0 | INCHRW | Input character and wait |
| 1 | OUTCHR | Output a character |
| 2 | INCHRS | Input character and skip |
| 3 | OUTSTR | Output a string |
| 4 | INCHWL | Input character, wait, line mode |
| 5 | INCHSL | Input character, skip, line mode |
| 6 | GETLIN | Get line characteristics |
| 7 | SETLIN | Set line characteristics |
| 10 | RESCAN | Reset input stream to command |
| 11 | CLRBFI | Clear typein buffer |
| 12 | CLRBFO | Clear typeout buffer |
| 13 | SKPINC | Skips if a character can be input |
| 14 | SKPINL | Skips if a line can be input |
| 15-17 | (Reserved for Expansion) | |

INCHRW  TTCALL 0,ADR

This command inputs a character into location ADR. ADR may be an AC or any other location in the user's low segment. If there is no character yet typed, the program waits for it.

OUTCHR  TTCALL 1,ADR

This command outputs a character to the teletype, from location ADR. Only the low order 7 bits of the contents of ADR are used, the rest need not be zeroes.

If there is no room in the output buffer, the program waits until room is available. ADR may be in high segment.

```
INCHRS      TTCALL  2,ADR
```

This command is similar to INCHRW, except that it skips on a successful return, and does not skip if there is no character in the input buffer, it never puts the job into a wait.

```
TTCALL      2,ADR
JRST        NONE            ;NO TYPEIN
JRST        DONE            ;CHARACTER IN ADR
```

```
OUTSTR      TTCALL  3,ADR
```

This command outputs a string of characters in ASCIZ format:

```
TTCALL      3,MESSAGE

MESSAGE:    ASCIZ           /TYPE THIS OUT/
```

ADR may be in high segment

```
INCHWL      TTCALL  4,ADR
```

This command is the same as INCHRW, except that it decides whether or not to wait on the basis of lines rather than characters, as such, it is the preferred way of inputting characters, since INCHRW causes a swap to occur for each character rather than each line (compare DDT and PIP input, for instance).

```
INCHSL      TTCALL  5,ADR
```

This command is the same as INCHRS, except that its decision whether to skip is made on the basis of lines rather than characters.

```
GETLIN      TTCALL  6,ADR
```

This command takes one argument, from location ADR, and returns one word, also in ADR, the argument is a number, representing a teletype line. If the argument is negative, the line number controlling the program is assumed. If the line number is greater than those defined in the system, a zero answer is returned.

The normal answer format is as follows:

| | |
|---|---|
| Right half of ADR: | The line number. |
| Left half of ADR: | Bits, as follows: |

| Bit | Meaning |
|---|---|
| 0 | Line is a pseudo-teletype. |
| 1 | Line is the CTY. |
| 2 | Line is a display console. |
| 3 | Line is a dataset data line. |

| Bit | Meaning |
|---|---|
| 4 | Line is a dataset control line. |
| 5 | Line is half-duplex. |
| 11 | A line has been typed in by the user |
| 12 | A rubout has been typed. |
| 13 | "Control F" switch is on. |
| 14 | "Control P" switch is on. |
| 15 | "Control B" switch is on. |
| 16 | "Control Q" (paper tape) switch is on. |
| 17 | Line is in a "talk" ring. |

SETLIN TTCALL 7,ADR

This command allows a program to set and clear some of the bits described for GETLIN. They may be changed only for the controlling teletype. The bits which may be modified are bits 13, 14, 15 and 16. Example:

```
SETO   AC, 0
TTCALL 6,AC
TLZ    AC,BIT 13
TLO    AC,BIT 14
TTCALL 7,AC
```

RESCAN TTCALL 14,0

This command is intended for use only by the CCL CUSP. It causes the Input Buffer to be re-scanned from the point where the last command began. Obviously, if it is executed other than before the first input, that command may no longer be in the buffer. ADR is not used, (but is address checked).

CLRBFI TTCALL 11,0

This command causes the Input Buffer to be cleared (as if the user had typed a number of "Control U's"). It is intended to be used when an error has been detected, such that a user probably would not want any commands to be executed which he might have typed ahead.

CLRBFO TTCALL 12,0

This command causes the output buffer to be cleared, as if the user had typed "CONTROL O". It should be used only rarely, since usually one wants to see all output, up to the point of an error. It is included primarily for completeness.

SKPING TTCALL 13,0

This command skips if the user has typed at least one character. It does not skip if no character have been typed, however it never inputs a character. It is useful for a compute based program which wants to occasionally check for input and, if any, go off to another routine (such as FORTRAN Operating System) to actually do the input.

SKPINL TTCALL 14,0

This command is the same as SKPING except that a skip occurs if a line has been typed.

### 5.1.4 Special Status Bits (Full Duplex Software only)

An INIT or OPEN, with bit 28 a one, suppresses echoing on the Teletype. This is useful for LOGIN to eliminate the mask for the password.

### 5.1.5 Paper Tape Input from the Teletype (Full Duplex Software only)

Paper tape input is possible from a Teletype equipped with a paper tape reader, controlled by the XON and XOFF characters. When commanded by the XON character, the Teletype service will read paper tapes, starting and stopping the paper tape as needed and continuing until the XOFF character is read or typed in. While in this mode of operation, any RUBOUTS will be discarded and no free line feeds will be inserted after carriage returns. Also, TABS and FORMFEEDS will not be simulated on Model 33's, to insure output of the reader control characters. In order to use paper tape processing, the Teletype with paper tape reader must be connected by a full duplex connection and only ASCII paper tapes are intended to be used.

The correct operating sequence for reading a paper tape in this way is as follows:

```
.R PIP <RETURN>
*DSK: FILE←TTY: <XON><RETURN><LINEFEED>
THIS IS WHAT IS ON TAPE
MORE OF SAME
LAST LINE
↑Z
*<XOFF>
```

## 5.2 PAPER TAPE READER

Device Mnemonic - PTR

Buffer Size - $43_8$ words

### 5.2.1 Data Modes (Input Only)

> NOTE: To initialize the paper tape reader, the input tape must be threaded through the reading mechanism and the FEED button depressed.

5.2.1.1 A (ASCII) - Blank tape (000), RUBOUT (377), and null characters (200) are ignored. All other characters are truncated to seven bits and appear in the buffer. The physical end of the paper tape

serves as an end-of-file and results in the character 032 ↑Z) appearing in the buffer.

5.2.1.2 AL (ASCII Line) - Character processing is the same as for the A mode. The buffer is terminated by LINE FEED, FORM, or VT.

5.2.1.3 I (Image) - There is no character processing. The buffer is packed with 8-bit characters exactly as read from the input tape. Physical end of tape is the end-of-file indication but does not cause a character to appear in the buffer.

5.2.1.4 IB (Image Binary) - Characters not having the eighth hole punched are ignored. Characters are truncated to six bits and packed six to the word without further processing. This mode is useful for reading binary tapes having arbitrary blocking format.

5.2.1.5 B (Binary) - Checksummed binary data is read in the following format. The right half of the first word of each physical block contains the number of data words that follow and the left contains half a folded checksum. The checksum is formed by adding the data words using 2s complement arithmetic, then splitting the sum into three 12-bit bytes and adding these using 1s complement arithmetic to form a 12-bit checksum. The data error status flag (IODERR) is raised if the checksum miscompares. Because the checksum and word count appear in the input buffer, the maximum block length is 40. The byte pointer, however, is initialized so as not to pick up the word count and checksum word.

Again, physical end of tape is the end-of-file indication but does not result in putting a character in the buffer.

## 5.3 PAPER TAPE PUNCH

Device Mnemonic - PTP

Buffer Size - $43_8$ words

### 5.3.1 Data Modes

5.3.1.1 A (ASCII) - The eighth hole is punched for all characters. Tape-feed without the eighth hole (000) is inserted after form-feed. A rubout is inserted after each vertical or horizontal tab. Null characters (000) appearing in the buffer are not punched.

5.3.1.2 AL (ASCII Line) - The same as A mode. Format control must be performed by the user's program.

5.3.1.3 I (Image) - Eight-bit characters are punched exactly as they appear in the buffer with no additional processing.

5.3.1.4 IB (Image Binary) - Binary words taken from the output buffer are split into six 6-bit bytes and punched with the eighth hole punched in each line. There is no format control or checksumming performed by the I/O routine. Data punched in this mode is read back by the paper tape reader in the IB mode.

5.3.1.5 B (Binary) - Each bufferful of data is punched as one checksummed binary block as described for the paper tape reader. Several blank lines are punched after each bufferful for visual clarity.

### 5.3.2 Special Programmed Operator Service

The first output programmed operator of a file causes about two fanfolds of blank tape to be punched as leader. Following a CLOSE, an additional fanfold of blank tape is punched as trailer. No end-of-file character is punched automatically.

## 5.4 LINE PRINTER

Device Mnemonic - LPT

Buffer Size - $34_8$ words

### 5.4.1 Data Modes

5.4.1.1 A (ASCII) - ASCII characters are transmitted to the line printer exactly as they appear in the buffer. See the PDP-10 System Reference Manual, for a list of the vertical spacing characters.

5.4.1.2 AL (ASCII Line) - This mode is exactly the same as A and is included for programming convenience. All format control must be performed by the user's program; this includes placing a RETURN, LINE-FEED sequence at the end of each line.

### 5.4.2 Special Programmed Operator Service

The first output programmed operator of a file and the CLOSE at the end of a file cause an extra form-feed to be printed to keep files separated.

## 5.5 CARD READER

Device Mnemonic - CDR

Buffer Size - $36_8$ words

### 5.5.1 Data Modes

5.5.1.1 A (ASCII) - All 80 columns of each card are read and translated to 7-bit ASCII code. Blank columns are translated to spaces. At the end of each card a carriage-return/line-feed is appended. A card with the character 12-11-0-1 punched in column 1 is an end-of-file card. Columns 2 through 80 are ignored, and an end-of-file character 032 appears as the last character in the input buffer. The end-of-file button on the card reader has the same effect as the end-of-file card. As many complete cards as can fit are placed in the input buffer, but cards are not split between two buffers. Using the standard-sized buffer, only one card is placed in each buffer.

Cards are normally translated as IBM 026 card codes. If a card containing a 12-0-2-4-6-8 punch in column 1 is encountered, any following cards are translated as 029 codes (see Table 5-2 PDP-10 Card Codes) until the 029 conversion mode is turned off. The 029 mode is turned off either by a RELEASE command or by a card containing a 12-2-4-8 punch in column 1. Columns 2 through 80 of both of these cards are ignored.

5.5.1.2 AL (ASCII Line) - Exactly the same as the A mode.

5.5.1.3 I (Image) - All 12 punches in all 80 columns are packed into the buffer as 12-bit bytes. The first 12-bit byte is column 1. The last word of the buffer contains columns 79 and 80 as the left and middle bytes respectively. The end-of-file card and the end-of-file button are processed the same as in the A mode with the character 0032 appearing in the buffer as the last character of the file. Cards are not split between two buffers.

5.5.1.4 B (Binary) - Card column 1 must contain a 7-9 punch to verify that the card is in binary format. The absence of the 7-9 punch results in raising the IOIMPM (improper mode) flag in the card reader status word. Card column 2 must contain a 12-bit checksum as described for the paper tape reader binary format. Columns 3 through 80 contain binary data, 3 columns per word for up to 26 words. Cards are not split between two buffers. The end-of-file card and the end-of-file button are processed the same as in the A mode with a word containing 003200000000 appearing as the last word in the file.

Table 5-2
PDP-10 Card Codes

| CHAR | PDP10 ASCII | DEC 029 | DEC 026 | CHAR | PDP10 ASCII | DEC 029 | DEC 026 |
|---|---|---|---|---|---|---|---|
| SPACE | 040 | | | @ | 100 | 8 4 | 8 4 |
| ! | 041 | 11 8 2 | 12 8 7 | A | 101 | 12 1 | 12 1 |
| " | 042 | 8 7 | 0 8 5 | B | 102 | 12 2 | 12 2 |
| # | 043 | 8 3 | 0 8 6 | C | 103 | 12 3 | 12 3 |
| $ | 044 | 11 8 3 | 11 8 3 | D | 104 | 12 4 | 12 4 |
| % | 045 | 0 8 4 | 0 8 7 | E | 105 | 12 5 | 12 5 |
| & | 046 | 12 | 11 8 7 | F | 106 | 12 6 | 12 6 |
| ' | 047 | 8 5 | 8 6 | G | 107 | 12 7 | 12 7 |
| ( | 050 | 12 8 5 | 0 8 4 | H | 110 | 12 8 | 12 8 |
| ) | 051 | 11 8 5 | 12 8 4 | I | 111 | 12 9 | 12 9 |
| * | 052 | 11 8 4 | 11 8 4 | J | 112 | 11 1 | 11 1 |
| + | 053 | 12 8 6 | 12 | K | 113 | 11 2 | 11 2 |
| , | 054 | 0 8 3 | 0 8 3 | L | 114 | 11 3 | 11 3 |
| - | 055 | 11 | 11 | M | 115 | 11 4 | 11 4 |
| . | 056 | 12 8 3 | 12 8 3 | N | 116 | 11 5 | 11 5 |
| / | 057 | 0 1 | 0 1 | O | 117 | 11 6 | 11 6 |
| 0 | 060 | 0 | 0 | P | 120 | 11 7 | 11 7 |
| 1 | 061 | 1 | 1 | Q | 121 | 11 8 | 11 8 |
| 2 | 062 | 2 | 2 | R | 122 | 11 9 | 11 9 |
| 3 | 063 | 3 | 3 | S | 123 | 0 2 | 0 2 |
| 4 | 064 | 4 | 4 | T | 124 | 0 3 | 0 3 |
| 5 | 065 | 5 | 5 | U | 125 | 0 4 | 0 4 |
| 6 | 066 | 6 | 6 | V | 126 | 0 5 | 0 5 |
| 7 | 067 | 7 | 7 | W | 127 | 0 6 | 0 6 |
| 8 | 070 | 8 | 8 | X | 130 | 0 7 | 0 7 |
| 9 | 071 | 9 | 9 | Y | 131 | 0 8 | 0 8 |
| : | 072 | 8 2 | 11 8 2 OR 11 0 | Z | 132 | 0 9 | 0 9 |
| ; | 073 | 11 8 6 | 0 8 2 | [ | 133 | 12 8 2 | 11 8 5 |
| < | 074 | 12 8 4 | 12 8 6 | \ | 134 | 11 8 7 | 8 7 |
| = | 075 | 8 6 | 8 3 | ] | 135 | 0 8 2 | 12 8 5 |
| > | 076 | 0 8 6 | 11 8 6 | ↑ | 136 | 12 8 7 | 8 5 |
| ? | 077 | 0 8 7 | 12 8 2 OR 12 0 | ← | 137 | 0 8 5 | 8 2 |

## 5.6 CARD PUNCH

Device Mnemonic - CDP

Buffer Size - $35_8$ words

### 5.6.1 Data Modes

5.6.1.1 A (ASCII) - ASCII characters are converted to card codes and punched (up to 80 characters per card). Tabs are simulated by punching from 1 to 7 blank columns; form-feeds and carriage returns are ignored. Line-feeds cause a card to be punched. All other nontranslatable ASCII characters cause a question mark to be punched. Cards can be split between buffers. Attempting to punch more than 80 columns per card causes the error bit IOBKTL to be raised.

Cards are normally punched with DEC026 card codes. If bit 26 (octal 1000) of the status word is on (from INIT, OPEN, or SETSTS), cards are punched with DEC029 codes. The first card of any file indicates the card code used (12-0-2-4-6-8 punch in column 1 for DEC029 card codes; 12-2-4-8 punch in column 1 for DEC026 card codes).

5.6.1.2 AL (ASCII Line) - The same as A mode.

5.6.1.3 IB (Image Binary) - Up to 26 2/3 data words will be punched in columns 1-80. The buffer set up by the Monitor will only contain room for 26 data words. Image binary will cause exactly one card to be punched for each output. The CLOSE will punch the last partial card, and then punch an EOF card (12-11-0-1 in column 1).

5.6.1.4 B (Binary) - Column 1 will contain the word count in rows 11-2. A 7-9 punch will also be in column 1. Column 2 will contain a checksum; columns 3-80 will contain up to 26 data words, 3 columns per word. Binary will cause exactly one card to be punched for each output. The CLOSE will punch the last partial card, and then punch an EOF card (12-11-0-1 in column 1).

5.6.2 Special Programmed Operator Service

Following a CLOSE, an end-of-file card is punched.

Both the first card of the file (the one that identifies the card code used) and the end-of-file card are laced in columns 2 through 80 for easy identification of files. These laced punches are ignored by the card reader service routine.

## 5.7 DECTAPE

Device Mnemonic - DTA0, DTA1, ..., DTA7

Buffer Size - $202_8$ words

### 5.7.1 Data Modes

5.7.1.1 A (ASCII) - Data is written on DECtape exactly as it appears in the buffer. No processing of checksumming of any kind is performed by the service routine. The self-checking of the DECtape system is sufficient assurance that the data is correct. See the description of DECtape format below for further information concerning blocking of information.

5.7.1.2 AL (ASCII Line) - Same as A.

5.7.1.3 I (Image) - Same as A. Data consists of 36-bit words.

5.7.1.4 IB (Image Binary) - Same as I.

5.7.1.5 B (Binary) - Same as I.

5.7.1.6 DR (Dump Records) - This mode is accepted but actually functions as dump mode 17.

5.7.1.7 D (Dump) - Data is read into or written from anywhere in the user's core area without regard to the standard buffering scheme. Control for read or write operations must be via a command list in core memory. The command list format is as described in Chapter 4, "Unbuffered (Dump) Modes;" any positive number appearing in a command list terminates the list. Dump data is automatically blocked into standard-length DECtape blocks by the DECtape control. Unless the number of data words is an exact multiple of the standard length of a DECtape block ($128_{10}$), after each output programmed operator, the remainder of the last block written is wasted. The input programmed operator must specify the same number of words that the corresponding output programmed operator specified in order to skip over the wasted fractions of blocks.

## 5.7.2 DECtape Block Format

A standard reel of DECtape consists of 578 ($1102_8$) prerecorded blocks each capable of storing 128 ($200_8$) 36-bit words of data. Block numbers which label the blocks for addressing purposes are recorded between blocks. These block numbers run from 0 to $1101_8$. Blocks 0, 1, and 2 are normally not used during time-sharing and are reserved for a bootstrap loader. Block $100_{10}$ ($144_8$) is the directory block which contains the names of all files on the tape and information relating to each file. Blocks $1_{10}$ through $99_{10}$ ($1\text{-}143_8$) and $101_{10}$ through $577_{10}$ ($145\text{-}1101_8$) are usable for data.

If in the process of DECtape I/O, the I/O service routine is requested to use a block number larger than $1101_8$ or smaller than 0, the Monitor sets the Block Too Large flag (bit 21) in the file status and returns.

### 5.7.3 DECtape Directory Format

The directory block (block $100_{10}$) of a DECtape contains directory information for all files on that tape; a maximum of 22 files can be stored on any one DECtape.

| | | |
|---|---|---|
| Words 0 through $82_{10}$ | | The first 83 words of the directory contain "slots," each "slot" representing one of the 577 (blocks 1 through $1101_8$ are represented in these 83 words) blocks on the DECtape. Each slot occupies five bits (seven slots are stored per word) and contains the number of the file ($1\text{-}26_8$) to which the block the slot represents is assigned. |
| Words 83 through $104_{10}$ | | The next 22 words contain the filenames of the 22 files residing on the DECtape. Word 83 contains the filename for file #1, word 84 the filename #2, etc. Filenames are stored in 6-bit code. |
| Words 105 through $126_{10}$ | | The next 22 words contain the extension names and dates of the 22 files, in the same relative order as their filenames above. |
| | Bits 0 through $17_{10}$ | The extension name of the file (in 6-bit code) |
| | Bits 18 through $23_{10}$ | Number of 1K blocks minus 1 needed to load the file (maximum value=53). This information is stored for SAVEd files only. |
| | Bits 24 through $35_{10}$ | The date the file was last updated, according to the formula: ((year-1964)*12+(month-1))*31+day-1 |
| Word $127_{10}$ | | Unused. |

The message

BAD DIRECTORY FOR DEVICE DTAn: EXEC CALLED FROM USER LOC n

is produced whenever any of the following conditions are detected.

a. A parity error while reading the directory block.

b. No "slots" are assigned to the file number of the file.

c. The tape block which may possibly be the first block of the file (i.e., the first block for the file encountered while searching backwards from the directory block) cannot be read.

### 5.7.4 DECtape File Format

A file consists of any number of DECtape blocks. Each block contains:

| | | |
|---|---|---|
| Word 0 | Left half | The link. The link is the block number of the next block in the file. If the link is zero, this block is the last in the file. |

| | | |
|---|---|---|
| | Right half | Bits 18 through 27: The block number of the first block of the file.<br>Bits 28 through 35: A count of the number of words in this block which are used (maximum $177_8$). |
| Words 1 through $177_8$ | | Data packed exactly as the user placed in his buffer or in Dump Mode files, the next 127 words of memory. [1] |

### 5.7.5 Special Programmed Operators Service

Several programmed operators are provided for manipulating DECtape. These allow the user to manipulate block numbers and to handle directories.

In addition to the operators above, INPUT, OUTPUT, CLOSE, and RELEAS have special effects. When performing nondump input operations, the DECtape service routine reads the links in each block to determine the next block to read and when to raise the end-of-file flag.

When an OUTPUT is given, the DECtape service routine examines the left half of the first data word in the output buffer (the word containing the word count in the right half). If this half word contains -1, it is replaced with a 0 before being written out, and the file is thus terminated. If this half word is greater than -1, it is not changed and the service routine uses it as the block number for the next OUTPUT.

Table 5-3
DECtape Programmed Operators

| Programmed Operator | Effect |
|---|---|
| USETI D, E | Sets the DECtape on device channel D to input block E next. Input operations on this DECtape must not be active because otherwise the user has no way of determining which buffer contains block E. |
| USETO D, E | Similar to USETI but sets the output block number USETO waits until the device is inactive before setting up the new output block number. |
| UGETF D, E | Places the number of the first free block of the file in user's location E. |
| ENTER D, E | User's location E, E+1, E+2, and E+3, must be reserved for a directory entry. The DECtape service routine searches the directory for a filename and extension that match the contents of E and the left half of E+1. If no match is found and there is room in the directory, the service routine places the first free block number into the right half of E+1, places the date in E+2 (unless already non-zero), and places the necessary information into the directory. If a match is found, similar actions occur, but the new entry replaces the old. If there is no room in the directory, ENTER returns to the next location. Otherwise, ENTER skips one location. |

[1] The Monitor compresses the user's core image by squeezing out blocks of two or more consecutive zeroes before creating the SAVed files; files with extension .SAV may be read in Dump Mode, but must be re-expanded before being run. The Monitor takes this action after input on a RUN or GET.

Table 5-3 (Cont)
DECtape Programmed Operators

| Programmed Operator | Effect |
|---|---|
| LOOKUP D, E<br>error return | Similar to ENTER but sets up an input file. The contents of E and E+1 are matched against the filenames and extension names in the DECtape directory. If a match is found, information about the file is read from the directory into the appropriate portions of the 4-word block beginning at E. The first block of the file is then found as follows.<br>1. The first 83 words of the DECtape directory are searched in a backwards manner, beginning with the slot immediately prior to the directory block, until the first slot containing the desired file number is found.<br>2. The block associated with this slot is then read in and bits 18 through 27 of the first word of the block (these bits contain the block number of the first block of the file) are checked. If they are equal to the block number of this block, then this block is the first block of the file; if not, then the block with that block number is read as the first block of the file.<br>LOOKUP then skips one location.<br>If no match is found, LOOKUP returns to the user's program at the next location. |
| CALL D, [SIXBIT/UTPCLR/] | UTPCLR clears the directory of the DECtape on device channel D. A cleared directory has zeroes in the first 83 words except in those slots related to blocks 0, 1, 2, and $100_{10}$ and nonexistent blocks 1102 through $1105_8$. Only the directory block (block 100) is affected by UTPCLR; the other blocks are unaffected. This programmed operator does nothing if the device on channel D is not DECtape. |
| RENAME D, E | This programmed operator is used to alter the name and extension of a file or to delete it from the DECtape. Locations E to E+3 are as in LOOKUP and ENTER. To be RENAMEd a file must first be CLOSEd on channel D, in order to identify for the RENAME UUO. RENAME then seeks out this file and enters the information specified in E through E+2 into the retrieval information and proper directory. If the contents of E is zero, RENAME has the effect of deleting the file. The error return is given if the new file name and extension already exist or if neither a LOOKUP nor an ENTER has been done to identify the file to be renamed. |

For both INPUT and OUTPUT, block 100 (the directory) is treated as an exception case. If the user's program gives

USETI D, $144_8$

to read block 100, it is treated as a 1-block file.

The CLOSE operator places a -1 in the left half of the first word in the last output buffer, thus, terminating the file.

The RELEAS operator writes the copy of the directory which is normally kept in core onto block 100, but only if any changes have been made. Certain console commands, such as KJOB or CORE 0, perform an implicit RELEAS of all devices and, thus, write out a changed directory even though the user's program failed to give a RELEAS.

Two other special programmed operators are available: MTAPE D, 1 and MTAPE D, 11. MTAPE D, 1 rewinds the DECtape and moves it into the end zone at the front of the tape. MTAPE D, 11 rewinds and unloads the tape, pulling the tape completely onto the lefthand reel. These commands affect only the physical position of the tape, not the "logical" position. When either is used, the user's job can be swapped out while the DECtape is rewinding; however, the job cannot be swapped out if an INPUT or OUTPUT is done while the tape is rewinding.

### 5.7.6 Special Status Bits

If an attempt is made to write on a unit with the WRITE-LOCK switch on, the improper mode flag (bit 18) is set in the file status word.

5.7.6.1 Special DECtape Status Bits - An INIT or SETSTS to a DECtape with bit 29 ON informs DTASER (the DECtape service routine) that the DECtape is in nonstandard format. This implies that no file-structured operations will be performed on that tape. Blocks will be read or written sequentially; no links will be generated (output) or recognized (input). The first block to be read or written must be set by a USETI or USETO. In Dump Mode, $200_8$ data words per block will be read or written (as opposed to the normal $177_8$ words). No "dead reckoning" will be used on a search for a block number, as the tape may be composed of blocks shorter than 200 words. The ENTER, LOOKUP, and UTPCLR UUOs are treated as no-ops. Block 0 of the tape may not be read or written in Dump Mode if bit 29 is ON, as the data must be read in a forward direction and block 0 normally cannot be read forward.

### 5.7.7 Important Considerations

The DECtape service routine reads the directory from a tape the first time it is required to perform a LOOKUP, ENTER, or UGETF; the directory image remains in core until a new ASSIGN command is executed from the console. To inform the DECtape service routine that a new tape has been mounted on an assigned unit, the user must use an ASSIGN command. The directory from the old tape could be transferred to the new tape, thus destroying the information on that tape unless the user reassigns the DECtape transport every time he mounts a new reel.

## 5.8 MAGNETIC TAPE

Magnetic tape format is industry compatible, 7- or 9-channel 200, 556, and 800 bpi (see description below).

Device Mnemonic - MTA0, MTA1,...,MTA7

Buffer Size - $203_8$ words

### 5.8.1 Data Modes

5.8.1.1 A (ASCII) - Data appears to be written on magnetic tape exactly as it appears in the buffer. No processing or checksumming of any kind is performed by the service routine. The parity checking of the magnetic tape system is sufficient assurance that the data is correct. Normally, all data, both binary and ASCII, is written with odd parity and at 556 bits per inch. A maximum of 200 words per record is standard. The word-count is not written on the tape.

5.8.1.2 AL (ASCII Line) - Same as A.

5.8.1.3 I (Image) - Same as A but data consists of 36-bit words.

5.8.1.4 IB (Image Binary) - Same as I.

5.8.1.5 B (Binary) - Same as I.

5.8.1.6 DR (Dump Records) - Standard fixed length records (128 words is the standard unless installation standard is changed with MONGEN) are read into or written from anywhere in the user's core area without regard to the standard buffering scheme. Control for read or write operations must be via a command list in core memory. The command list format is as described in Chapter 4, "Unbuffered (Dump) Modes." For input operations a new record is read for each word in the command list (except GOTO words); if the record terminates before the command word is satisfied, the service routine reads the next records. If the command word runs out before the record terminates, the remainder of the record is ignored. For each output command word, as many standard length records are written followed by one short record to exactly write all of the words onto the tape.

5.8.1.7 D (Dump) - Variable length records are read into or written from anywhere in the user's core area without regard to the standard buffering scheme. Control for read or write operations must

be via a command list in core memory. The command list format is as described in Chapter 4, "Unbuffered (Dump) Modes." For input operations a new record is read for each word in the command list (except GOTO words); if the record terminates before the command word is satisfied, the service routine skips to the next command word. If the command word runs out before the record terminates, the remainder of the record is ignored. For each output command word, exactly one record is written. See Section 4.4.1.2 for command list format.

### 5.8.2 Magnetic Tape Format

Magnetic tape format can be generally described as unlabelled, industry compatible format. That is, as far as the user is concerned, the tape contains only data records and end-of-file marks which signal the end of the data set or the end of the file. Files are read from and written on the tape in a sequential manner.

An end-of-file mark consists of a record containing a $17_8$ (for 7-channel tapes) or a $23_8$ (for 9-channel tapes). End-of-file marks are used in the following manner.

a. No end-of-file mark precedes the first file on a magtape.

b. An end-of-file mark follows every file.

c. Two end-of-file marks follow a file if that file is the last or only file on the tape.

Files are written on and read from a magtape in a sequential manner. A file consists of an integral number of physical records, separated from each other by interrecord gaps (area on tape in which no data is written). There may or may not be more than one logical record in each physical record.

### 5.8.3 Special Programmed Operator Service

CLOSE performs a special function for magnetic tape. When an output file is closed (both dump and nondump), the I/O service routine automatically writes two end-of-file marks and backspaces over one of them. If another file is now opened, the second end-of-file is wiped out leaving one end-of-file between files. At the end of the in-use portion of the tape, however, there appears a double end-of-file character which is defined as the logical end of tape. When an input dump file is closed, the I/O service routine automatically skips to the next end-of-file.

A special programmed operator called MTAPE provides for such tape manipulation functions as rewind, backspace record, backspace file, 9-channel tape initialization, etc. The format is

MTAPE D, FUNCTION

where D is the device channel on which the magnetic tape unit is initialized. FUNCTION is selected according to the following table:

Table 5-4
MTAPE Functions

| Function | Action |
|---|---|
| 0 | No operation; wait for spacing and I/O to finish |
| 1 | Rewind to load point |
| 11 | Rewind and unload[1] |
| 7 | Backspace record |
| 17 | Backspace file |
| 3 | Write end of file |
| 6 | Skip one record |
| 13 | Write 3 inches of blank tape |
| 16 | Skip one file |
| 10 | Space to logical end of tape |
| 100 | Digital Compatible; 9-channel[2] |
| 101 | Initialize for 9-channel tape[3] |

MTAPE waits for the magnetic tape unit to complete whatever action is in progress before performing the indicated function, including no operation (0). Bits 18 through 25 of the status word are then cleared, the indicated function is initiated, and control is returned to the user's program immediately. It is important to remember that when performing buffered input/output, the I/O service routine can be reading several blocks ahead of the user's program. MTAPE affects only the physical position of the tape and does not change the data that has already been read into the buffers.

5.8.3.1 Backspace File on Magtape - Issuing a backspace file command to a magtape unit will move the tape in the reverse direction until the tape has A) passed the end of file mark or B) reached the beginning of the tape. This means that the end of the backspace file operation will position the tape heads either immediately in front of a file mark or at the beginning of the tape.

[1]On the 516 Control, this function is not currently implemented as such, but is treated as a Rewind function only.

[2]Digital Compatible mode writes (or reads) 36 data bits in five frames of a 9-track magtape. It can be any density, any parity, and is not industry compatible. This mode is in effect until a RELEASE D, or an MTAPE D, 100 is executed.

[3]Industry compatible 9-channel mode writes (or reads) 32 data bits per word in four frames of a 9-track magtape and ignores the last four bits of a word. It must be 800 bpi density, odd parity.

In most cases it is desireable to skip forward over this file mark. This is decidedly not the case if you've reached the beginning of the tape; in this case giving a skip file command would indeed skip the entire first file on the tape stopping at the beginning of the second file, rather than leaving the tape positioned at the beginning of the first file.

Therefore a typical (incorrect) sequence for backspace file would be:

```
MTAPE   MT, 17          ;Backspace file
CALLI   WAIT            ;*Wait for completion*
STATO   MT, 4000        ;Beginning of tape?
MTAPE   MT, 16          ;No, skip over file mark
```

Note that it is necessary to wait after the backspace file instruction in order to insure that the tape is moved to the end of file mark or the beginning of the tape before testing to see whether or not it is the beginning of the tape. The instruction CALLI WAIT cannot be used for this purpose; it waits only for the completion of I/O transfer operation. (Backspace file is a spacing operation, not an I/O transfer operation.)

Instead, use the following sequence for backspace file:

```
MTAPE   MT, 17          ;Backspace file
MTAPE   MT, 0           ;Wait for completion
STATO   MT, 4000        ;Beginning of tape?
MTAPE   MT, 16          ;No, skip over file mark
```

In this case the device service routine waits until the magtape controller is free and proceeds to issue the MTAPE MT, 0 command which tells the tape control to do nothing. Thus the service routine has waited until the completion of the previous operation before issuing the MTAPE MT, 0 and the appropriate wait sequence has been achieved.

### 5.8.4 9-Channel Magtape

Nine-channel magtape may be written and read in two ways: normal Digital Compatible format, and industry compatible format.

5.8.4.1 Digital Compatible Mode - Digital Compatible mode is the usual mode and will allow old 7-channel user mode programs to read and write 9-channel tapes with no modification. Digital Compatible mode writes 36 data bits in five bytes of a nine track magtape. It can be any density, and parity, and is not industry compatible. The software mode is specified in the usual manner during initialization or with a setsts. User mode I/O is handled precisely as in the case of 7-track magtape. It is assumed that most DEC magtapes will be written and read this way.

Data Word on Tape

| Tracks | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| B0 | B1 | B2 | B3 | B4 | B5 | B6 | B7 | P |
| B8 | B9 | B10 | B11 | B12 | B13 | B14 | B15 | P |
| B16 | B17 | B18 | B19 | B20 | B21 | B22 | B23 | P |
| B24 | B25 | B26 | B27 | B28 | B29 | (B30) | (B31) | P |
| 0 | 0 | (B30) | (B31) | B32 | B33 | B34 | B35 | P |

P=Parity
BN=Bit N in core

Data Word in Core - 5 magtape bytes/ 36-bit word. Parity bits are unavailable to the user. Bits are written on tape as shown in diagram, note that bits 30 and 31 get written twice and that tracks 8 and 9 of byte 5 contain 0. On reading parity bits and tracks 8 and 9 of byte 5 are ignored, the or of bits (B30) is read into bit 30 of the data word, the or of bits (B31) is read into bit 31.

5.8.4.2 Industry Compatible Mode - For reading and writing industry compatible 9-channel magtapes, an MTAPE D, 101 UUO must be executed to set the status. MTAPE D, 101 is meaningful for 9-channel magtape only and is ignored for all other devices. In the left half of the status word, bit 2 (which cannot be read by the user program) may be cleared (which returns the device to 9-channel Digital Compatible status) by a RELEAS, a call to EXIT, or an MTAPE D, 100 UUO. These MTAPE UUO's act only as a switch to and from industry compatible mode and in no other way affect I/O status, except to set the density to 800 BPI and odd parity.

On INPUT, four 8-bit bytes are read into each word in the buffer, left justified with the remaining four bits of the word containing error checking information.

On OUTPUT, the leftmost four 8-bit bytes of each word in the buffer are written out in four frames, with the remaining four rightmost bits of the word being ignored.

Data Word on Tape

| Tracks | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| B0 | B1 | B2 | B3 | B4 | B5 | B6 | B7 | B32 |
| B8 | B9 | B10 | B11 | B12 | B13 | B14 | B15 | B33 |
| B16 | B17 | B18 | B19 | B20 | B21 | B22 | B23 | B34 |
| B24 | B25 | B26 | B27 | B28 | B29 | B30 | B31 | B35 |

Data Word in Core - four magtape bytes carry 4 8-bit bytes from data word, parity bits are obtained as shown when reading. Rightmost four bits are ignored on writing. (bits 32-35)

5.8.4.3 Changing Modes - MTAPE CH, 101 automatically sets density at 800 bits (or 808 eight-bytes) per inch and sets odd parity. Note that buffer headers are set up when necessary by the Monitor in the usual manner according to the I/O mode the device is initialized in. Byte pointers and byte counts in buffer header will have to be changed by the user in order to operate on eight-bit bytes.

5.8.5 Special Status Bits

Special bits of the status word are reserved for selecting the density and parity mode of the magnetic tape. Table 5-4 lists the bits that are set and cleared by INIT or SETSTS.

Table 5-5
Magnetic Tape Special Status Bits

| Bit | Action |
|---|---|
| 18[1] | Improper mode. When set to one during an output operation means that the write enable ring is out. |
| 24[1] | I/O Beginning of Tape. The tape is at the load point. |
| 25[1] | I/O Tape END. The tape is at or past the end point. |
| 26 | I/O Parity. 0 for odd parity, 1 for even parity.[2] |
| 27-28 | I/O Density. 00 or 10 = 556 bpi<br>01 = 200 bpi<br>11 = 800 bpi |
| 29 | I/O No Read Check. Suppress automatic error correction if bit 29 is a 1. Normal error correction is to repeat the desired operation 10 times before setting an error status bit. |

[1]These bits indicate special magnetic tape conditions and are set by the magnetic tape service routine when the conditions occur.

[2]Odd parity is preferred. Even parity should be used only when creating a tape to be read in BCD (Binary Coded Decimal) on another computer.

5.9 DISK

Device Mnemonic - DSK

Buffer Size - $203_8$ words (of which $200_8$ words are data)

5.9.1 Data Modes

5.9.1.1 A (ASCII) - Data is written on the disk exactly as it appears in the buffer. Data consists of 36-bit words.

5.9.1.2 AL (ASCII Line) - Same as A.

5.9.1.3 I (Image) - Same as A.

5.9.1.4 IB (Image Binary) - Same as I.

5.9.1.5 B (Binary) - Same as I.

5.9.1.6 DR (Dump Records) - Functions exactly the same as D.

5.9.1.7 D. Dump - Data is read into or written from anywhere in the user's core area without regard to the normal buffering scheme. Control for read or write operations must be via a command list in core memory. The command list format is as described in Chapter 4, "Unbuffered (Dump) Modes." The disk control automatically measures dump data into standard-length disk blocks of 200 octal words. Unless the number of data words is an exact multiple of the standard length of a disk block (200 words) after each command word in the command list, the remainder of that block is wasted.

5.9.2 Structure of Files on Disk

The file structure of the disk system has been designed to minimize the number of disk seeks for sequential or random accessing using either buffered or dump mode I/O. The assignment of physical space for data is performed automatically by the Monitor as logical files are written or deleted by user programs. Files may be of any length, and each user may have as many files as he wishes, as long as disk space is available. No initial estimate of file length or number of files need be given by users or their programs. Files may be simultaneously read by more than one user at a time, thus allowing data sharing. A new version of a file may be recreated by one user while other users continue to read the old version, thus allowing for smooth replacement of shared programs and data files. Finally, one user may selectively update portions of a file, rather than creating a new one (see "General Notes," 5.9.3.3).

5.9.2.1 Addressing by Monitor - The file structure described in this section is generally transparent to the user, and a detailed knowledge of this material is not essential for effective user-mode use of the disk. There are two programs in the Time-Sharing Monitor that service the disk, DSKSER and DSKINT. DSKSER is the device service routine for a disk and references a disk by symbolic addressing only. This routine is essentially independent of what physical disk is attached to the system. DSKINT serves only

two functions: 1) that of translating the logical addressing used elsewhere in the system to the physical addressing of the particular disk being utilized, and 2) controlling the physical disk. The monitor can be thought of as seeing all disks in the same manner; a change of disks requires only a change in DSKINT to provide the proper software interface between the physical device and the rest of the system.

All references made herein to addresses on the disk refer to the logical or relative addresses used by the system and not to any physical addressing scheme involving records, sectors, tracks, etc., that may pertain to a particular physical device. The basic unit which may be addressed is a logical disk block which consists of $200_8$ 36-bit words.

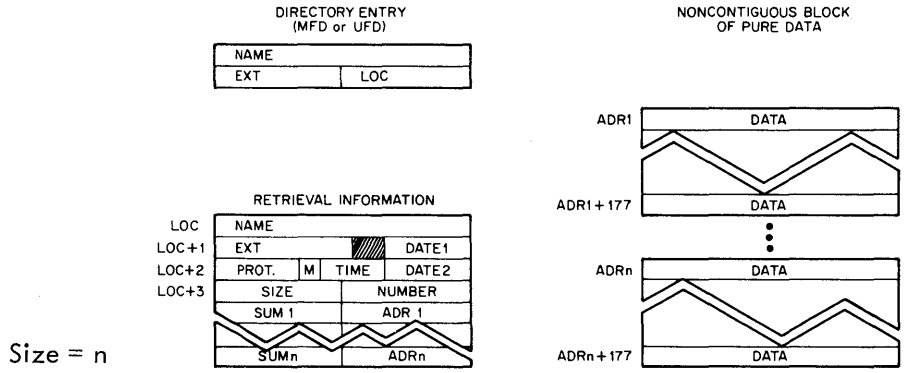
5.9.2.2 Storage Allocation Table (SAT) Blocks - There is a storage allocation table on the disk, which reflects the current status of every addressable block on the disk. These SAT blocks are contained in a file with the name "*SAT* .SYS". This file may be used by any user, but can only be modified by the Monitor. Each addressable block on the disk is represented by one particular bit within the SAT blocks.

If a particular bit is on, it indicates that the corresponding block is filled with data (all blocks on the disk are filled when any information is written on them); if the bit is off, it indicates that the corresponding block is empty or available to be written on. The disk can be wiped out by zeroing the SAT blocks (which is exactly what is done when the disk is refreshed). The disk may optionally be "refreshed" whenever the Monitor is reloaded.

5.9.2.3 File Directories - There are two levels of directories on the disk; one is referenced mainly by the system and the other is referenced by individual users. There is only one higher level directory, known as the Master File Directory (MFD). One of the functions of the MFD is to serve as a directory for individual User's File Directories (UFD's). A UFD is a particular user's own directory and will contain the names of files he has written on the disk. The UFD itself is a file like any other file except that its filename is a binary number combination (project-programmer) rather than a 6-bit code and its extension is always UFD in SIXBIT. The binary combination consists of a left half, which is the project number, and a right half, which is called the programmer number. When a user is logged in under a specific project-programmer number and references the disk, he is actually referencing his own area through the UFD having his project-programmer number as its name. He may, of course, specifically code his routine to reference files listed in the UFD's of other users or the MFD; whether he is successful or not will then depend upon the type of protection that has been specified for the file he is trying to reference.

5.9.2.4 File Format - All disk files (including MFD and UFDs) are composed of two parts: 1) pure data, and 2) information needed by the system to retrieve this data. Each data block contains exactly 200 (octal) words. If a partially filled buffer is output to the disk by a user, a full block is written with trailing zeros filling in to make $200_8$ words. Word counts associated with individual blocks are not retained by the system. If such a partial block is input later, it will appear to have a full $200_8$ data words.

There are three links in the chain by which the system references data on the disk. The first link is the 2-word directory entry in the UFD, which points to the Retrieval Information block(s), which in turn points to the individual pure data blocks. This chain is transparent to the user, who may look upon the directory as having 4-word entries analogous to DECtapes.




## Directory Entry

NAME - Filename in 6-bit ASCII, unless the directory is the MFD and the file is a UFD; in that case, NAME is a project-programmer number in binary.

EXT - Filename extension in 6-bit ASCII; if NAME is a project-programmer number, EXT is UFD.

LOC - Address of the first block on the disk that contains Retrieval Information for this file.

## Retrieval Information

NAME and EXT as above; used to check hardware for possible read error, and to check against software malfunctions. (A failure to match NAME and EXT results in the message "INCORRECT RETRIEVAL INFORMATION".)

DATE1 - In format of DATE UUO; date file last referenced (RENAME, or ENTER, or INPUT done). (Bits 24-35)

DATE2 - Same format as DATE1; date file originally created (ENTER) (bits 24-35).

PROT. - Protection; see below (bits 0-8).

M - Data Mode (ASCII, Binary, Dump, etc.) (bits 9-12).

TIME - 24-hour time (in minutes) that file was originally created (bits 13-23).

SIZE - If negative, this portion indicates the number of words in the file, where all blocks with the possible exception of the last are assumed to contain a full $200_8$ words. If positive, this is a count of the number of $200_8$-word blocks contained in the file. For files of less than $2^{17}$ words, the negative word count is used; for larger files, the positive block count is used instead.

NUMBER - Programmer Number.

SUM 1, ...SUMn - Checksum; two's complement, end-around-carry, sum of data in data-block whose disk address is ADR 1.

ADR1, ...ADRn - Address of data block (logical block number on disk).

## Protection

The first nine bits of the third word of a file's retrieval information are used to specify the protection of the file. This is a necessary procedure since the disk is shared by many users, who may each desire to keep certain files from being written over, read, or deleted by other users.

The total number of users is divided into three categories:

a. Owner of file; (person whose programmer number is the same as that in the right half of the NAME field of the UFD in which the file is entered).

b. Project members; (users whose project number is the same as that in the left half of the NAME field of the UFD in which the file is entered).

c. All other users.

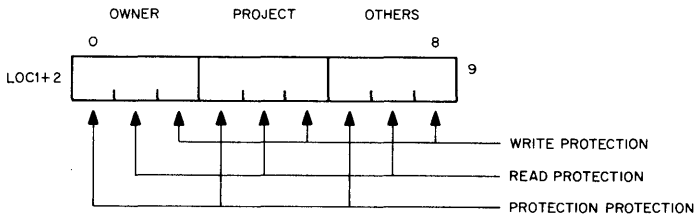
There are three types of protection against each of the three categories of users:

(1) Protection - The protection itself cannot be altered.

(2) Read protection - The file may not be read.

(3) Write Protection - The file may not be rewritten, renamed, or deleted.

The protection mask (see above) consists of the first nine bits of the third word of retrieval information; each bit (when on) represents a particular type of protection against a specific category of user, according to the following scheme. However, owner protection-protection and owner read-protection are ignored lest the file become totally inaccessible.




All files created with an ENTER are given the protection, $055_8$ by the Monitor; if some other protection mask is desired, the RENAME UUO must be employed by the user. (Also see Section 4.4.2.5, "File Protection".)

## 5.9.3 User Programming for the Disk

5.9.3.1 Format - The actual file structure of the disk is generally transparent to the user. In programming for input/output on the disk, a format analogous to that of DECtapes is used; that is, the user assumes a 4-word directory entry similar in form to the first four words of retrieval information. The UUO format is approximately the same as for DECtapes:

UUO D, E

Where UUO is an input/output programmed operator and D specifies the user channel associated with this device. E points to a 4-word directory entry in the user's program which has the following format:

| | | | | |
|---|---|---|---|---|
| E | NAME | | | |
| E+1 | EXT | | | DATE1 |
| E+2 | PROT | M | TIME | DATE2 |
| E+3 | PROJECT NUMBER | | PROGRAMMER NUMBER | |

OR

| | |
|---|---|
| -WORD COUNT | Ø |

(Note that E+3 differs from the fourth word of retrieval information)
(See Retrieval Information, Paragraph 5.8.2.4 for description)

5.9.3.2 Special Functions of Programmed Operators (UUO's) -

ENTER D,E
error return

Causes the Monitor to store away the 4-word directory entry for later entry into the proper UFD when user channel D is CLOSEd or RELEASed.

NAME - The filename must be non-zero, if not, an error return results.

EXT - The file extension may be zero; if so, the Monitor will leave it zero.
DATE1 - The correct date is always filled in by the Monitor.
PROT - The protection is always supplied by the Monitor as 055. The RENAME may be used to change protection after file has been completely written and a CLOSE done.
M - The data mode is supplied by the Monitor as set by the user in the last INIT, or SETSTS UUO on channel D.
TIME, DATE2 - If both of these are 0, the Monitor supplies the current date and time as the creation date and time for the file. If either is non-zero, the Monitor will use the TIME and DATE2 supplied by the user in E+2; thus files may be copied without changing the original creation time and date.
PROJECT-NUMBER, PROGRAMMER-NUMBER - If both of these are 0, the project-number and programmer-number (binary) under which the user is logged-in is supplied by the Monitor. Otherwise the Monitor will use the project-number and programmer-number supplied by the user in E+3, however, it is generally not possible to create (ENTER) files in another user's area of the disk, since UFDs are usually write-protected against all but the owner.

With certain types of error returns peculiar to the disk, the right half of E+1 is set to a specific number to indicate which type of error caused the return. These numbers have the following significance:

0 - E contained a zero file name
1 - E+3 contained an incorrect (or nonexistent) project-programmer number.
2 - File already exists, but is write-protected.
3 - File was being created, recreated, updated, or renamed.

No user, except an administrator with project number 1, may create a UFD, since the MFD is universally write-protected. The LOGIN CUSP (running under the administrator project number) creates a UFD for any user the first time he logs into the system.

When an ENTER is executed by the Monitor on a file that already exists, a new file by that name is written, and those bits in the SAT blocks that correspond to the blocks of the old file are zeroed when the CLOSE (or RELEASE) UUO is executed thereby retrieving space and making it available to any other user.

LOOKUP D, E
error return

Causes the Monitor to read the appropriate UFD. If a later version of the file is being written, the old version pointed to by the UFD will be read.

NAME - The filename in SIXBIT

EXT - The file extension in SIXBIT. A zero extension is not treated in any special manner.

DATE1, PROT, M, TIME, DATE2 are ignored. The Monitor returns these quantities to the user in E+1 and E+2.

PROJECT-NUMBER, PROGRAMMER-NUMBER - If both of these are 0, the project-number and programmer-number (binary) under which the user is logged-in is supplied by the Monitor. Otherwise the Monitor will use the project-number, programmer-number supplied by the user in E+3. Thus, it is possible to read files in other user's directories, provided that the file's protection mask permits reading. The Monitor returns the negative word count (or positive block count for large files) in the LH of E+3, 0 in RH of E+3.

The numbers placed by the Monitor in the right half of E+1 upon an error return have a significance analogous to that described for the ENTER UUO:

0 - File was not found

1 - Incorrect project-programmer number in E+3

2 - Protection failure

3 - File was being created (no earlier version existed).

If the file is currently being recreated, the old file is used.

RENAME D, E
error return

This programmed operator is used to alter the name, extension, and/or protection of a file or to delete a file from the disk. Locations E through E+3 are as described above. RENAME is the only UUO that can set the protection of a file to that specified in E+2. To be RENAMEd a file must first be CLOSEd on channel D, in order to identify for the RENAME UUO. RENAME then seeks out this file and enters the information specified in E through E+2 into the retrieval information and proper directory. If the contents of E is zero, RENAME has the effect of deleting the file.

The error return numbers in the right half of E+1 are the same as for ENTER, with the added possibilities:

4 - Tried to RENAME file to already-existing name.

5 - Neither LOOKUP nor ENTER has been done to identify the file to be renamed.

USETO D, A

USETI D, A

These programmed operators are treated identically by the disk service routines. Their function is to notify the service routine that a particular block is to be used on the next INPUT or OUTPUT on channel D. A is a number that designates a

particular block relative to the beginning of the file. If A is greater than the current size of the file (in blocks), the next OUTPUT will write a block immediately after the file; the next INPUT will cause the end-of-file flag to be set. A=1 refers to the first block of the file (i.e., block 0).

If A = 0 or if no previous LOOKUP or ENTER has been done, this UUO will set the improper mode error bit (see bit 18, Table 4-4, and Section 4.4.4).

5.9.3.3 General Notes - Three types of "writing" on the disk may be distinguished. If a user does an ENTER with a filename which did not previously exist in his UFD, he is said to be "creating" that file. If the filename did previously exist in his UFD, he is said to be superceding that file; the old version of the file stays on the disk (and is available to anyone who wants to read it) until the user does the output CLOSE (at this point, his UFD is changed to point to the new version of the file and the old version is either deleted immediately or marked for deletion later if someone is currently reading it; the space occupied by deleted files is always reclaimed in the SAT tables - see Section 5.8.2.2). Finally, if a user does a LOOKUP followed by an ENTER (the order is important) on the same filename on the same user channel, he will be able to modify selected blocks of that file, using USETO and USETI UUOs, without creating an entirely new version of it; this third type of writing is called "updating" and eliminates the need to copy a file when making only a small number of changes.

As a standard practice, user programs should read, create, and supercede (new file with same filename) files on different user channels. However, for compatibility with DECtapes, it is possible to read and create, or read and supercede, two files on the same user channel as long as all OUTPUTs and the CLOSE output are done before the LOOKUP and the first input, or vice versa. In other words, a CLOSE UUO is required between successive LOOKUPs and ENTERs unless updating is intended.

When issuing a RENAME UUO, the user must insure that the status at locations E through E+3 are as he desires them to be. Since an ENTER or LOOKUP, as well as CLOSE, must have preceded the RENAME; the contents of E through E+3 will have been altered, or filled if the E is the same for all UUO's.

CALL [SIXBIT/RESET/] - Any files which are in the process of being written, but have not be CLOSEd or RELEASed, will be deleted and the space reclaimed. If a previous version of the file with the same name and extension existed, it will remain on the disk (and in the UFD) unchanged.

If the programmer wants to retain the newly created file and have the older version deleted, he must CLOSE or RELEASE the file before doing a RESET UUO.

## 5.10 INCREMENTAL PLOTTER

Device Mnemonic - PLT

Buffer Size - 43 (octal) words

5.10.1 The plotter takes 6-bit characters with the bits of each character decoded as follows:

| Pen Raise | Pen Lower | -X Drum Up | +X Drum Down | +Y Carr-iage Left | -Y Carr-iage Right |
|---|---|---|---|---|---|

Do not combine pen raise or lower with any of the position functions. (For more details on the incremental plotter, see the PDP-10 System Reference Manual, DEC-10-HGAA-D.)

5.10.2 Data Modes

5.10.2.1 A (ASCII) Five, 7-bit characters per word are transmitted to the plotter exactly as they appear in the buffer. Since the plotter is a 6-bit device, the leftmost bit of each character is ignored.

5.10.2.2 AL (ASCII LINE) This mode is identical to the A mode.

5.10.2.3 I (IMAGE) Six, 6-bit characters per word are transmitted to the plotter exactly as they appear in the buffer.

5.10.2.4 B (BINARY) This mode is identical to the I mode.

5.10.2.5 IB (IMAGE BINARY) This mode is identical to the I mode.

5.10.2.6 DR (DUMP RECORDS) Not available.

5.10.2.7 D (DUMP) Not available.

5.10.3 The first OUTPUT operator causes the plotter pen to be lifted from the paper before any user data is sent to the plotter. The CLOSE operator causes the plotter pen to be lifted after all user data is sent to the plotter. These two pen-up commands are the only modifications the monitor makes to the user output file.

## 5.11 DISPLAY WITH LIGHT PEN (TYPE 30 and TYPE 340)

Device Mnemonic - DIS

Buffer Size - None (uses device-dependent dump mode only - 15)

### 5.11.1 Data Words

#### 5.11.1.1 ID (Image Dump - 15)

An arbitrary length area in the user area may be displayed on the scope. The command list format is as described in Chapter 4, "Unbuffered (Dump) Modes," with the addition for the Type 30 display, that, if RH = 0, and LH $\neq$ 0, then LH specifies the intensity for the following data (4 to 13).

### 5.11.2 Background

The purpose of the monitor service routine for the VR-30 is to maintain a flicker-free picture on the display during time-sharing. To do this, the picture data must be available for display at least every two jiffies. This necessitates that the display data remain in core. At present, this means that the user program must also remain in core. To minimize swapping of other programs and to make available a larger block of free core for other users, the user program is shuffled toward the top of core between pictures.

### 5.11.3 Display UUO's

The input/output UUO's for both displays operate as follows:

```
INIT D, 15                  ;MODE 15 ONLY
SIXBIT /DIS/                ;DEVICE NAME
0                           ;NO BUFFERS USED
ERROR RETURN                ;DISPLAY NOT AVAILABLE
NORMAL RETURN

CLOSE D,                    ;STOPS DISPLAY AND
    or                      ;RELEASES DEVICE AS
RELEAS D,                   ;DESCRIBED IN MANUAL
```

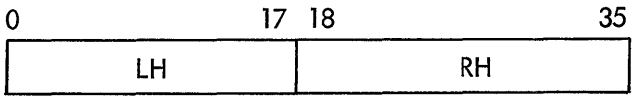
5.11.3.1 INPUT D, ADR

If a light pen hit has been detected since the last INPUT command, then C(ADR) is set to the location of last light pen hit.

If no light pen hit has been detected since last INPUT command, then C(ADR) is set to -1.

5.11.3.2 OUTPUT D, ADR

ADR specifies the first address of a table of pointers. This table is composed of pointers with the following format:



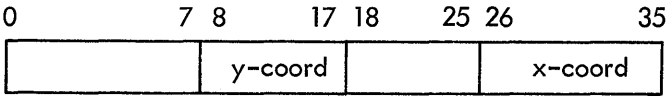

For the VR-30 Display:

If LH = 0 and RH = 0, then this is the end of the command list.

If LH ≠ 0 and RH = 0, then LH is the desired intensity for the following data or commands. The intensity ranges from 4 to 13, where 4 is the dimmest and 13 is the brightest.

If LH = 0 and RH ≠ 0, then RH is the address of the next pointer. Successive pointers are interpreted beginning at RH.

If LH ≠ 0 and RH ≠ 0, then -LH words beginning at address RH+1 are output as data to the display. The format of the data word is the following:




For the 340 Display:

If RH = 0, then this is the end of the command list.

If LH = 0 and RH ≠ 0, then RH is the address of the next pointer. Successive pointers are interpreted beginning at RH.

If LH ≠ 0 and RH ≠ 0, then -LH words beginning at address RH+1 are output as data to the display. The format of the data word is described in the 340 programming manual.

An example of a valid pointer list for the VR-30 Display is:

```
        OUTPUT      D, LIST     ;OUTPUT DATA
                                ;POINTED TO BY LIST
LIST:   XWD         5, 0        ;INTENSITY 5 (DIM)
        IOWD        1, A        ;PLOT A
        IOWD        5,SUBP1     ;PLOT SUBPICTURE 1
```

```
        XWD     13,0        ;INTENSITY 13 (BRIGHT)
        IOWD    1,C         ;PLOT C
        IOWD    2,SUBP2     ;PLOT SUBPICTURE 2
        XWD     0,LIST1     ;TRANSFER TO LIST 1
LIST1:  XWD     10,0        ;INTENSITY 10 (NORMAL)
        IOWD    1,B         ;PLOT B
        IOWD    1,D         ;PLOT D
        XWD     0,0         ;END OF COMMAND LIST
A:      XWD     6,6         ;Y =    6, X = 6
B:      XWD     70,105      ;Y =   70, X = 105
C:      XWD     105,70      ;Y =  105, X = 70
D:      XWD     1000,200    ;Y = 1000, X = 200
SUBP1:  BLOCK   5           ;SUBPICTURE 1
SUBP2:  BLOCK   2           ;SUBPICTURE 2
```

An example of a valid pointer list for the 340 Display is:

```
        OUTPUT  D, LIST     ;OUTPUT DATA POINTED
                            ;TO BY POINTER IN LIST
LIST:   IOWD    1,A         ;SET STARTING POINT TO (6,6)
        IOWD    5,SUBP1     ;DRAW A CIRCLE
        IOWD    1,C         ;SET STARTING POINT TO (70, 105)
        IOWD    5,SUBP1     ;DRAW A CIRCLE
        IOWD    1,B         ;SET STARTING POINT TO (105, 70)
        IOWD    2,SUBP2     ;DRAW A TRIANGLE
        XWD     0,LIST1     ;TRANSFER TO LIST1
LIST1:  IOWD    1,D         ;SET STARTING POINT TO (1000, -200)
        IOWD    5,SUBP1     ;DRAW A CIRCLE
        IOWD    1,A         ;SET STARTING POINT TO (6,6)
        IOWD    2,SUBP2     ;DRAW A TRIANGLE
        XWD     0,0         ;STOP
A:      X = 6       Y = 6
B:      X = 105     Y = 70
C:      X = 70      Y = 105
D:      X = 1000    Y = -200
SUBP1:  BLOCK   5           ;DRAW A CIRCLE
SUBP2:  BLOCK   2           ;DRAW A TRIANGLE
```

The example shows the flexibility of this format. The user can display a subpicture by merely setting up a pointer to it. He can also display the same subpicture in many different places by setting up pointers to the subpicture, each preceded by a pointer to commands for the display to reset its coordinates.

## 5.12 CALL AC, [SIXBIT/DEVCHR/] or CALLI AC, 4

The user may determine the physical characteristics associated with a logical device name

by executing a DEVCHR UUO. The DEVCHR UUO returns the following information in the AC referred.

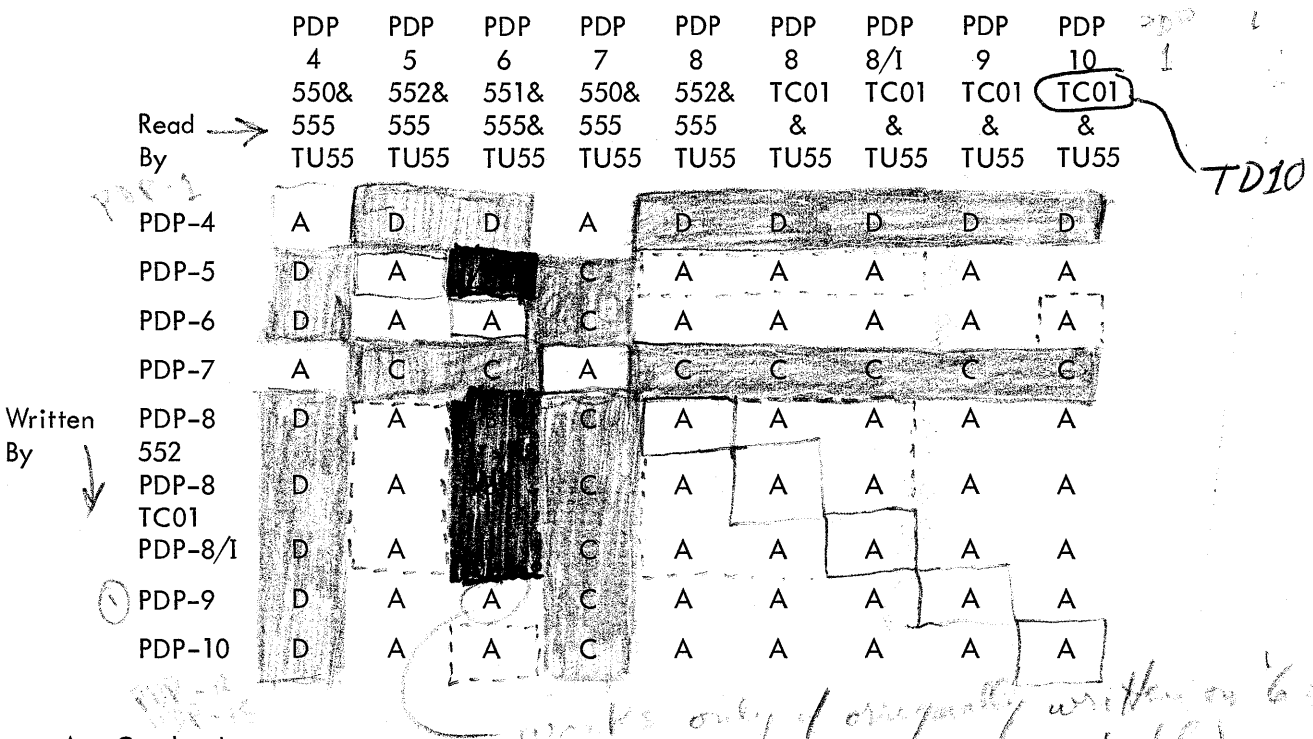
| | | |
|---|---|---|
| $(AC)_L$: | 1 | Device can do output |
| | 2 | Device can do input |
| | 4 | Device has a directory (DTA or DSK) |
| | 10 | Device is a TTY |
| | 20 | Device is a magnetic tape |
| | 40 | Device is available to this job or is already assigned to this job |
| | 100 | Device is a DECtape |
| | 200 | Device is a paper tape reader |
| | 400 | Device is a paper tape punch |
| | 1000 | Device has a long dispatch table (that is, UUO's other than INPUT, OUTPUT, CLOSE, and RELEASE perform real actions) |
| | 2000 | Device is a display |
| | 4000 | TTY in use as an I/O device |
| | 10000 | TTY in use as a user console (even if detached) |
| | 20000 | TTY attached to a job |
| | 40000 | Device is a line printer |
| | 100000 | Device is a card reader |
| | 200000 | Device is a disk |
| | 400000 | DECtape directory is in core (this bit is cleared by an ASSIGN or DEASSIGN command to that unit) |
| $(AC)_R$: | 400000 | Device assigned by a console command |
| | 200000 | Device assigned by program (INIT UUO) |
| | Remaining Bits: | If bit 35-n contains a 1, then mode n is legal for the device. |

NOTE

The mode number (0 through 17) must be converted to decimal; for example, mode $17_8$ is represented by bit $35-15_{10}$ or bit 20.

# APPENDIX 1

## DECtape Compatibility Between DEC Computers



| Written By ↓ / Read → | PDP 4 550& 555 TU55 | PDP 5 552& 555 TU55 | PDP 6 551& 555& TU55 | PDP 7 550& 555 TU55 | PDP 8 552& 555 TU55 | PDP 8 TC01 & TU55 | PDP 8/I TC01 & TU55 | PDP 9 TC01 & TU55 | PDP 10 TC01 & TU55 |
|---|---|---|---|---|---|---|---|---|---|
| PDP-4 | A | D | D | A | D | D | D | D | D |
| PDP-5 | D | A | C | C | A | A | A | A | A |
| PDP-6 | D | A | A | C | A | A | A | A | A |
| PDP-7 | A | C | C | A | C | C | C | C | C |
| PDP-8 552 | D | A | C | C | A | A | A | A | A |
| PDP-8 TC01 | D | A | C | C | A | A | A | A | A |
| PDP-8/I | D | A | C | C | A | A | A | A | A |
| PDP-9 | D | A | A | C | A | A | A | A | A |
| PDP-10 | D | A | A | C | A | A | A | A | A |

A = Can be done

B = Can not be done because of difference in writing checksum

C = Can be done with programmed checksum

D = Can probably be done as in (C) except that PDP-4 is too slow for calculating the exclusive or checksum in line - this must be done before writing.

NOTE: PDP-10 will not allow search to find first or last blocks when searching from the end-zone.

# APPENDIX 2

Size of Multiprogramming non-disk Monitor (Reentrant 4 series, Version 50) June, 1969

There are three components to the Monitor:

1) Required code (4.7K)
2) Optional device code (0-4.4K)
3) Tables and buffers per job (73 words per job)

A. Required code (Assuming all features)

| | |
|---|---|
| Lower core | 96. |
| COMMON | 409. |
| CLKCSS | 82. |
| CLOCK1 | 367. |
| COMCON | 1322. |
| CORE1 | 182. |
| DLSINT | 48. |
| ERRCON | 214. |
| SCNSRF | 1260. |
| SEGCON | 602. |
| SYSINT | 78. |
| UUOCON | 1144. |
| | 4692. words (Decimal) |

B. Optional devices

| | | | Complete system |
|---|---|---|---|
| DTA | 1284. | +N(1)*146. N(1) = 8 | 2612. |
| MTA | 452. | +N(2)*9. N(2) = 2 | 470. |
| PTY | 176. | +N(3)*10. N(3) = 2 | 196. |
| CDR | 220. | | 220. |
| CDP | 308. | | 308. |
| DIS | 190. | | 190. |
| LPT | 100. | | 100. |
| PLT | 65. | | 65. |

| Optional devices | | Complete system |
|---|---|---|
| PTP | 167. | 167. |
| PTR | 105. | 105. |

3067. +N(1)=146.+N(2)*9.N(3)*10. 4433.

C. Tables and buffers

18. words of tables per job

55. word of TTY device data block space per job

---

73. words per job

Total for complete 8 user system = 4692. + 443. + 8.*73. = 9709.

WARNING: The Monitor will continue to grow despite our best efforts to prevent it. Most new features are put in with conditional assembly so that a customer can reduce this size of the Monitor by giving up some of the new features.

These sizes are subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation.

# APPENDIX 3

Size of Swapping Monitor (Reentrant 4 series, Version 50) June, 1969

There are three components to the Monitor:

1) Required code (10K)
2) Optional device code (0-4K)
3) Tables and buffers per job (1K for every 8 jobs)

A. Required code (Assuming all features)

| | |
|---|---|
| Lower core | 96. |
| COMMON | 475. |
| CLOCK1 | 376. |
| COMCON | 1592. |
| CORE1 | 214. |
| DLSINT | 48. |
| DSKINT | 130. |
| DSKSRB | 2448. |
| ERRCON | 211. |
| SCHEDB | 741. |
| SCNSRF | 1264. |
| SEGCON | 709. |
| SYSINI | 81. |
| UUOCON | 1190. |
| | 10375. words (Decimal) |

B.

| Optional devices | | | | Complete system |
|---|---|---|---|---|
| DTA | 1286. | +N(1)*146. | N(1) = 8 | 2454. |
| MTA | 452. | +N(2)*9. | N(2) = 2 | 470. |
| PTY | 166. | +N(3)*10. | N(3) = 2 | 196. |
| CDR | 220. | | | 220. |
| CDP | 308. | | | 308. |
| DIS | 191. | | | 191. |
| LPT | 104. | | | 104. |

| Optional Devices | | Complete system | |
|---|---|---|---|
| PLT | 80. | 80. | |
| PTP | 167. | 167. | |
| PTR | 105. | 105. | |
| | 3089. +N(1)=146.+N(2)*9.+N(3)*10. | | 4295. |

C. Tables and buffers

21. words of tables per job

54. words of DSK device data block space per job
(1.5 files/job)

55. word of TTY device data block space per job

130. words per job

Total for complete 16 user system = 10375. + 3987. + 16.*130. = 16442.

WARNING: The Monitor will continue to grow despite our best efforts to prevent it. Most new features are put in with conditional assembly so that a customer can reduce this size of the Monitor by giving up some of the new features.

For a complete Swapping System (all devices):

| | | |
|---|---|---|
| 8 | JOBS | 15.7K |
| 16 | JOBS | 16.7K |
| 24 | JOBS | 17.7K |
| 32 | JOBS | 18.7K |
| 40 | JOBS | 19.7K |
| 48 | JOBS | 20.7K |
| 56 | JOBS | 21.7K |
| 64 | JOBS | 22.7K |

These sizes are subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation.

# ADDENDUM 1
## Concise Command Language (CCL) for the PDP-10 Time-Sharing Monitors

TRITE

This should be the 1st chapter; not the last

## 1. SCOPE

This document describes the use of the Concise Command Language[1] (CCL) features of the PDP-10 Time-Sharing Systems.

The discussion in this document assumes that the reader is at least slightly familiar with the use of the PDP-10 Time-Sharing system. Each section begins with fairly simple concepts and progresses toward more complex cases.

Flush irrelevant or obvious

It is assumed that the reader has some knowledge of the following PDP-10 CUSPs (Commonly Used System Programs):

FORTRAN IV or MACRO 10,

LOADER,

PIP, and either of the editing programs, TECO or LINED ( a version of EDITOR).

Detailed information of the implementation of the CCL system is also included in this document.

## 2. INTRODUCTION

The CCL (Concise Command Language) ~~system has been added to the PDP-10 Time-Sharing System with the primary purpose of~~ reduces the amount of typing (both input and output), required for a user to accomplish common tasks, such as the translation, loading, and execution of source language programs. A secondary result is that routine operations can be performed more rapidly by using the CCL system, since there is much less machine time spent waiting for type-in from the user. There are fewer typing errors, since there is less type-in, and there is less to learn, for the beginning user, before programs can successfully be run.

CCL commands are to all intents and purposes new Monitor commands as far as the user is concerned. They are typed at the Monitor level.

The CCL system is implemented only for PDP-10 configurations containing a disk, and relies heavily on temporary disk files (which can be ignored by the user).

At the same time, an attempt has been made to maintain within the CCL system much of the generality of the more detailed use of the PDP-10 CUSPs. Much of this document is devoted to those more complex uses of the CCL system.

---

[1] COMPIL, the CCL control cusp, was developed by William F. Weiher, of the Artificial Intelligence Project, Stanford University.

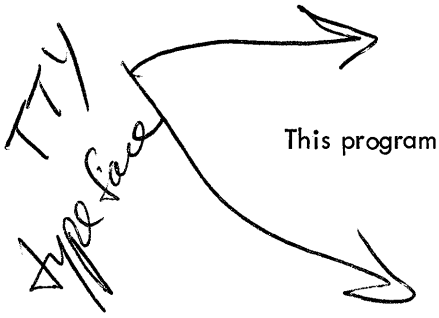
The beginning user can make good use of the CCL system without knowing all of these details, as will be shown in the simple examples in Section 3.

## 3. SIMPLE EXAMPLES[1]

Perhaps the best example of the simplicity of the use of the CCL system is the following:

Assume that a user has one, and only one, file on his disk area, a file called PROG.F4, whose contents are:

```
      TYPE 100↵
100   FORMAT (' HELLO ')↵
      END↵
```




This program can be compiled, loaded and executed by typing simply:

```
1)    .EXECUTE PROG↵
```

The typeout will be:

```
FORTRAN: PROG
LOADING
LOADER 5K CORE
HELLO
EXIT
↑C
.
```

Operations requiring PIP are also greatly simplified. To type out the contents of the above file, one need only type:

```
2)    .TYPE PROG.F4↵
```

To list one's disk directory, the command is:

```
3)    .DIRECT↵
```

Similarly, to edit the text file above, the command

```
4)    .EDIT PROG.F4↵
```

will initialize the line editor to that file, or

```
5)    .TECO PROG.F4↵
```

will initialize TECO to that file.

---

[1] Throughout this document, computer typeouts are indicated by underscoring, and the ↵ symbol is used to represent the RETURN key.

↑ CARRIAGE

The equivalent commands for the previous examples without CCL are (type-in only):

1)
```
.R F40↵
*DSK:PROG←DSK:PROG↵
*↑C
.R LOADER↵
*/E PROG (ALTMODE)
```

2)
```
.R PIP↵
*TTY:←DSK:PROG.F4↵
```

3)
```
.R PIP↵
*TTY:←DSK:/L↵
```

4)
```
.R LINED↵
*SPROG.F4↵
```

5)
```
.R TECO↵
*EBPROG.F4 (ALTMODE) Y (ALTMODE) (ALTMODE)
```

# 4. COMMAND DESCRIPTION

## 4.1 COMPILE-Class Commands

### 4.1.1 General

This section describes the use of the following group of commands:

COMPILE, LOAD, EXECUTE, and DEBUG

The COMPILE command will be described first, then the other three commands will be explained as extensions of COMPILE.

The argument of a COMPILE command is, in its simplest case, a filename or a list of filenames. In more complex cases, there are many optional switches. While reading these sections the reader should be careful to distinguish between "compile-switches", "processor-switches", and "loader-switches". They will be distinctly separated by these names.

### 4.1.2 The COMPILE Process

4.1.2.1 General - The purpose of the COMPILE command is to produce one or a group of relocatable binary files, representing the specified program(s). This process may require the use of the MACRO assembler, or of the FORTRAN compiler, or both. (Other translators may be added to this list in the future). If the list of programs is extensive, there may be both source and binary versions of some programs, and some of them may not require recompilation while others may. The COMPIL program

makes these decisions and directs the compilations and assemblies, according to the rules and commands which will be described here.

4.1.2.2 Program Names - A file on the PDP-10 disk is identified by a filename of up to six characters, and a filename extension of up to three characters. Associated with each such file is a creation date and time. (DECtape files have a date, but no time). Certain file extensions imply particular forms of programs:

| Extension | Meaning |
|---|---|
| .MAC | MACRO language source file |
| .F4 | FORTRAN IV source file |
| .REL | RELocatable binary file |
| .SAV | Core dump, from SAVE command |
| blank | Source file, unspecified language |

The compile process uses these extensions and dates to direct the compilations.

4.1.2.3 COMPILING a Program - The command

.COMPILE FILE↵

causes the following actions:

1) Determine whether a file named FILE.REL exists, and if so determine its date and time.

2) Determine whether a source file exists by the name FILE (with a null extension) or failing that) by the names FILE.MAC or FILE.F4.

3) If there is a source file, and its date and time are at least as recent as those of the .REL file (if any), then have it translated to a .REL file. Select the correct translator as follows:

> If the source is .MAC, use MACRO.
> If the source is .F4, use FORTRAN IV.
> If the source has another extension (or a null extension), use the "standard processor." The "standard processor" is FORTRAN IV at the beginning of each command, but may be changed by the use of COMPILE switches (see below).
> Ambiguity of processors can most easily be avoided by always using the conventional extensions of .MAC or .F4 for source files.

Input files may appear on devices other than the disk, if so specified, or on disk areas belonging to other users. Output files, both binary and listing (see below), will be placed on the user's disk area. The user's own area will be searched for .REL files during the process of checking for the most recent .REL file. For example, if the user is logged in as [ 20,20], and the command

.COMPILE PROG [ 30,30 ]↵

is executed, both PROG.REL [30,30] and PROG.REL [20,20] will be searched for before compiling the source file. If a compilation does occur, the output will be PROG.REL [20,20].

4.1.2.4 Lists of Programs - The argument of the COMPILE command may be a string of program names, separated by commas. Programs in both FORTRAN and MACRO languages may be mixed in such a list. For example, if the user had files named A.MAC, B.F4 and C.MAC, then either of the following commands would cause the three programs to be translated, by the proper translator:

1) .COMPILE A.MAC, B.F4, C.MAC↵
   or more simply,
2) .COMPILE A, B, C↵

4.1.2.5 COMPILE Switches - The COMPILE-class commands may be modified by a number of switches. These switches are words preceded by slashes ("/"), as opposed to letters preceded by slashes more commonly found in other command strings. The switches are delimited by any non-alphanumeric character, usually space or comma. The names of the switches may be abbreviated to the first letter or letters, provided that the abbreviation uniquely identifies a particular switch.

Compile-switches may be either "temporary" or "permanent". A temporary switch is appended to the end of a filename, without an intervening space, and has effect only on the processing of that file. Example:

.COMPILE A,B/SWITCH,C↵

In this case, the action implied by "SWITCH" would be applied only to file B.

A permanent switch is set off from filenames by a space or comma. It takes effect on all following files, unless modified by another, later switch. Example:

.COMPILE A,/SWITCH B,C↵

Here, "SWITCH" applies to both file B and file C.

Specific compile-switches will be discussed in the following sections. Some are relevant only to the LOAD, EXECUTE, and DEBUG commands, and will be discussed in those sections.

4.1.2.6 Compilation Listings - Listing files may be generated by use of switches. The listings may be of the ordinary or the cross-reference type.

The compile-switches "LIST" and "NOLIST" cause listing and non-listing of programs. These switches may be used as either temporary or permanent switches:

.COMPILE /LIST A,B,C↵

will generate listings of all three programs.

.COMPILE A/LIST, B,C↵

will generate a listing only of program A.

.COMPILE /LIST A, B/NOLIST, C↵

will generate listings of programs A and C.

The compile-switch "CREF" is just like "LIST", except that a cross-reference listing is generated, which must be processed by the program "CREF".

In either case, the result of the listing is a disk file, with the extension .LST, which must be listed later. (See the "LIST" and "CREF" commands).

Since the "LIST", "NOLIST", and "CREF" switches are so commonly used, the switches "L", "N", and "C" are defined with the corresponding meanings, even though there are (for instance) other switches beginning with the letter "L". Thus the command

.COMPILE /L A↵

produces a listing file "A.LST" (as well as, of course, "A.REL").

4.1.2.7 The "Standard Processor" - The "standard processor" is used to compile or assemble programs which do not have the extensions .MAC, .F4, or .REL. There are a number of switches for setting the "standard processor". It should be emphasized here, however, that this subject can be disregarded if all source files are kept with the appropriate extensions.

If the command

.COMPILE A↵

is executed, and there is a file named "A.", that is, with a blank extension, then "A." will be translated to "A.REL" by the "standard processor". Similarly, if the command

.COMPILE FILE.NEW↵

is executed, the extension ".NEW", although meaningful to the user, does not specify a language, so the "standard processor" will be used. For these cases the user must be able to control the setting of the "standard".

The "standard processor" is FORTRAN IV at the beginning of each COMPILE-class command.

The "standard processor" may be changed by the following compile-switches:

| | |
|---|---|
| MACRO | change standard to MACRO |
| M | same as MACRO |
| FORTRAN | change standard to FORTRAN IV |
| F | same as FORTRAN |
| REL | change standard to use RELocatable binary; i.e., use existing .REL files, even though a newer source file may be present. (Useful primarily in LOAD, EXECUTE, DEBUG commands). |

These switches may be used as "temporary" or "permanent". For example, assume that programs A, B, and C exist on the disk, with blank extensions. Then

.COMPILE A, B/M, C ↵

will cause A and C to be translated by FORTRAN, B by MACRO.

.COMPILE A, /M B, C ↵

will cause A to be translated by FORTRAN, B and C by MACRO.

NOTE

Programs with .MAC and .F4 extensions are always translated by the extension implied processor, regardless of the "standard processor."

4.1.2.8 Forced Compilation - It was stated earlier that the compilation (or assembly) occurs if the source file is at least as recent as the relocatable binary file. If the binary is newer than the source, there is not normally any need to perform the translation.

There are cases, however, where such extra translation may be desirable, as for instance, when one desires a listing of the assembly. To force such an assembly, the switch "COMPILE" is provided, again in both temporary and permanent form. For example:

.COMPILE /CREF / COMPILE A, B, C ↵

will create cross-reference listing files A.LST, B.LST, and C.LST, even though current .REL files may exist. In fact, the binary files will also be recreated.

The corresponding switch "NOCOMPILE" is also provided, to turn off the forced-compile mode. Note that this differs from the /REL switch which turns off even the normal compilation caused by a source file newer than the .REL file.

### 4.1.3 LOAD Commands

4.1.3.1 General - The LOAD command is an extension of the COMPILE command. It takes as its argument the same sort of list of program names and switches as does the COMPILE command, with the addition of some further optional switches, to be described below.

The first action of the LOAD command is to invoke the COMPILE process, as described in Section 4.1.2. That is, the program names are checked for the existence of source and binary files, and assemblies and compilations are performed as required by the dates and times of the files and by the compile-switches in the command.

The second action of the LOAD command is the running of the LOADER, and the loading of the .REL files specified in the LOAD command.

Various loader actions can be directed by means of switches to be described below.

At the end of loading, the LOADER EXITs to the Monitor.

#### 4.1.3.2 Compile-Switches for the LOAD Process

4.1.3.2.1 Library Searches - The LOADER normally performs a library search of the FORTRAN library. Sometimes it is necessary to search other files as libraries. To do this, the compile-switches "LIBRARY" and (its complement) "NOSEARCH" are provided.

These switches may be used as either "permanent" or "temporary" (review Section 4.1.2.5).

For example, suppose a special library file named SPCLIB.REL were kept on device SYS at a particular installation. Then to compile and load a user program, library search the special library, and then search the normal FORTRAN library, the following command could be used:

```
.LOAD MAIN,SYS:SPCLIB/LIB↵
```

At this point, it should be noted that the program SPCLIB is not assembled simply because its source file is presumably not on device SYS. The COMPILE process will compile any program named in the command string, if its source is present and not older than the .REL file, unless prevented by the /REL switch.

4.1.3.2.2 Loader Maps - Loader maps are produced during the loading process by the compile-switch "MAP". When this switch is encountered, a loader map is requested from the Loader. The map will be written with filename MAP.MAP, in the user's disk area.

This compile-switch is the one exception to the "permanent compile-switch" rule, in that it causes only one map to be output, even though it may appear as a permanent switch.

4.1.3.3 Loader-Switches in a CCL Command - In unusually complex loading processes, it may be necessary to pass loader-switches to the LOADER to direct its operation. The most common examples are: loading with symbols (loader-switch /S), setting a program origin (loader-switch /no), causing an early search of the FORTRAN library (loader-switch /F), or preventing the library search (loader-switch /P).

These switches must be passed to the LOADER, but not to the assembler or compiler. This is accomplished by the % character. The syntax of % is the same as that of "/" in the Loader's command string; that is it takes one letter following it, or a sequence of digits and one letter. Thus to set a program origin of 6000 for program C, one might type:

.LOAD A, B, %6000OC, D↵

The relation between loader-switches and filenames is maintained, so that, for instance

%SFILE is passed as /SFILE while
FILE%S is passed as FILE/S

(These have different meanings to the LOADER.)

## 4.1.4 EXECUTE Command

4.1.4.1 General - The EXECUTE command is a further extension of the COMPILE-class commands. It causes all of the actions of the LOAD command, and takes the same arguments. In addition, however, the EXECUTE command causes the LOADER to begin execution of the program at its starting address at the completion of loading. Thus, in one command, the CCL system proceeds from compilation of the source files, through the loading process, and into execution of the program.

4.1.4.2 Error Comment - The assembler, compiler, and LOADER keep a count of errors encountered during their operation. If an EXECUTE command encounters any errors, the loading will occur, but the message "EXECUTION DELETED" will be printed, and the LOADER will EXIT to the Monitor without starting the program.

## 4.1.5 DEBUG Command

The DEBUG command is nearly identical to the EXECUTE command. It takes the same arguments and switches as both EXECUTE and LOAD, and invokes the COMPILE and LOAD processes as previously described. The one difference is that loading is begun with the "/T" switch to the Loader. That is, the debugging program DDT is loaded as the first program, subsequent programs are loaded with local symbols, and DDT is entered at the completion of loading (rather than either an EXIT or START operation).

### 4.1.6 Remembered Arguments

Each time a COMPILE-class CCL command is performed, it is "remembered" as a file on the disk, so that its arguments may be re-used by a later COMPILE-class command. If a COMPILE-class command is executed with no argument, the previous argument will be recalled and substituted into the command.

For example, suppose that the following command is performed:

.EXECUTE A, B, C↵

and that a syntax error is found while compiling program C, so that the execution is deleted. The user should edit program C, and then type simply

.EXECUTE↵

The system will recall the arguments (A, B, C), recompile program C, and then load and execute the three programs.

If the use of DDT seems required, the command

.DEBUG↵

would also recall the necessary arguments.

## 4.2 EDIT-Class Commands

### 4.2.1 General

These commands call in the PDP-10 editing programs, and cause them to open a specified text file for editing. There are four of these commands; two address TECO, and two address LINED (a disk-oriented version of EDITOR). For each editor, one command causes an existing file to be opened for changes, and the other causes a new file to be created. Each command requires as its argument a filename, with an (optional) extension. This file-name may be implied (see below, Section 4.2.4).

### 4.2.2 LINED Commands

4.2.2.1 EDIT Command - This command causes LINED to open a file for modifications. The file must already exist, and be a sequence-numbered text file. Example:

.EDIT NAME.MAC↵

This is equivalent to:

```
.R LINED↵
*SNAME.MAC↵
```

4.2.2.2 CREATE Command - This command causes LINED to open a new file for creation. Example:

```
.CREATE NAME.F4↵
```

This is equivalent to:

```
.R LINED↵
*SNAME.F4 (ALTMODE)
```

4.2.3 TECO Commands

4.2.3.1 TECO Command - This causes TECO to open a file for modifications. The file must already exist. Example:

```
.TECO NAME.MAC↵
```

This is equivalent to:

```
.R TECO↵
*EBNAME.MAC (ALTMODE) Y (ALTMODE) (ALTMODE)
```

4.2.3.2 MAKE Command - This command causes TECO to open a file for creation. Example:

```
.MAKE NAME.F4↵
```

This is equivalent to:

```
.R TECO↵
*EWNAME.F4 (ALTMODE) (ALTMODE)
```

4.2.4 Implied EDIT-Files

Each time an EDIT-class CCL command is performed, it is "remembered" as a file on the disk, so that the filename last edited may be recalled for the next edit. For example, if the command

```
.CREATE PROG1.MAC↵
```

is performed, then later in the operating session, the command

```
.EDIT↵
```

may be used in place of

```
.EDIT PROG1.MAC ↵
```

assuming no other EDIT-class command was used in the interim.

## 4.3 PIP-Class Commands

These commands use PIP and CREF to perform common file-handling operations, with new abbreviated commands.

### 4.3.1 TYPE Command

The TYPE command causes PIP to type the contents of a file, or files, on the user's Teletype. For example, the command

```
.TYPE FILE1.F4 ↵
```

will cause the program FILE1.F4 on the disk to be printed.

The TYPE command also accepts a list of filenames, or the "wild" filename "*". Filenames may include device names and project-programmer numbers. For example

```
.TYPE DTA1:A.F4,DSK:B.MAC,*.DAT ↵
```

will cause typing, in sequence, of file A.F4 from DECtape 1, then B.MAC from the disk and finally all disk files with the extension DAT.

### 4.3.2 LIST Command

The LIST command is exactly the same as the TYPE command (4.3.1) except that the named files are listed on the line-printer (device LPT) instead of the user's console (device TTY).

### 4.3.3 DIRECTORY Command

4.3.3.1 General - The DIRECTORY command produces a directory listing of the user's disk files, listed on the Teletype. If a device name is supplied, with an optional colon, the directory of that device will be listed. For example, either

```
.DIRECT DTA1: ↵     or
.DIRECT DTA1 ↵
```

will produce a directory listing of DECtape 1.

4.3.3.2 DIRECTORY Switches - The DIRECTORY command accepts two optional switches, which may appear either before or after the device name.

4.3.3.2.1 /F Switch - The /F switch causes the short form of the directory to be listed (omitting dates). Example:

```
.DIRECT/F↵
```

4.3.3.2.2 /L Switch - The /L switch causes the listing to be sent to the LPT. For example,

```
.DIRECT SYS:/L ↵
```

lists the system directory on the line printer.

### 4.3.4 DELETE Command

The DELETE command may be used to delete a file or files from the disk or a DECtape. The argument of DELETE is a list of filenames, which may include device names. If no device is supplied, DSK is assumed. The "wild" filename "*" is allowed.

Examples:

```
.DELETE ABC.MAC          (deletes DSK:ABC.MAC)
.DELETE DTA1:ABC.MAC,DEF.MAC↵
                         (deletes two DECtape files)
.DELETE DTA1:A,B,DSK:*.TMP↵
                         (deletes two DECtape files,
                         and all .TMP files from the disk)
```

Note that a device name remains in effect until changed or until the end of the command. Thus in the second example, DEF.MAC, and in the last example, B, are deleted from the DECtape.

### 4.3.5 RENAME Command

The RENAME command changes the names of files on the disk or (if specified) on DECtapes. The argument is a pair of filenames separated by an equals sign, or a list of such pairs, separated by commas. The old filename appears at the right of the equals sign: the new name is at the left. If a device name is included, it is specified with the new name, and remains in effect until explicitly changed, or until the end of the command. The device is initially assumed to be the DSK.

Examples:

```
.RENAME NEW = OLD↵
.RENAME DTA1:NEW1.MAC = OLD1.MAC,N2 = 02↵
.RENAME *.MAC = *.XYZ↵
.RENAME DTA1 :N = 0, DSK: N = 0↵
```

### 4.3.6 CREF Command

The CREF command takes no arguments. It causes the CREF cross-reference program to be run, and produces any cross-reference listings which have been generated by previous COMPILE-class commands. The CREF program then deletes its file of listing names, so that subsequent CREF commands will not produce these same listings again.

## 5. EXTENDED COMMAND FORMS

The commands described in the preceding section are sufficient for the compilation and execution of one or a few programs at a time. By use of the extended forms of the CCL commands to be described in this section, it becomes a simple task to assemble large groups of programs such as the FORTRAN library or the Time-Sharing Monitor itself.

### 5.1 Command Files

#### 5.1.1 The @ File

When the number of program names and switches is large, it is desirable to place them in a file rather than typing them in for each compilation. This is accomplished by the "@file" construction. At any point in a CCL command, after the first word, the "@file" may appear, where "file" may include a filename, extension, and project-programmer number. If the extension is blank, the command file will be looked for with both the blank extension, and the extension ".CMD". The information in the command file is then "plugged in", replacing the characters "@file".

For example, if a file "DSK:NAMES.CMD" contains the following line,

```
/LIST A, B, C, D, E
```

then the command

```
.COMPILE /LIST A, B, C, D, E, F↵
```

could be replaced by

```
.COMPILE @NAMES, F↵
```

Command files may themselves contain the "@" construction, and this process may continue to a depth of nine files. If this indirecting process should result in files pointing in a loop, the maximum depth will rapidly be exceeded and an error message will be produced.

### 5.1.2 Formatting within a Command File

In a very large command file, such as one used to assemble the Time-Sharing Monitor, questions of format and commentary must be defined. The following rules describe the handling of format characters in command files.

a. Except that they delimit words, spaces are ignored. Similarly, the characters TAB, VTAB, and FORM are treated like spaces.

b. The characters CARRIAGE RETURN, LINE-FEED, and ALTMODE are ignored if the first non-blank character after a sequence of returns, linefeeds and altmodes is a comma. Otherwise, they are treated as commas by the COMPILE-class commands, and as command terminators by the EDIT and PIP class commands.

c. Since strings of returns and linefeeds are considered together, blank lines are completely ignored.

d. Comments may be included in command files by the use of a semicolon. All text from a semicolon through the following linefeed, inclusive, is ignored.

e. Command files may be sequenced. The sequence numbers are ignored.

## 5.2 The "+" Construction (COMPILE-class commands only)

In many cases, a single MACRO program may be produced from a collection of input files. The most common case of this is the Time-Sharing system, where most files consist of a parameter file, S.MAC, possibly a switch file such as FT50SB.MAC, and a file which is the body of the program, such as APRSER.MAC. This is specified by the following command:

```
.COMPILE S+FT50SB+APRSER
```

In this construction, the name given to the output files, .REL and .LST (if any), is that of the last of the input files, i.e., APRSER in this example.

The individual source files in the "+" construction may each contain device, extension, and project-programmer number information as well as a filename.

## 5.3 The "=" Construction (COMPILE-class commands only)

Normally, the filename of the binary file is the same as that of the source file, with the extension specifying the difference between them. It is sometimes desirable to suspend this convention, and this can be done via the "=" construction, best shown by example: Suppose a source program is

named SOURCE.MAC, and a binary file is desired with the name BINARY.REL rather than SOURCE.REL. The following command accomplishes this:

```
.COMPILE BINARY=SOURCE↵
```

This same technique may be used to specify an output name to the file produced via the "+" construction. To give the name WHOLE.REL to the binary produced by PART1.MAC and PART2.MAC, type the following:

```
.COMPILE WHOLE=PART1+PART2↵
```

## 5.4 The "< >" Construction (COMPILE-class commands only)

A further simplification can be introduced when a number of programs are to be assembled with the same parameter file. To assemble the three monitor files LPTSER.MAC, PTPSER.MAC, and PTRSER.MAC, each with the parameter file S.MAC, one could type

```
.COMPILE S+LPTSER,S+PTPSER,S+PTRSER↵
```

This can be simplified by the use of angle brackets, as follows:

```
.COMPILE S+<LPTSER,PTPSER,PTRSER>↵
```

It is not permissible to place a "+" term after the angle brackets, such as:

```
.COMPILE <LPTSER,PTPSER,PTRSER>+S↵
```

or

```
.COMPILE S+<LPTSER,PTPSER,PTRSER>+FT50SB↵
```

## 5.5 Processor Switches

The LIST and CREF operations are implemented by passing switches to the assembler and compiler in their commands. It is occasionally necessary to pass other switches to these processors, such as tape positioning commands. A mechanism is provided for this in the CCL system.

First, recall that for each translation (assembly or compilation) a command string is sent to the translator, containing three parts: source files, binary output file, and listing file. The CCL system passes these command strings to the processor without the user's being concerned with the details. However, if the user does wish to add switches to these files, the following technique may be used.

First, group the switches according to the three files, and according to each source file if the "+" construction is used.

Secondly, group the switches with commas and a pair of parentheses for each source filename, as follows:

If only source switches are present, place them in parentheses:

(SSSS)

If binary switches are present, place them after the source switches and a comma:

(SSSS,BBBB)

If listing switches are present, place them after the binary switches and another comma:

(SSSS,BBBB,LLLL)

Finally, place this parenthesized string immediately after the source-file name. For example, to assemble the first two files on MTA0, naming the output files A and B, type

.COMPILE/MACRO A=MTA0: (W),B=MTA0:↵

This will cause the following MACRO command strings to be generated:

A ← MTA0:(W)
B ← MTA0:

## 6. IMPLEMENTATION

### 6.1 General

The CCL system is implemented as a combination of additions to the Time-Sharing Monitor, modifications to the PDP-10 cusps (MACRO, F40, PIP, TECO, LINED, LOADER and CREF), and a new cusp (COMPIL) which deciphers the CCL commands and constructs commands for the cusps. These commands are written as temporary files on the disk, as are the monitor-level commands. COMPIL and the other cusps transfer control directly from one to another without requiring typed-in commands.

### 6.2 Temporary Files

This section describes the uses of the various temporary files, their names and their contents. The first three characters of each filename are the job number of the controlling job, in decimal, with leading zeroes to make three digits. The names listed here will assume job number one.

### 6.2.1 001SVC.TMP

This file contains the most recent COMPILE-class command which included arguments. It is used to remember those arguments, as described in Section 4.1.6.

### 6.2.2 001EDS.TMP

This file contains the most recent EDIT-class command which included an argument. It is used to remember that argument, as described in Section 4.2.4.

### 6.2.3 001MAC.TMP

This file contains commands to MACRO. It is written by COMPIL, and read by MACRO. It contains one line for each program to be assembled, and (if required) the command

NAME!

to cause MACRO to transfer control to the named cusp ("name" may be F40, LOADER, etc.).

### 6.2.4 001FOR.TMP

This file corresponds exactly to the one described in the preceding paragraph, except that it is read by the FORTRAN IV compiler, F40.

### 6.2.5 001PIP.TMP

This file is written by COMPIL and read by PIP. It contains ordinary PIP commands to implement the DIRECTORY, LIST, TYPE, RENAME, and DELETE commands.

### 6.2.6 001CRE.TMP

This file is written by COMPIL and read by CREF. It contains commands to CREF corresponding to each file which has produced a CREF listing on the disk.

COMPIL also reads this file, if it exists, each time a new CREF listing is generated, to prevent multiple requests for the same file, and to prevent discarding other requests which may not yet have been listed.

The commands are of the form:

←PROG1
←PROG2

etc. That is, the default devices are assumed by CREF.

### 6.2.7 001EDT.TMP

This file is written by COMPIL for each EDIT-class command, and is read by either LINED or TECO.

For the commands MAKE or CREATE, it contains the command

Sfile.ext (ALTMODE)

For the commands TECO or EDIT, it contains the command

Sfile.ext (RETURN) (LINEFEED)

### 6.2.8 Other Uses of these Files

These command files are read by the CUSPs when the CUSPs are started at C(JOBSA) + 1, as described below. Thus it is possible to utilize this feature of the CUSPs without using COMPIL.

For example, the CCL system does not allow putting listings from the translators on any device except the disk. Since this may be undesirable at times, one could place MACRO commands in a file named 001MAC.TMP, directing the listing files to a magnetic tape, and then have MACRO read this file for its commands.

## 6.3 The Run Process

The cusps have been modified to allow sequential running of cusps without user intervention, and reading of commands from disk files. The command to run another program is

NAME!

This causes the program named NAME to be read from SYS, and started at C(JOBSA) +1.

Using this starting address causes the CUSP to read commands from the command file, rather than the user's console.

## 6.4 Error Count

The translators and LOADER cause location JOBERR (42) to be incremented for each error they detect. Then this location is tested by the LOADER when an EXECUTE command causes it to attempt execution. If it finds JOBERR non-zero, it types "EXECUTION DELETED" rather than starting the program.

## 7. ERRORS

The COMPIL program makes various checks to see that the CCL commands are consistent, and (in the COMPILE-class commands) that the required source files exist. There are also some "impossible" errors, which imply some sort of system failure.

The error messages generated by the COMPIL program are listed below. In addition, the programs called by COMPIL may generate error messages, such as assembly or compilation errors. Those typeouts are described in the relevant manuals for MACRO, FORTRAN, LOADER, PIP, and CREF. The message "EXECUTION DELETED" will be produced by the LOADER if either it or one of the translators has detected any errors.

Errors detected by COMPIL cause an EXIT to the Monitor.

Table 7-1
COMPIL Program Error Messages

| Message | Meaning |
|---|---|
| SYNTAX AND TYPING ERRORS | |
| COMMAND ERROR | COMPIL cannot decipher the command. |
| UNRECOGNIZABLE SWITCH | An embiguous or undefined word followed a slash ("/"). |
| PROCESSOR CONFLICT | Use of the "+" construction has resulted in a mixture of source languages. |
| RESOURCE AND FILE FAILURES | |
| DEVICE NOT AVAILABLE | The specified device could not be referenced. |
| NO SUCH FILE - file.ext | The specified file could not be found. This file may be one specified as a source, or one required for COMPIL's operation. |
| NOT ENOUGH CORE | The system cannot supply sufficient core for COMPIL's to use for buffers, or to read in a cusp. |

Table 7-1 (Cont)
COMPIL Program Error Messages

| Message | Meaning |
|---|---|
| COMMAND COMPLEXITY TOO GREAT | |
| TOO MANY SWITCHES or TOO MANY NAMES | The command complexity exceeds table space in the COMPIL program. |
| NESTING TOO DEEP | The @ construction has exceeded a depth of nine. This may be due to a loop of @ files. |
| "IMPOSSIBLE" ERRORS | |
| DISK NOT AVAILABLE | Cannot reference device DSK. |
| INPUT ERROR | An I/O error occurred while reading a temporary command file from the disk |
| LINKAGE ERROR | An I/O error occurred while reading a cusp from SYS. |
| FILE IN USE OR PROTECTED | A temporary command file could not be entered in the user's UFD. |
| OUTPUT ERROR | An I/O error occurred while writing a temporary command file on the disk. |

## 8. SUMMARY

The various components of the CCL commands are summarized here, with references to their descriptions in the body of this document.

### 8.1 Commands

| Commands | Minimal Abbreviation[1] | |
|---|---|---|
| COMPILE | COM | 4.1.2 |
| CREATE | CREA | 4.2.2.2 |
| CREF | CREF | 4.3.6 |
| DEBUG | DEB | 4.1.5 |
| DELETE | DEL | 4.3.4 |
| DIRECTORY | DI | 4.3.3 |
| EDIT | ED | 4.2.2.1 |
| EXECUTE | EX | 4.1.4 |
| LIST | LI | 4.3.2 |

[1] These abbreviations may change if additional Monitor commands are added.

| Commands | Minimal Abbreviation[1] | |
|---|---|---|
| LOAD | LOA | 4.1.3 |
| MAKE | M | 4.2.3.2 |
| RENAME | REN | 4.3.5 |
| TECO | TE | 4.2.3.1 |
| TYPE | TY | 4.3.1 |

## 8.2 Compile-Switches

| | |
|---|---|
| Compile-Switches | 4.1.2.5 |
| C | 4.1.2.6 |
| COMPILE | 4.1.2.8 |
| CREF | 4.1.2.6 |
| F | 4.1.2.7 |
| FORTRAN | 4.1.2.7 |
| L | 4.1.2.6 |
| LIBRARY | 4.1.3.2 |
| LIST | 4.1.2.6 |
| M | 4.1.2.7 |
| MACRO | 4.1.2.7 |
| MAP | 4.1.3.2 |
| N | 4.1.2.6 |
| NOCOMPILE | 4.1.2.8 |
| NOLIST | 4.1.2.6 |
| NOSEARCH | 4.1.3.2 |
| REL | 4.1.2.7 |

## 8.3 Special Characters

| | |
|---|---|
| . | 4.1.2.2 |
| , | 4.1.2.4 |
| [ ] | 4.1.2.3 |
| / | 4.1.2.5, 4.3.3.2 |
| % | 4.1.3.3 |
| = | 4.3.5, 5.3 |
| @ | 5.1 |
| ; | 5.1 |
| + | 5.2 |
| < > | 5.4 |
| ( ) | 5.5 |
| ! | 6.3 |

[1] These abbreviations may change if additional Monitor commands are added.

## 8.4 File-Naming Conventions

### 8.4.1 Temporary Files for COMPIL Program:

Files are named nnnxxx.TMP, where nnn is the user's job number in decimal, and xxx specifies the use of the file. Assuming job number 1, the files are:

| | |
|---|---|
| 001MAC.TMP | Passes commands to MACRO |
| 001FOR.TMP | Passes commands to FORTRAN IV |
| 001PIP.TMP | Passes commands to PIP |
| 001CRE.TMP | Passes commands to CREF |
| 001EDT.TMP | Passes commands to LINED and TECO |
| 001SVC.TMP | Saves COMPILE-class commands |
| 001EDS.TMP | Saves EDIT-class commands |

### 8.4.2 Standard Meanings for File Extensions:

| | |
|---|---|
| .TMP | Temporary file |
| .MAC | Source file in MACRO language |
| .F4 | Source file in FORTRAN IV language |
| .LST | Listing or CREF data |
| .REL | Relocatable binary file |
| .CMD | Command file, for @ construction |
| .SAV | Core dump, from SAVE command |
| blank | Unspecified ASCII text file |

# ADDENDUM II

## Re-entrant User Capability for PDP-10 Time Sharing System

100-118-005-01

"RENMON.MAN"

4 SERIES

# RE-ENTRANT USER CAPABILITY FOR PDP-10 TIME SHARING SYSTEM

T. HASTINGS 30 JUN 1969

V002

CHANGES LISTED IN ORDER OF MOST RECENT FIRST
FIRST VERSION IN WHICH CHANGE APPEARED WILL BE PUT ON EVERY LINE SO CHANGED.

CHANGED FROM VERSION V001(-01) 11 APR 69 TO VERSION V002(-01) 30 JUN 69

21. WRITING REENTRANT USER PROGRAMS.
ADDED SECTION B HOW TO WRITE PROGRAM AS ONE SOURCE FILE, WHICH IS MORE CONVENIENT.

TABLE OF CONTENTS

1. BACKGROUND

2. MOTIVATION

3. DESIGN GOALS

4. DEFINITIONS

5. RESTRICTIONS ON UUOS AND MONITOR COMMANDS

6. SAVE,SSAVE COMMANDS

7. MODIFYING SHARED SEGMENTS DURING EXECUTION

8. SET USER-MODE WRITE PROTECT UUO (SETUMP) (36)

8.1 RESET UUO

9. THE ALLOCATION OF VIRTUAL CORE

9.1 CORE UUO

9.2 CORE COMMAND

10. GET COMMAND

11. REMAP UUO (37)

12. RUN UUO (35)

13. GETSEG UUO (40)

14. SPY UUO (42)

15. SUPERCEDING SHARED SEGMENTS

16. USING THE LINKING LOADER

17. ASSEMBLER PSEUDOP-HISEG

18. MODIFICATIONS TO LINKING LOADER

19. USING DDT

20. JOB DATA AREA (JOBDAT)

21. WRITING REENTRANT USER PROGRAMS

22. MONITOR ALLOCATION OF SWAPPING SPACE

23. MONITOR ALLOCATION OF PHYSICAL CORE

24. MONGEN DIALOG QUESTIONS
ONCE ONLY DIALOG QUESTIONS

25. GLOSSARY

## 1. BACKGROUND

A GLOSSARY OF UNFAMILIAR TERMS OR TERMS WHOSE MEANINGS MAY BE UNIQUE WITHIN PDP-10 LITERATURE MAY BE FOUND AT THE END OF THE DOCUMENT, SORTED IN ORDER OF THEIR APPEARANCE IN THE TEXT. WHEREVER POSSIBLE TERMS HAVE BEEN SELECTED TO CORRESPOND TO COMMON USAGE IN THE TECHNICAL LITERATURE. ONE WORD IN PARTICULAR, THE WORD "RE-ENTRANT", DESERVES SOME COMMENT: RE-ENTRANT IS AN ADJECTIVE REFERRING TO A SEQUENCE OF INSTRUCTIONS WHICH MAY BE ENTERED BY MORE THAN ONE USER PROCESS AT A TIME. THUS A SINGLE COPY OF A REENTRANT PROGRAM MAY BE SHARED BY A NUMBER OF USERS AT THE SAME TIME, THEREBY INCREASING SYSTEM ECONOMY. AS SUCH, THE PDP-10 TIME SHARING MONITOR HAS ALWAYS BEEN LARGELY "RE-ENTRANT". NEVERTHELESS, PDP-10 LITERATURE NOW USES THE PHRASE "REENTRANT MONITOR" TO REFER TO THE 4 SERIES MONITOR WHICH PROVIDES THE RE-ENTRANT USER PROGRAM CAPABILITY, AS DESCRIBED IN THIS DOCUMENT.

THE OLD ONE RELOCATION REGISTER HARDWARE ON PDP-6'S AND EARLY PDP-10'S REQUIRED THAT A USER AREA BE A SINGLE SEGMENT OF LOGICAL AND PHYSICAL CORE. THIS MEANT THAT EACH USER HAD TO HAVE A SEPARATE COPY OF A PROGRAM EVEN THOUGH A LARGE PART OF IT WAS THE SAME AS FOR OTHER USERS. THE NEW TWO RELOCATION REGISTER HARDWARE PERMITS A USER AREA TO BE DIVIDED INTO TWO SEGMENTS OF LOGICAL AND PHYSICAL CORE, THE MONITOR WILL ALLOW ONE OF THE SEGMENTS OF EACH USER AREA TO BE THE SAME AS ONE OR MORE OTHER USERS, SO THAT ONLY ONE PHYSICAL COPY OF SUCH A SHARED SEGMENT NEED EXIST NO MATTER HOW MANY USERS ARE USING IT. THIS SHARED SEGMENT WILL USUALLY BE WRITE PROTECTED BY HARDWARE TO GUARRANTEE THAT IT IS NOT ACCIDENTALLY MODIFIED. A PROGRAM COMPOSED OF A SHARABLE AND A NON-SHARABLE SEGMENT IS SAID TO BE RE-ENTRANT. FOR HARDWARE DESCRIPTION SEE PDP-10 REFERENCE MANUAL DEC-10 HGAA-D, DECEMBER 1968 SUPPLEMENT OR PROGRAMMING DEPARTMENT MEMO 100-118-001-03

THIS MANUAL SUPERSEDES PROGRAMMING DEPARTMENT MEMOS:

| | |
|---|---|
| 100-118-002-02 | PDP-10 REENTRANT SOFTWARE |
| 100-118-003-00 | WRITING REENTRANT PROGRAMS |
| 100-118-004-00 | REENTRANT SOFTWARE - MODIFICATIONS AND ADDITIONS |

## 2. MOTIVATION

THE MOTIVATION FOR ADDING A RE-ENTRANT CAPABILITY IS TO INCREASE THE NUMBER OF USERS WHICH CAN BE HANDLED BY A GIVEN SIZE TIME-SHARING CONFIGURATION. THIS IS ACCOMPLISHED BY MORE EFFECTIVE USE OF THE FOLLOWING SYSTEM RESOURCES:

1) MORE EFFECTIVE USE OF CORE MEMORY SINCE ONLY ONE COPY OF A SHARED SEGMENT WILL EXIST FOR THE ENTIRE SYSTEM, INSTEAD OF ONE COPY FOR EACH USER USING THE SEGMENT.

2) MORE EFFECTIVE USE OF THE SWAPPING STORAGE SINCE ONE COPY OF A SHARED SEGMENT CAN BE SHARED BY ALL USERS USING IT.

3) MORE EFFECTIVE USE OF THE SWAPPING CHANNEL SINCE A SHARED SEGMENT WILL ONLY BE READ ONCE NO MATTER HOW MANY USERS IN CORE ARE USING IT.

4) MORE EFFECTIVE USE OF THE SWAPPING CHANNEL SINCE MOST SHARED SEGMENTS WILL NOT BE MODIFIED DURING EXECUTION SO THAT THEY WILL NOT NEED TO BE WRITTEN BACK ONTO SWAPPING STORAGE.

5) MORE EFFECTIVE USE OF THE STORAGE CHANNEL SINCE A SHARED SEGMENT WILL ONLY BE READ ONCE FROM THE STORAGE DEVICE WHEN THE PROGRAM IS ACCESSED FOR THE FIRST TIME BY ANYONE. THEREAFTER IT WILL EXIST ON THE FASTER SWAPPING STORAGE.

## 3. INITIAL DESIGN GOALS

1. THE CHANGES FOR RE-ENTRANT SOFTWARE WILL NOT INVALIDATE ANY EXISTING SOFTWARE. THIS MEANS THAT ALL

   1) SAVED FILES (.DMP AND .SAV) WILL STILL RUN AS USUAL

   2) RELOCATABLE BINARY (.REL) WILL STILL LOAD AND RUN AS USUAL

   3) MACRO SOURCE (.MAC) WILL ASSEMBLE, LOAD AND RUN AS USUAL

   4) FORTRAN SOURCE (.F4) WILL COMPILE, LOAD AND RUN AS USUAL

   THIS MEANS THAT THE MONITOR MUST HANDLE TWO TYPES OF PROGRAMS:

   1) OLD-STYLE NON-RE-ENTRANT PROGRAMS

   2) NEW-STYLE RE-ENTRANT PROGRAMS

2. THE RE-ENTRANT MODIFICATION WILL BE USEFUL FOR THE 10/40 SYSTEM WITH DECTAPE AND NO DISK, AS WELL AS THE 10/40 SYSTEM WITH DISK AND THE 10/50 SWAPPING SYSTEM.

3. MINIMIZE THE CHANGES TO CUSP'S REQUIRED TO:

   1) MAKE THEM RE-ENTRANT THEMSELVES

   2) MAKE THEM GENERATE OTHER RE-ENTRANT PROGRAMS

4. MAKE THE TERMINAL OPERATING CHARACTERISTICS OF RE-ENTRANT PROGRAMS BE THE SAME AS THE CURRENT SYSTEM. THUS THE USER DOES NOT NEED TO KNOW WHETHER A PROGRAM HE IS RUNNING IS RE-ENTRANT OR NOT. (OBVIOUSLY A PROGRAMMER WILL HAVE TO KNOW WHEN HE WRITES THE PROGRAM.)

5. WRITING RE-ENTRANT PROGRAMS WILL BE EASY ENOUGH SO THAT CUSTOMERS WILL WANT TO WRITE THEM. ALSO CUSTOMERS MUST BE ABLE TO SHARE RE-ENTRANT PROGRAMS WHICH ARE IN THEIR OWN DIRECTORIES. THUS SHARING WILL NOT BE RESTRICTED TO PROGRAMS IN THE SYSTEM CUSP DIRECTORY.

6. WHEN HE WRITES HIS PROGRAM, A PROGRAMMER CAN CHOOSE WHETHER OR NOT A GET (ALSO R, AND RUN) IS TO INTIALIZE THE IMPURE SEGMENT FROM SECONDARY STORAGE. FOR EXAMPLE, MACRO WILL WANT TO HAVE THE INITIAL SYMBOL TABLE LOADED INTO THE IMPURE SEGMENT, WHILE TECO WILL PROBABLY WANT NOTHING LOADED INTO IT AND WILL CLEAR THE IMPURE SEGMENT ITSELF.

7. MAKE IT EASY TO MODIFY THE MONITOR TO HANDLE RE-ENTRANT USER PROGRAMS.

8. LIMIT A USER AREA TO ONLY ONE SHARABLE SEGMENT AT A TIME. IN ADDITION THE SEGMENT WILL EXIST IN ITS ENTIRETY IN BOTH LOGICAL AND PHYSICAL CORE AND ON THE SWAPPING DEVICE. ALSO NOT MAKE FORTRAN OBJECT PROGRAMS BE SHARABLE. HOWEVER, WE WILL NOT DO ANYTHING TO PREVENT THIS POSSIBILITY A FEW YEARS FROM NOW. DDT NOT SHARABLE BY ITSELF INITIALLY EITHER. MAYBE EVENTUALLY AN INVISIBLE, SHARABLE DDT.

9. RE-ENTRANT PROGRAMS WILL CONTINUS TO BE SELF INITIALIZING, AS ARE THE EXISTING CUSP'S. IT MUST ALWAYS BE POSSIBLE TO TYPE CONTROL C START AT ANY TIME AND HAVE THE PROGRAM START OVER CORRECTLY. THIS IS GOOD PROGRAMMING PRACTICE, MAKES DEBUGGING EASIER AND MEANS THAT THE MONITOR WILL NOT HAVE TO DO I/O TO INITIALIZE

   IMPURE SEGMENT ON A START COMMAND.

10. RE-ENTRANT PROGRAMS NEED NOT BE DEBUGGED IN ORDER TO BE SHARED. THE SYSTEM WILL NOT FAIL, WHEN ONE USER ENCOUNTERS A BUG WHILE SHARING A SEGMENT. IN FACT, THE OTHER USERS WILL BE UNAFFECTED BY A BUG ENCOUNTERED BY ONLY ONE USER.

11. AS SOON AS A NEW RE-ENTRANT PROGRAM IS CREATED BY ANY USER WHILE THE SYSTEM IS RUNNING, IT WILL BE ABLE TO BE SHARED BY ALL USERS SO AUTHORIZED BY THE OWNER OF THE PROGRAM. THE MONITOR WILL NOT HAVE TO BE REASSEMBLED, RELOADED WITH THE LINKING LOADER, OR RESTARTED IN ORDER TO START SHARING THE NEW PROGRAM. USER'S WHO ARE IN THE PROCESS OF SHARING THE OLDER VERSION WILL NOT BE AFFECTED UNTIL THEY ARE THROUGH.

12. (THIS DESIGN GOAL HAS NOT BEEN MET YET) ON SYSTEMS EMPLOYING THE SMALL, FAST BURROUGHS DISK FOR BOTH SWAPPING AND FOR STORAGE, THE SAVED FILE AND THE SHARED SEGMENT WILL BE SAME INSTEAD OF DIFFERENT COPIES, THUS USING THE DISK MORE EFFICIENTLY. THIS MEANS THAT THE FORMAT FOR SAVED FILES AND SHARED SEGMENTS WILL PROBABLY HAVE TO BE THE SAME.

13. AVOID DESIGN WHICH CLOSES THE DOOR TO FUTURE EXTENSIONS AND IMPROVEMENTS WHEREVER POSSIBLE.

14. MAKE IT EASY FOR ALL PROGRAMMERS TO WRITE RE-ENTRANT PROGRAMS IN SUCH A WAY THAT THE SOURCES AND BINARY ARE THE SAME FOR A RE-ENTRANT VERSION FOR PDP-10 AND A NON-RE-ENTRANT VERSION FOR PDP-6. THE DECISION TO PRODUCE A NON-RE-ENTRANT VERSION WILL BE POST-PONED TO LINKING LOAD TIME.

## 4. DEFINITIONS

A SEGMENT IS A CONTIGUOUS REGION OF A USER'S CORE IMAGE WHICH THE MONITOR MAINTAINS AS A CONTIGUOUS UNIT IN PHYSICAL CORE AND/OR AS A POSSIBLY FRAGMENTED UNIT ON THE SWAPPING DEVICE. A SEGMENT MAY CONTAIN INSTRUCTIONS OR DATA OR BOTH. IT IS THE TASK OF THE MONITOR TO DETERMINE THE ALLOCATION AND MOVEMENT OF SEGMENTS IN CORE AND THE SWAPPING DEVICE. A PROGRAM OR USER JOB IS COMPOSED OF ONE OR TWO SEGMENTS.

A SHARABLE SEGMENT IS A SEGMENT WHICH IS THE SAME FOR ALL USERS, SO THAT THE MONITOR KEEPS ONLY ONE COPY FOR THE SYSTEM IN CORE AND/OR ON THE SWAPPING DEVICE, NO MATTER HOW MANY USERS ARE USING IT. ON THE OTHER HAND A NON-SHARABLE SEGMENT IS A SEGMENT WHICH IS DIFFERENT FOR EACH USER IN CORE AND/OR ON THE SWAPPING DEVICE.

THE TWO RELOCATION AND PROTECTION REGISTERS OF THE PDP-10 PERMIT A USER PROGRAM TO BE COMPOSED OF ONE OR TWO SEGMENTS AT ANY POINT IN TIME. THESE REGISTERS DIVIDE A USER'S CORE INTO TWO PARTS, THE REQUIRED LOW SEGMENT STARTING AT USER 0 AND THE OPTIONAL HIGH SEGMENT SEGMENT STARTING AT USER 400000 OR THE END OF THE LOW SEGMENT WHICHEVER IS GREATER. THE LOW SEGMENT ALWAYS CONTAINS THE USERS ACCUMULATORS, JOB DATA AREA (JOBDAT), INSTRUCTIONS AND/OR DATA, I/O BUFFERS, AND DDT SYMBOLS. THUS A USER CORE IMAGE IS COMPOSED OF A LOW SEGMENT FROM 1K TO 256K WORDS IN MULTIPLES OF 1K AND A HIGH SEGMENT FROM 0K TO 128K WORDS, ALSO IN MULTIPLES OF 1K. A HIGH SEGMENT MAY BE SHARABLE OR NON-SHARABLE WHILE A LOW SEGMENT IS ALWAYS NON-SHARABLE. THE HIGH SEGMENT ALSO MAY OR MAY NOT BE WRITE LOCKED.

THERE ARE JUST THREE TYPES OF USER PROGRAMS, RE-ENTRANT PROGRAMS, ONE SEGMENT NON-RE-ENTRANT PROGRAMS, AND TWO SEGMENT NON-RE-ENTRANT PROGRAMS. A RE-ENTRANT PROGRAM IS ALWAYS COMPOSED OF TWO SEGMENTS, A LOW SEGMENT WHICH USUALLY CONTAINS JUST DATA AND A SHARABLE (HIGH) SEGMENT WHICH USUALLY CONTAINS INSTRUCTIONS AND CONSTANTS.

THE LOW SEGMENT IS SOMETIMES LOOSELY REFERRED TO AS THE IMPURE SEGMENT AND THE SHARABLE HIGH SEGMENT IF WRITE PROTECTED IS CALLED THE PURE SEGMENT. HOWEVER, IMPURE SUGGESTS MODIFIABLE RATHER THAN NON-SHARABLE AND PURE SUGGEST WRITE PROTECTED RATHER THAN SHARABLE. HENCE, A HIGH SEGMENT CAN BE PURE OR IMPURE AND SHARABLE OR NON-SHARABLE WHILE A LOW SEGMENT IS ALWAYS IMPURE AND NON-SHARABLE.

A ONE SEGMENT NON-RE-ENTRANT PROGRAM IS COMPOSED OF A SINGLE LOW SEGMENT CONTAINING INSTRUCTIONS AND DATA. THIS TYPE CORRESPONDS TO USER PROGRAMS WRITTEN FOR A MACHINE WITH ONLY A SINGLE RELOCATION AND PROTECTION REGISTER. A TWO SEGMENT NON-RE-ENTRANT PROGRAM IS COMPOSED OF A LOW SEGMENT AND A NON-SHARABLE HIGH SEGMENT. THIS TYPE OF PROGRAM IS USEFUL FOR THE RARE CASES WHEN THERE IS REQUIREMENT FOR TWO FIXED ORIGINED DATA AREAS TO GROW AND SHRINK INDEPENDENTLY DURING EXECUTION.

VIRTUAL ADDRESSING SPACE OF THE 3 TYPE OF PROGRAMS:

| RE-ENTRANT | NON-RE-ENTRANT | |
|---|---|---|
| | ONE SEGMENT | TWO SEGMENT |
| Ø | Ø | Ø |
| Ø-MAXL | Ø-MAXL | Ø-MAXL |
| NON-SHARABLE | NON-SHARABLE | NON-SHARABLE |
| 4ØØØØØ-MAXH | | 4ØØØØØ-MAXH |
| SHARABLE | | NON-SHARABLE |

WHERE MAXL AND MAXH ARE THE HIGHEST LEGAL ADDRESS IN THE LOW AND HIGH SEGMENTS RESPECTIVELY.

A FILE IS A COLLECTION OF 36 BIT WORDS COMPRISING COMPUTER INSTRUCTIONS AND/OR DATA. A FILE CAN BE OF ARBITRARY LENGTH, LIMITED ONLY BY THE AVAILABLE SPACE ON THE DEVICE AND THE USERS MAXIMUM ALLOTMENT ON IT. A NAMED FILE IS A FILE WHICH IS UNIQUELY IDENTIFIED IN THE SYSTEM BY ITS FILE NAME (UP TO 6 CHARACTERS), AND THE DIRECTORY NAME (PROJECT, PROGRAMMER NUMBERS OF OWNER FOR DISKS; PHYSICAL DEVICE NAME FOR DECTAPE AND MAGTAPE) IN WHICH THE FILE NAME AND EXTENSION APPEAR. THE FILE NAME IS ARBITRARY AND IS SPECIFIED BY THE OWNER AT THE TERMINAL WHILE THE EXTENSION IS USUALLY ONE OF A SMALL NUMBER OF STANDARD NAMES WHICH IDENTIFIES THE TYPE OF INFORMATION IN THE FILE AND IS USUALLY SPECIFIED BY PROGRAM. A NAMED FILE MAY BE WRITTEN BY A USER PROGRAM IN BUFFERED OR DUMP MODE OR A MIXTURE AND MAY BE READ AND/OR MODIFIED SEQUENTIALLY OR RANDOMLY WITH BUFFERED OR DUMP MODE INDEPENDENT OF HOW IT WAS WRITTEN. NAMED FILES ARE STORED ON THE STORAGE DEVICE WHICH MAY BE THE DISK, AND/OR

DECTAPE. FURTHERMORE, EACH NAMED FILE HAS CERTAIN ACCESS PRIVILEGES ASSOCIATED WITH IT WHICH SAY WHICH USERS CANNOT READ THE FILE, WRITE THE FILE, OR CHANGE THE ACCESS PRIVILEGES OF THE FILE. FOR PURPOSES OF SIMPLICITY THE UNIVERSE OF USERS IS DIVIDED INTO THREE GROUPS: THE OWNER OF THE FILE, THE OTHER USERS IN HIS PROJECT, AND THE REST OF THE USERS.

FILES AND SEGMENTS HAVE CERTAIN SIMILARITIES AND DIFFERENCES. BOTH ARE NAMED, ONE DIMENSIONAL ARRAYS OF 36 BIT WORDS. A FILE CAN BE ARBITRARILY LONG UP TO SIZE OF DISK OR DECTAPE, WHILE THE ENTIRITY OF A SEGMENT MUST FIT INTO PHYSICAL CORE. BOTH MAY BE SHARED FOR READING, HOWEVER ONLY ONE USER IS ALLOWED TO RECREATE OR UPDATE A FILE AT A TIME, WHILE MANY USER'S COULD SHARE A SEGMENT FOR WRITTING (PROVIDED THEY ESTABLISH A WELL DEFINED INTERLOCK DISCIPLINE AND USE THE SLEEP UUO WHEN BLOCKED).

ALTHOUGH USERS CAN SHARE NAMED FILES CONCURRENTLY, THE SHARING IS MUCH DIFFERENT THAN THE SHARING OF SEGMENTS. WHEN MANY USERS ARE READING FROM THE SAME OR DIFFERENT PORTIONS OF A FILE AT THE SAME TIME, EACH USER IS GIVEN HIS OWN COPY OF THE PORTION OF THE FILE HE IS READING (IT IS READ INTO HIS LOW SEGMENT VIA AN INPUT UUO). A FILE MAY BE READ,

CREATED, RECREATED OR UPDATED. A FILE IS CREATED IF NO FILE BY THE SAME NAME EXISTED WHEN THE FILE WAS OPENED FOR WRITING. A FILE IS RECREATED IF ANOTHER FILE BY THE SAME NAME ALREADY EXISTED. A PORTION OF A CREATED OR RECREATED FILE MAY BE SUBSEQUENTLY UPDATED BY MODIFYING (IN PLACE) OF ONE OR MORE OF THE BLOCKS OF THE FILE. OTHER USERS MAY BE READING A FILE WHILE ONE USER IS RECREATING IT. THE OLDER VERSION OF THE FILE IS DELETED ONLY WHEN ALL THE READERS HAVE FINISHED. ONLY ONE USER CAN OPEN A FILE FOR RECREATION AND UPDATING AT A TIME, SUBSEQUENT USERS GET AN ERROR RETURN. THUS A FILE EXISTS ON THE STORAGE DEVICE AND PIECES OF IT AND OTHER FILES CAN EXIST IN DIFFERENT PARTS OF THE LOW SEGMENT OF ONE OR MORE USERS.

A SEGMENT DIFFERS FROM A FILE IN THAT IT NEVER EXISTS ON THE STORAGE DEVICE; IT EXISTS ONLY IN CORE OR ON THE SWAPPING DEVICE. FUTHERMORE, A SEGMENT ALWAYS EXISTS IN ITS ENTIRETY AS A CONTIGUOUS UNIT. IN ORDER TO SAVE A SEGMENT SO THAT IT CAN BE RECALLED AND RUN AT A LATER TIME, THE USER USES THE SAVE (SSAVE) CONSOLE COMMAND WHICH WRITES A COPY OF ONE OR BOTH SEGMENTS ONTO THE STORAGE DEVICE AS NAMED FILES. TO RECALL THE PROGRAM LATER, THE USER TYPES THE GET, R, OR RUN COMMANDS, RUN OR GETSEG UUOS WHICH INITIALIZE ONE OR BOTH SEGMENTS FROM THE APPROPRIATE NAMED FILES. THESE COMMANDS MUST KNOW WHETHER OR NOT A HIGH SEGMENT IS SHARABLE. A HIGH SEGMENT WILL BE SHARABLE, IF THE FILE WHICH INITIALIZED IT HAD EXTENSION .SHR. A FILE EXTENSION OF .HGH WILL INDICATE THAT THE HIGH SEGMENT IS TO BE NON-SHARABLE.

## 5. RESTRICTION ON UUO'S AND MONITOR COMMANDS

THERE WILL BE A NUMBER OF RESTRICTIONS ON THE INVOLVEMENT OF A HIGH SEGMENT IN MONITOR UUO'S AND MONITOR COMMANDS. THESE RESTRICTIONS ARE MOTIVATED BY A DESIRE TO PROTECT NAIVE AND MALICIOUS USERS FROM CLOBBERING OTHERS WHILE SHARING SEGMENTS AND BY A DESIRE TO MINIMIZE MONITOR CHANGES TO HANDLE TWO SEGMENT PROGRAMS. HOWEVER, ALL UUO'S CAN BE EXECUTED FROM EITHER THE LOW OR HIGH SEGMENT. ALTHOUGH, SOME OF THEIR ARGUMENTS CANNOT BE IN OR REFER TO THE HIGH SEGMENT.

1) NO BUFFERS, BUFFER HEADERS, OR DUMP MODE COMMAND LISTS MAY EXIST IN THE HIGH SEGMENT FOR READING OR WRITING WITH ANY DEVICE. NO I/O WILL BE DONE INTO OR OUT OF THE HIGH SEGMENT EXCEPT SAVE, SSAVE COMMANDS.

2) NO STATUS, CALL, OR CALLI UUO WILL ALLOW A STORE INTO THE HIGH SEGMENT. THE EFFECTIVE ADDRESS OF LOOKUP, ENTER, INPUT, OUTPUT, AND RENAME UUOS MUST NOT BE IN THE HIGH SEGMENT (ADDRESS CHECK ERROR MESSAGE IF ATTEMPTED).

3) HOWEVER, AS A CONVENIENCE IN WRITING USER PROGRAMS, THE MONITOR WILL MAKE A SPECIAL CHECK SO THAT THE INIT UUO CAN BE EXECUTED FROM THE HIGH SEGMENT EVEN THOUGH THE CALLING SEQUENCE IS IN THE HIGH SEGMENT. AND THE MONITOR WILL ALSO ALLOW THE EFFECTIVE ADDRESS OF THE CALL UUO (CONTAINS THE SIXBIT MONITOR FUNCTION NAME) AND THE EFFECTIVE ADDRESS OF THE OPEN UUO (CONTAINS THE STATUS BITS, DEVICE NAME AND BUFFER HEADER ADDRESSES) TO BE IN THE HIGH SEGMENT. HOWEVER, THE BUFFER HEADERS THEMSELVES MUST BE IN THE LOW SEGMENT.

4) THE MONITOR COMMANDS E (EXAMINE USER AREA) AND D (DEPOSIT USER AREA) WILL WORK FOR BOTH LOW AND HIGH SEGMENTS, PROVIDED THE USER HAS THE ACCESS RIGHT TO READ(E) AND WRITE(D) THE HIGH FILE WHICH INITIALIZED THE HIGH SEGMENT. SEE MODIFYING SHARABLE SEGMENTS FOR DEFINITION OF ACCESS PRIVILEGES.

## 6. THE SAVE AND SSAVE COMMANDS

THE SAVE COMMAND WILL SAVE ANY USER PROGRAM (I.E., RE-ENTRANT, ONE SEGMENT NON-RE-ENTRANT, OR TWO-SEGMENT NON-RE-ENTRANT, AS ONE OR TWO FILES, SUCH THAT WHEN LOADED BY A GET, R, OR RUN COMMAND IT WILL BECOME A NON-RE-ENTRANT (I.E., NON-SHARABLE) PROGRAM. THUS A USER CAN ALWAYS SAVE THE PROGRAM HE IS RUNNING WITHOUT KNOWING WHETHER IT IS RE-ENTRANT OR NOT AND COME BACK LATER AND START IT UP AGAIN. HE NEED NOT WORRY THAT SOMEONE ELSE HAS REPLACED THE ORIGINAL FILE WITH A DIFFERENT VERSION.

IF THE JOB CONTAINS ONLY A LOW SEGMENT, SAVE WILL WRITE ONE FILE WITH EXTENSION OF .SAV, AS IN 3 SERIES MONITORS. HOWEVER IF THE JOB HAS TWO SEGMENTS, SAVE WILL WRITE THE HIGH SEGMENT WITH FILE EXTENSION OF .HGH AND THE LOW SEGMENT WITH FILE EXTENSION .LOW. THUS IT IS POSSIBLE TO HAVE BOTH A TWO SEGMENT AND A ONE SEGMENT

VERSION OF THE SAME PROGRAM (MACRO.HGH+ MACRO.LOW AND MACRO.SAV). LOW SEGMENT FILES WILL BE ŻERO COMPRESSED ON ALL DEVICES (DTA,MTA, AND DSK), BUT HIGH SEGMENT FILES WILL NOT BE SINCE THE HIGH SEGMENT MAY BE SHARED AT THE TIME OF SAVE AND SO CANNOT BE COMPRESSED. THUS SAVED FILES ARE JUST ORDINARY BINARY FILES ON ALL DEVICES AND SO CAN BE COPIED WITH PIP (USING/B SWITCH, OF COURSE). HOWEVER, FILES WHICH ARE TO BE LOADED BY TENDMP FROM DECTAPE MUST STILL BE WRITTEN USING SAVE OR SSAVE COMMAND SINCE TENDMP REQUIRES THAT THE FIRST BLOCK OF A FILE ALSO BE THE LOWEST. IN ORDER TO SAVE FILE SPACE, THE SAVE COMMAND WILL NOT WRITE THE ENTIRE HIGH SEGMENT. IT WILL ONLY WRITE UP THROUGH THE HIGHEST (RELATIVE TO HIGH SEGMENT ORIGIN) LOCATION LOADED BY LINKING LOADER AS SPECIFIED BY C(LH) OF JOBHRL(ANALOGOUS TO LH OF JOBSA FOR LOW SEGMENT). IF LH IS Ø (HIGH SEGMENT CREATED BY CORE OR REMAP UUO) OR DDT IS IN USE THE ENTIRE HIGH SEGMENT WILL BE WRITTEN, SO THAT PATCHES WILL BE SAVED.

IN ORDER TO SAVE FILE SPACE AND IO TIME ON GETS, IT IS POSSIBLE FOR MOST PROGRAMS TO BE WRITTEN SO THAT ONLY THE HIGH SEGMENT CONTAINS NON-ŻERO DATA. WHEN THIS IS THE CASE, SAVE WILL ONLY WRITE THE HIGH SEGMENT(EXTENSION,HGH). THE LINKING LOADER INDICATES TO THE SAVE COMMAND THAT NOTHING WAS LOADED ABOVE THE JOB DATA AREA (FIRST 14Ø LOCATIONS) IN THE LOW SEGMENT BY SETTING THE LH OF JOBCOR IN THE JOB DATA AREA TO THE HIGHEST LOCATION LOADED IN THE LOW SEGMENT WITH NON-ŻERO DATA. SEE SECTION ON WRITING REENTRANT USER PROGRAMS BELOW. HOWEVER THERE ARE A NUMBER OF LOCATIONS IN THE JOB DATA AREA WHICH NEED TO BE INITIALIŻED ON A GET EVEN THOUGH THERE IS NO OTHER DATA IN THE LOW SEGMENT. THE SAVE COMMAND COPIES THESE LOCATIONS INTO THE FIRST 1Ø (OCTAL) LOCATIONS OF THE HIGH SEGMENT, PROVIDED THAT THE HIGH SEGMENT IS NOT SHARABLE (SEE BELOW). THESE 1Ø LOCATIONS ARE REFERRED TO AS THE VESTIGIAL JOB DATA AREA. CONSEQUENTLY, THE LINKING LOADER WILL LOAD HIGH SEGMENT PROGRAMS STARTING AT 4ØØØ1Ø. SEE THE JOB DATA AREA SECTION BELOW FOR THE DESCRIPTION OF LOCATIONS SAVED IN THIS MANNER.

IN ORDER TO SAVE A PROGRAM SO THAT THE HIGH SEGMENT WILL BE SHARABLE ON SUBSEQUENT GETS, A NEW SAVE COMMAND HAS BEEN ADDED, CALLED SSAVE(S FOR SHARABLE). IT WORKS EXACTLY LIKE SAVE EXCEPT THAT, IF A HIGH SEGMENT EXISTS, IT WRITES A FILE WITH EXTENSION .SHR INSTEAD OF .HGH. A SUBSEQUENT GET WILL CAUSE THE HIGH SEGMENT TO BE SHARABLE. IF THE PROGRAM DOES NOT HAVE A HIGH SEGMENT, NO ERROR MESSAGE IS GIVEN. THIS ALLOWS THE USER TO ALWAYS USE THE SSAVE COMMAND WHEN SAVING CUSPS WITHOUT HAVING TO KNOW WHICH ONES ARE SHARABLE.

IN ORDER TO PREVENT USER CONFUSION, SAVE AND SSAVE DELETE A PREVIOUS FILE WITH ,SHR OR .HGH THE EXTENSION WHICH THE OTHER COMMAND WOULD HAVE CREATED AFTER A SUCCESSFUL WRITE. THUS SAVE DELETES .SHR AND SSAVE DELETES

.HGH. BOTH COMMANDS ALSO DELETE A FILE WITH EXTENSION .LOW IF THE HIGH SEGMENT WAS WRITTEN AND THE LOW SEGMENT WAS NOT.

THE FOLLOWING TABLE COMPARES SAVE AND SSAVE:

| SEGMENTS WRITTEN | | SAVE | | SSAVE | |
|---|---|---|---|---|---|
| LOW | HIGH | WRITES | DELETES | WRITES | DELETES |
| X | | SAV | | SAV | |
| X | X | LOW+HGH | SHR | LOW+SHR | HGH |
| | X | HGH | SHR | SHR | HGH |

THE REGULAR ACCESS RIGHTS OF THE SAVED FILE INDICATE WHETHER A USER CAN DO A GET, R, OR RUN COMMAND. THE GET, R, AND RUN COMMANDS WILL ASSUME THAT THE USER WANTS TO READ (I.E., EXECUTE BUT NOT MODIFY) THE HIGH SEGMENT INDEPENDENT OF THE ACCESS RIGHTS OF THE FILE USED TO INITIALIZE THE SEGMENT. THEREFORE THE MONITOR WILL ALWAYS ENABLE THE HARDWARE USER-MODE WRITE PROTECT TO PREVENT THE USER PROGRAM FROM STORING INTO THE SEGMENT INADVERTENTLY. THE PROGRAM MAY TURN OFF THE WRITE PROTECT IF IT WISHES. SEE SETUWP UUO BELOW.

IN ORDER TO DEBUG A RE-ENTRANT CUSP WHICH IS IN THE SYSTEM DIRECTORY, THE USER SHOULD MAKE A PRIVATE, NON-SHARABLE COPY, RATHER THAN MODIFYING THE SHARED VERSION AND POSSIBLY CAUSING OTHER USERS HARM. A PRIVATE, NON-SHARABLE COPY CAN BE MADE IN THREE EASY STEPS:

1. GET SYS CUSP

2. SAVE DEV CUSP

3. GET DEV CUSP

STEP 2 WRITES A FILE IN THE USER DIRECTORY MARKED AS NON-SHARABLE. HOWEVER THE HIGH SEGMENT IN THE USER'S ADDRESSING SPACE REMAINS SHARABLE. STEP 3 OVERLAYS THE SHARABLE PROGRAM WITH THE NON-SHARABLE ONE FROM THE USER'S DIRECTORY. THEN THE USER CAN MAKE PATCHES AND INSERT BREAKPOINTS WHILE OTHER USERS SHARE THE VERSION IN THE SYSTEM DIRECTORY. SINCE THE SEGMENT NAME INCLUDES THE DIRECTORY NAME, THE MONITOR WILL KEEP THE SHARED AND THE NON-SHARED VERSIONS SEPARATE FROM EACH OTHER. A SHARABLE PROGRAM MAY BE SUPERCEEDED IN THE DIRECTORY AT ANY TIME BY USING AN SSAVE COMMAND. THE MONITOR WILL CEAR THE HIGH SEGMENT NAME IN ITS TABLE OF STORABLE SEGMENTS IN USE, BUT WILL NOT REMOVE THE SEGMENT FROM OTHER USER'S ADDRESSING SPACE. ONLY USERS DOING A GET,R,RUN COMMAND OR RUN,GETSEG UUO WILL GET THE NEW SHARABLE VERSION.

AN OBSCURE RESTRICTION EXITS WHEN SAVING A SHARABLE PROGRAM WITH ONLY A HIGH FILE WITH EITHER SAVE OR SSAVE, THE MONITOR WILL NOT MODIFY THE VESTIGIAL JOB DATA AREA UNLESS THE USER HAS WRITE PRIVILEGES TO THE FILE WHICH ORIGINALLY INITIALILIZED THE SHARED SEGMENT. OTHERWISE UNAUTHORIZED USERS COULD MODIFY THE FIRST 10 WORDS OF A SHARED SEGMENT, THIS MEANS THAT EVEN THOUGH THE USER CHANGES THE STARTING ADDRESS(JOBSA), THE VERSION NUMBER(JOBVER), THE CORE SIZE FOR LOW SEG ON GET(RH OF JOBCOR SET BY SAVE OR GET THIRD ARG) OR THE HIGHEST LOCATION IN LOW SEGMENT LOADED WITH NON-ZERO DATA (LH OF JOBCOR) ORR THE REENTER ADDRESS(RH JOBREN) BY USING THE DEPOSIT COMMAND OR SAVE 3RD ARG, IT WILL HAVE NO EFFECT ON SAVE AND SUBSEQUENT GET.

THIS RESTRICTION DOES NOT EXIST IF A LOW FILE IS WRITTEN TOO, SINCE GET READS LOW FILE AFTER HIGH FILE, SO THAT THE REAL JOB DATA AREA LOCATIONS ARE SET FROM THE LOW FILE.

## 7. MODIFYING SHARED SEGMENTS DURING EXECUTION

USUALLY A HIGH SEGMENT WILL BE WRITE PROTECTED, HOWEVER, IT IS POSSIBLE FOR A USER PROGRAM TO TURN OFF UWP (USER WRITE PROTECT) OR TO INCREASE OR DECREASE CORE ASSIGNMENTS OF A SHARED SEGMENT USING THE SETUWP AND CORE UUO'S. THESE TWO UUO'S WILL BE LEGAL FROM EITHER HIGH OR LOW SEGMENT PROVIDED THAT THE SHARABLE PROGRAM HAS NOT BEEN "MEDDLED WITH". THIS MEANS THAT EVEN THE MALICIOUS USER CAN HAVE THE PRIVILEGE OF RUNNING SUCH A PROGRAM EVEN THOUGH HE DOES NOT HAVE THE ACCESS RIGHTS TO MODIFY THE FILE USED TO INITIALIZE THE SHARABLE SEGMENT.

MEDDLING IS DEFINED AS ANY OF THE FOLLOWING,EVEN IF THE USER HAS PRIVILEGES TO WRITE THE FILE WHICH INITIALIZES THE HIGH SHARABLE SEGMENT:
(IT IS NOT CONSIDERED MEDDLING TO DO ANY OF THE FOLLOWING TO A NON-SHARABLE PROGRAM <ONE OR TWO SEGMENT>)

1. START AND CSTART COMMANDS WITH AN ARGUMENT.
2. DEPOSIT COMMAND IN EITHER LOW OR HIGH SEGMENT.
3. RUN UUO WITH ANYTHING BUT 0 OR 1 IN LH OF AC (STARTING ADDRESS INCREMENT)
4. GETSEG UUO

IT WILL NEVER BE CONSIDERED "MEDDLING" TO TYPE <CONTROL>C FOLLOWED BY START(WITH NO ARG), CONT, CCONT, CSTART (WITH NO ARG), REENTER, DDT, SAVE, E

AS SOON AS A SHARABLE PROGRAM IS MEDDLED WITH,THE MONITOR SETS A BIT (MEDDLE) FOR THIS USER WHICH MAKES THE CLEARING OF UWP WITH SETUWP UUO AND THE REASSIGNMENT OF CORE FOR THE HIGH SEGMENT WITH THE CORE UUO,(EXCEPT TO REMOVE IT COMPLETELY) GIVE AN ERROR RETURN. ALSO AN ATTEMPT TO MODIFY THE HIGH SEGMENT WITH THE DEPOSIT WILL PRINT "OUT OF BOUNDS". UWP IS ALSO SET FOR THIS USER, IN CASE IT WAS OFF WHEN USER MEDDLED. AN EXCEPTION IS MADE AND THESE UUOS AND COMMAND ARE ALLOWED, IN SPITE OF MEDDLING, IF THE USER HAPPENS TO HAVE THE ACCESS PRIVILEGES TO WRITE THE FILE WHICH INITIALIZED THE SHARABLE SEGMENT. THUS A SYSTEMS PROGRAMMER COULD LOG IN UNDER 1,1 AND PATCH THE HIGH SEGMENT OF A SHARABLE CUSP WHILE IT WAS BEING SHARED. MORE USEFULLY THIS EXCEPTION ALLOWS USERS TO WRITE PROGRAMS WHICH ACCESS SHARABLE HIGH DATA SEGMENTS VIA THE GETSEG UUO(WHICH IS MEDDLING) AND THEN TURN OFF UWP USING SETUWP UUO TO THAT STORES WILL NOT TRAP. IN THE CASE OF DECTAPE, WRITE PRIVILEGES EXIST IF DECTAPE IS ASSIGNED TO THIS JOB (CANNOT BE SYSTEM TAPE) OR NOT ASSIGNED TO ANY JOB (AND IS NOT SYSTEM TAPE). OTHER NON-MEDDLING USERS CAN CONTINUE TO RUN THIS SHARED PROGRAM WITH PEACE OF MIND, KNOWING THAT ONLY AUTHORIZED USER'S OR THE PROGRAM ITSELF IS ALLOWED TO MODIFY THE SHARED SEGMENT THAT THEY ARE USING.

IN MULTICS THIS TECHNIQUE IS CALLED PROTECTED ENTRY POINTS INTO A SHARED PROGRAM. IF CONTROL CAN ONLY BE TRANSFERRED TO A SMALL NUMBER OF ENTRY POINTS WHICH THE SHARED PROGRAM IS PREPARED TO HANDLE, THEN IT CAN DO ANYTHING IT HAS THE PRIVILEGES TO DO, EVEN THOUGH THE PERSON RUNNING THE PROGRAM DOES NOT HAVE THESE PRIVILEGES.

THE ASSIGN (AND DEASSIGN, FINISH, KJOB IF DEVICE PREVIOUSLY ASSIGNED BY CONSOLE) MONITOR COMMAND CLEARS ALL SHARED SEGMENT NAMES CURRENTLY IN USE WHICH WERE INITIALIZED FROM THAT DEVICE, IF THE DEVICE IS REMOVABLE (DTA,MTA). OTHERWISE NEW USERS COULD CONTINUE TO SHARE THE OLD SEGMENT INDEFINITELY, EVEN IF A NEWER VERSION WERE MOUNTED ON THE DEVICE. OBVIOUSLY USERS WHO ARE IN THE MIDDLE OF SHARING A SHARABLE SEGMENT WILL CONTINUE TO DO SO UNTIL THEY ARE FINISHED, EVEN THOUGH THE SEGMENT NAME HAS BEEN CLEARED. THEREFORE, IT IS POSSIBLE TO UPDATE THE LIBRARY DURING REGULAR TIME SHARING, IF ONE

HAS THE COURAGE AND THE ACCESS PRIVILEGES. IN A DECTAPE SYSTEM A NEW CUSP TAPE CAN BE MOUNTED FOLLOWED BY AN ASSIGN SYS WHICH WILL CLEAR SEGMENT NAMES FOR THE PHYSICAL DEVICE (USUALLY DTAØ), BUT NOT ASSIGN THE DEVICE SINCE EVERYONE NEEDS TO SHARE IT.

8. SET USER-MODE WRITE PROTECT UUO

IF A USER PROGRAM WISHES TO STORE INTO A HIGH SEGMENT, IT WILL HAVE TO USE THE SET USER MODE WRITE PROTECT UUO (SETUWP). THE SETUWP UUO ALLOWS THE USER PROGRAM TO SET OR CLEAR THE HARDWARE USER-MODE WRITE PROTECT BIT (UWP) FOR THAT JOB AND TO OBTAIN THE PREVIOUS SETTING. IF THE SYSTEM (MONITOR AND HARDWARE) HAS A TWO REGISTER CAPABILITY, THIS UUO WILL ALWAYS GIVE THE OK RETURN UNLESS THE USER HAS BEEN MEDDLING WITHOUT WRITE PRIVILEGES. WHETHER THE PROGRAM HAS A HIGH SEGMENT OR NOT. THIS FOLLOWS DESIGN GOAL 14 WHICH ALLOWS USERS TO WRITE PROGRAMS FOR TWO REGISTER MACHINES WHICH WILL STILL RUN UNDER ONE REGISTER MACHINES, THEREBY MAINTAINING COMPATIBILITY OF SOURCE AND RELOCATABLE BINARY (BUT NOT SAVED FILES) BETWEEN PDP-10'S AND PDP-6'S (OR PDP-10'S WITHOUT SECOND RELOCATION REGISTER). IF THE SYSTEM HAS ONLY A ONE REGISTER CAPABILITY, THIS UUO WILL ALWAYS GIVE THE ERROR RETURN (BIT 35 OF AC=0 ON RETURN). THIS ALLOWS THE RARE USER PROGRAM TO FIND OUT WHETHER THE SYSTEM HAS A TWO SEGMENT CAPABILITY OR NOT (PROBABLY ONLY THE LOADER WILL WANT TO KNOW).

THE USER PROGRAM SPECIFIES THE DESIRED SETTING OF UWP IN BIT 35 OF AC (1 MEANS WRITE PROTECT, 0 MEANS WRITES OK). THE PREVIOUS SETTING OF UWP IS RETURNED IN BIT 35 OF AC, SO THAT ANY USER SUBROUTINE CAN PRESERVE THE PREVIOUS SETTING BEFORE CHANGING IT. THUS NESTED USER SUBROUTINES CAN BE WRITTEN WHICH EACH SET OR CLEAR UWP AS DESIRED, PROVIDED THEY SAVE THE PREVIOUS VALUE RETURNED BY SETUWP UUO AND RESTORE IT WHEN THEY RETURN TO THEIR CALLERS. THE ERROR RETURN WILL BE GIVEN IN A TWO RELOCATION REGISTER SYSTEM, IF THE USER HAS MEDDLED WITH THE PROGRAM AND DOES NOT HAVE WRITE ACCESS PRIVILEGES. SEE SECTION ON MODIFYING SHARABLE SEGMENT DURING EXECUTION. THIS UUO MAY BE EXECUTED FROM THE LOW OR HIGH SEGMENT, AS A CONVENIENCE, THE CALL [SIXBIT/RESET/] UUO WILL ALWAYS SET THE USER MODE WRITE PROTECT ON IF A HIGH SEGMENT EXISTS, WHETHER IT IS SHARABLE OR NOT, SO THAT PROGRAMS ALWAYS BEGIN PROTECTED.

```
CALL AC, [SIXBIT/SETUWP/]        CALLI AC,36
ERROR RETURN
OK RETURN
```

8.1 RESET UUO

THE CALL [SIXBIT /RESET/] OR CALLI 0 UUO(WHICH EVERY USER PROGRAM SHOULD BEGIN WITH) AUTOMATICALLY TURNS ON THE USER-MODE WRITE PROTECT BIT (UWP) SO THAT A PROGRAM CANNOT INADVERTENTY STORE INTO THE HIGH SEGMENT. SEE SET USER-MODE WRITE PROTECT UUO ABOVE.

## 9. THE ALLOCATION OF VIRTUAL CORE

THE 4 SERIES MONITOR CAN MAKE USE OF ALL OF THE SWAPPING SPACE BY FRAGMENTING SEGMENTS WHEN SPACE RUNS OUT. THUS THE MONITOR KEEPS TRACK OF THE TOTAL VIRTUAL CORE ASSIGNED TO ALL JOBS. SHAREABLE SEGMENTS COUNT ONLY ONCE AND DORMANT (SEE BELOW) DO NOT COUNT AT ALL. THE MONITOR WILL NOT ALLOW MORE VIRTUAL CORE TO BE GRANTED BY THE CORE UUO OR CORE COMMAND THAN THE SYSTEM HAS CAPACITY TO HANDLE. WHEN THE MONITOR IS STARTED THE UNUSED VIRTUAL CORE IS SET EQUAL TO THE AMOUNT OF SWAPPING SPACE PRE-ALLOCATED ON THE DISK, THUS THERE IS ALWAYS ROOM TO SWAP OUT THE LARGEST POSSIBLE JOB IN CORE AND SWAP IN ANOTHER JOB. THE CORE COMMAND WITH NO ARGUMENTS WILL PRINT THE VIRTUAL CORE LEFT IN THE SYSTEM.

### 9.1 THE CORE UUO

IN ORDER TO ALLOCATE CORE IN EITHER OR BOTH SEGMENTS THE LEFT HALF OF THE AC SPECIFIED IN THE CORE UUO WILL BE USED TO SPECIFY THE HIGHEST USER ADDRESS TO BE ASSIGNED IN THE HIGH SEGMENT. A LEFT HALF OF ZERO WILL MEAN THAT THE HIGH SEGMENT CORE ASSIGNMENT IS NOT TO BE CHANGED. A NON-ZERO LEFT HALF LESS THAN 400000, OR THE LENGTH OF THE LOW SEGMENT WHICHEVER IS GREATER, WILL ELIMINATE THE HIGH SEGMENT AND WILL ALWAYS BE LEGAL. OBVIOUSLY IF IT IS EXECUTED FROM THE HIGH SEGMENT, THERE WILL BE AN ILLEGAL MEMORY ERROR MESSAGE PRINTED WHEN THE MONITOR ATTEMPTS TO RETURN CONTROL TO THE ILLEGAL MEMORY. THE ERROR RETURN WILL BE GIVEN IF LH IS GREATER OR EQUAL TO 400000 AND EITHER SYSTEM DOES NOT HAVE TWO SEGMENT CAPABILITY OR THE USER HAS BEEN MEDDLING WITHOUT WRITE ACCESS PRIVILEGES - SEE SECTION ON MODIFYING SHARABLE SEGMENTS DURING EXECUTION. THUS EXISTING PROGRAMS WILL CONTINUE TO WORK SINCE THEY HAVE 0 IN THE LEFT HALF OF THE AC ON CORE UUO'S. A RH OF 0 WILL CONTINUE TO LEAVE THE LOW SEGMENT CORE ASSIGNMENT UNAFFECTED. THE MONITOR WILL CLEAR NEWLY ASSIGNED CORE, SO THAT PRIVACY OF INFORMATION WILL BE INSURED.

THE CORE UUO IS BEING CHANGED IN SWAPPING SYSTEMS SO THAT IT WILL RETURN THE MAXIMUM NUMBER OF 1K CORE BLOCKS AVAILABLE TO A USER. THIS WILL BE ALL OF CORE MINUS THE MONITOR, UNLESS AN INSTALLATION CHOOSES TO RESTRICT THE AMOUNT USING MONGEN DIALOG AND/OR ONCE ONLY DIALOG. NON-SWAPPING SYSTEMS WILL STILL RETURN THE NUMBER OF FREE + DORMANT 1K BLOCKS. THIS MAKES THE UUO AND THE CONSOLE COMMAND RETURN THE SAME INFORMATION. RESTRICTING THE MAXIMUM AVAILABLE USER CORE IMPROVES SYSTEM EFFICIENCY BY INCREASING NUMBER OF JOBS IN CORE SIMULTANEOUSLY.

```
MOVE    AC,[XWD HIGH ADR OR 0,LOW ADR OR 0]
CALL    AC,[SIXBIT /CORE/] OR CALLI AC,11
ERROR   RETURN
NORMAL  RETURN
C(AC)=MAX. NO. OF 1K BLOCKS POSSIBLE FOR THIS USER
```

THE CORE UUO WORKS IN TWO DISTINCT STEPS, OF WHICH THE PROGRAMMER MUST BE AWARE. FIRST, THE LOW SEGMENT IS REASSIGNED (IF RH NON-ZERO), AND THEN THE HIGH SEGMENT IS REASSIGNED(IF LH NON-ZERO). DURING THE FIRST STEP IF THE SUM OF THE NEW LOW SEGMENT AND THE OLD HIGH SEGMENT EXCEEDS THE MAXIMUM SIZE OF CORE ALLOWED TO A USER, THE ERROR RETURN IS GIVEN, THE CORE ASSIGNMENT IS UNCHANGED, AND THE MAXIMUM TOTAL OF CORE AVAILABLE TO THIS USER FOR LOW AND HIGH SEGMENTS IN 1K BLOCKS IS RETURNED IN THE AC. (FOR NON-SWAPPING SYSTEMS, THE NUMBER OF FREE AND DORMANT 1K BLOCKS IS RETURNED). DURING THE SECOND STEP IF THE SUM OF THE NEW LOW SEGMENT AND THE NEW HIGH SEGMENT EXCEEDS THE MAXIMUM SIZE OF CORE ALLOWED TO A USER, THE ERROR RETURN IS GIVEN,THE NEW LOW SEGMENT REMAINS ASSIGNED, BUT THE OLD HIGH SEGMENT IS UNCHANGED, AND THE MAXIMUM POSSIBLE SIZE OF CORE FOR THIS USER IN 1K BLOCKS IS RETURNED IN THE AC. THUS A PROGRAM WHICH IS INCREASING THE LOW SEGMENT AND DECREASING THE HIGH SEGMENT AT THE SAME TIME, SHOULD DO IT WITH TWO SEPARATE CORE UUO'S RATHER THAN JUST ONE.

IF THE NEW LOW SEGMENT IS SO LONG AS TO EXTEND BEYOND 377777, THE HIGH SEGMENT WILL BE SHIFTED UP IN THE VIRTUAL ADDRESSING SPACE RATHER THAN BE OVERLAYED. SUBSEQUENTLY, IF SUCH A LONG LOW SEGMENT IS SHORTENED TO 377777 OR LESS, THE HIGH SEGMENT WILL BE SHIFTED DOWN IN THE VIRTUAL ADDRESSING SPACE TO 400000 RATHER THN GROW LONGER IN LENGTH OR REMAIN WHERE IT WAS. OBVIOUSLY IF THE HIGH SEGMENT IS A PROGRAM, IT WILL NOT EXECUTE PROPERLY AFTER BEING SHIFTED, UNLESS IT GOES TO GREAT PAINS TO BE A SELF-RELOCATING PROGRAM IN WHICH ALL TRANSFER INSTRUCTIONS ARE INDEXED.

IF THE HIGH SEGMENT IS ELIMINATED BY A CORE UUO, A SUBSEQUENT CORE UUO WITH THE LEFT HALF GREATER THAN 400000 WILL CREATE A NEW, NON-SHARABLE SEGMENT. SUCH A SEGMENT CAN ONLY BECOME SHARED AFTER IT HAS BEEN NAMED BY THE APPROPRIATE ENTER UUO TO HAVE AN EXTENSION ,SHR, WRITTEN ONTO THE STORAGE DEVICE WITH OUTPUT UUOS, CLOSED SO THAT DIRECTORY ENTRY IS MADE, AND INITIALIZED FROM THE STORAGE DEVICE WITH A GET, R, OR RUN COMMAND OR RUN OR GETSEG UUO. THIS IS PRECISELY

THE SEQUENCE OF EVENTS WHICH THE LOADER, SAVE AND GET USE TO CREATE AND INITIALIZE NEW SHARABLE SEGMENTS.

## 9.2. THE CORE COMMAND

THE CORE CONSOLE COMMAND WILL NOT BE MODIFIED. IT WILL CONTINUE TO OPERATE ON THE LOW SEGMENT ONLY. CORE ALLOCATION SHOULD BE DONE BY PROGRAMS RATHER THAN PEOPLE. HOWEVER, A CORE 0 WILL CAUSE BOTH THE LOW AND THE HIGH SEGMENT TO DISAPPEAR FROM THE JOBS VIRTUAL ADDRESSING SPACE. THE CORE COMMAND IS NOT CONSIDERED TO BE MEDDLING. AS IN THE CORE UUO, THE MONITOR WILL CLEAR NEW CORE BEFORE ASSIGNING IT TO THE USER.

THE CORE CONSOLE COMMAND WITH NO ARGUMENTS WILL TYPE BACK MORE INFORMATION. FOR MONITORS WITH TWO RELOCATION REGISTER HARDWARE AND SOFTWARE IT WILL RESPOND WITH:

```
L+H/M CORE
VIR. CORE LEFT=N
WHERE   L=# 1K BLOCKS IN LOW SEGMENT
H=# 1K BLOCKS IN HIGH SEGMENT
M=MAXIMUM CORE AVAILABLE TO A USER.
  (SWAP SYSTEMS=MAX PHYSICAL USER CORE UNLESS
  INSTALLATION RESTRICTS)
  (NON-SWAP SYSTEMS=FREE + DORMANT CORE)
N=AMOUNT OF UNASSIGNED VIRTUAL CORE LEFT IN SYSTEM.
```

NON-SWAPPING SYSTEMS WILL RESPOND WITH JUST:

```
L/M CORE
```

WHERE M=FREE+DORMANT CORE
IN ALL CASES THE NUMBER AFTER THE SLASH IS THE SAME AS THE NUMBER RETURNED IN THE USER'S AC FOR THE CORE UUO.

10. GET,R, RUN COMMANDS

GET WILL ALWAYS ASSIGN THE PROPER AMOUNT OF CORE (ONE OR TWO SEGMENTS) NO MATTER WHAT THE PREVIOUS CORE ASSIGNMENT WAS. IT USED TO DO THIS ONLY IF THE USER HAD NO CORE. BECAUSE IT WILL FIRST GIVE BACK PREVIOUS CORE TO MINIMIZE SWAP TIME, GET FROM MAGTAPE MUST ALWAYS HAVE A THIRD ARGUMENT SAYING HOW MUCH CORE FOR LOW SEGMENT (SINCE THERE IS NO DIRECTORY TO TELL LENGTH).

11. THE REMAP UUO

THE REMAP UUO WILL TAKE THE TOP PART OF A LOW SEGMENT AND MAKE IT BE THE NEW HIGH SEGMENT. THE PREVIOUS HIGH SEGMENT(IF ANY) WILL BE REMOVED FROM THE USER'S ADDRESSING SPACE. THE LOW SEGMENT WILL BE AUTOMATICALLY SHORTENED BY THE AMOUNT REMAPPED. THE AMOUNT REMAPPED MUST BE IN MULTIPLES OF 1K DECIMAL WORDS. TO INSURE THIS THE MONITOR WILL OR IN 1777 INTO THE USERS REQUEST. IF THE ARGUMENT EXCEEDS LOW SEGMENT, NO REMAPPING WILL OCCUR, THE OLD HIGH SEGMENT WILL REMAIN IN ADDRESSING SPACE, AND THE ERROR RETURN WILL BE TAKEN. THE ERROR RETURN WILL ALSO BE GIVEN IF THE SYSTEM DOES NOT HAVE TWO REGISTER CAPABILITY.

```
MOVEI AC,  DESIRED HIGHEST ADR IN LOW SEG
CALL AC,  [SIXBIT   /REMAP/]   CALLI  AC,37

ERROR RETURN
OK RETURN
```

WHERE AC CONTAINS THE NEW HIGHEST LEGAL ADDRESS IN LOW SEGMENT, AFTER THE PART ABOVE IT HAS BEEN REMAPPED INTO THE HIGH SEGMENT. THE CONTENTS OF JOBREL WILL BE SET TO THE NEW HIGHEST LEGAL RELATIVE (USER) ADDRESS IN LOW SEGMENT. THE CONTENTS OF RH OF JOBHRL IN THE JOB DATA AREA WILL BE SET TO THE NEW HIGHEST LEGAL RELATIVE (USER) ADDRESS (401777 OR GREATER OR 0). THE HARDWARE RELOCATION WILL BE CHANGED AND UWP WILL BE SET ON.

THIS UUO HAS BEEN INCLUDED PRIMARILY FOR THE LOADER SO THAT IT CAN LOAD RE-ENTRANT PROGRAMS WHICH USE UP ALL OF PHYSICAL CORE. THE LOADER MIGHT EXCEED CORE IF IT HAD TO ASSIGN MORE CORE AND MOVE THE DATA FROM THE LOW TO THE HIGH SEGMENT WITH A BLT INSTRUCTION. GET WILL ALSO USE REMAP, SO THAT IT CAN DO I/O INTO LOW SEGMENT RATHER THAN HIGH SEGMENT.

12. A RUN UUO, ANALOGOUS TO THE RUN CONSOLE COMMAND, HAS BEEN IMPLEMENTED SO THAT PROGRAMS CAN TRANSFER CONTROL TO ONE ANOTHER. THIS UUO REPLACES BOTH THE LOW AND HIGH SEGMENTS OF THE USER'S ADDRESSING SPACE WITH THE PROGRAM BEING CALLED.

```
        MOVSI   AC,STARTING ADDRESS INCREMENT
        HRRI    AC,ADR OF 6 WORD ARG BLOCK
        CALL    AC,[SIXBIT /RUN/] OR CALLI  AC,35
ERROR RETURN(UNLESS HALT IN LH)
[NORMAL RETURN IS NOT HERE, BUT TO STARTING
ADDRESS + INCREMENT OF NEW PROGRAM]
```

THE SIX ARGUMENTS ARE, THE LOGICAL DEVICE NAME IN SIXBIT, A FILE NAME (EXTENSION AND PROJECT PROGRAMMER NUMBER ARE OPTIONAL) AND AN OPTIONAL CORE ASSIGNMENT.

```
E/      SIXBIT LOGICAL DEVICE NAME
E+1/    SIXBIT FILE NAME (FOR EITHER OR BOTH HIGH
        AND LOW FILES)
E+2/    LH=SIXBIT EXTENSION FOR LOW FILE (IF Ø,
        .LOW ASSUMED IF HIGH SEG EXISTS AND .SAV
        ASSUMED IF NO HIGH SEG)
E+3/
E+4/    PROJECT-PROGRAMMER NUMBER (IF Ø, USE
        CURRENT USER'S)
E+5/    RH =NEW HIGHEST USER ADDRESS TO BE ASSIGNED
        TO LOW SEGMENT LH IS IGNORED RATHER THAN
        SETTING HIGH SEGMENT.)
```

USUALLY A USER PROGRAM WILL SPECIFY ONLY E AND E+1 AND WILL SET E+2, E+4, AND E+5 TO Ø. THESE OTHER ARGUMENTS HAVE BEEN INCLUDED FOR COMPLETENESS. NOTE THAT E+1 THROUGH E+4 ARE SAME AS LOOKUP BLOCK.

UNFORTUNATELY, THE AC'S ARE DESTROYED BY THE RUN UUO SO THAT ARGUMENTS CANNOT BE PASSED TO THE NEXT PROGRAM. ALSO ALL THE USER I/O CHANNELS ARE RELEASED, SO THAT DEVICES CANNOT BE PASSED EITHER.

NOTE THAT PROGRAMS ON THE SYSTEM LIBRARY (CUSPS) SHOULD BE CALLED BY USING DEVICE SYS WITH THE PROJECT-PROGRAMMER NUMBER (E+4) OF Ø RATHER THAN DEVICE DSK AND PROJECT-PROGRAMMER NUMBER 1,1. THIS IS FOR 2 REASONS:

1. THE LIBRARY MAY BE ON DECTAPE.

2. WE WANT TO MOVE THE CUSPS FROM 1,1 TO 1,3 (OR SOME OTHER ONE) SO THAT 1,1 IS ONLY THE MFD.

THE EXTENSION (E+2) SHOULD ALSO BE Ø SO THAT THE USER PROGRAM DOES NOT NEED TO KNOW WHETHER THE CUSP IS REENTRANT OR NOT
(EXTENSION .LOW VERSUS .SAV). THE LEFT HALF OF AC WILL

BE ADDED TO AND STORED IN THE STARTING ADDRESS OF THE NEW PROGRAM (I.E. ADDED TO C(JOBSA)) BEFORE TRANSFERRING CONTROL TO THE NEW PROGRAM. THUS <CONTROL> C START WILL RESTART PROGRAM AT THE SAME LOCATION AS SPECIFIED BY THE RUN UUO, IN CASE THE USER WISHES TO START THE CURRENT CUSP OVER AGAIN. THE USER WILL BE CONSIDERED TO BE MEDDLING WITH THE PROGRAM IF THE LH OF AC IS NOT Ø OR 1. SEE SECTION ON MODIFYING SHARABLE SEGMENTS DURING EXECUTION.

AS A SYSTEM WIDE CONVENTION, PROGRAMS WHICH ACCEPT COMMANDS FROM A TELETYPE OR A FILE DEPENDING ON HOW THEY ARE STARTED, WILL DO SO AS CONTROLLED BY THE CALLING PROGRAM (I.E. PROGRAM DOING THE RUN UUO). Ø IN LH OF AC MEANS TYPE * AND ACCEPT COMMANDS FROM TTY, AND 1 MEANS ACCEPT THEM FROM A COMMAND FILE, IF IT EXISTS, OTHERWISE TYPE * AND ACCEPT COMMANDS FROM TTY. AS A CONVENTION SUCH PROGRAMS SHOULD LOOKUP THEIR COMMAND FILE WITH NAME OF FORM ###III.TMP WHERE III IS THE FIRST 3 (OR FEWER IF 3 DO NOT EXIT) OF THE CUSP DOING THE LOOKUP. "###" IS THE LAST 3 CHARACTERS OF THE DECIMAL CHARACTER EXPANSION (WITH LEADING Ø'S IF NECESSARY TO MAKE 3 CHARACTERS) OF THE BINARY JOB NUMBER. THE PURPOSE OF INCLUDING THE JOB NUMBER IS TO ALLOW A USER TO RUN 2 OR MORE JOBS UNDER THE SAME PROJECT-PROGRAMMER NUMBER.
EXAMPLE: ØØ9PIP.TMP,Ø39MAC.TMP. DECIMAL SHOULD BE USED RATHER THAN OCTAL, SO THAT A USER LISTING HIS DIRECTORY WILL SEE THE SAME NUMBER AS THE PJOB COMMAND TYPES. SUCH COMMAND FILES ARE TEMPORARY AND ARE DELETED BY LOGOUT SINCE THE EXTENSION IS .TMP.

;ROUTINE TO CREATE DECIMAL JOB NUMBER EXTENSION FROM JOB NUMBER

```
      CALLI AC,30              ;GET JOB# FROM MONITOR
      MOVEI T,3                ;INIT DIGIT COUNT
LUP:  IDIVI AC,12              ;GET RIGHT DIGIT INTO AC+1
      ADDI  AC+1,"Ø"-40        ;CONVERT TO SIXBIT
      LSHC  AC+1,-6            ;SAVE DIGIT IN AC+2
      SOJG  T,LUP              ;FORCE 3 DIGITS
      HLLM  AC+2,E             ;SAVE AS NAME (LH)
```

IN ORDER TO FACILITATE THE IMPLEMENTATION OF THE CONCISE COMMAND LANGUAGE, THE RUN UUO WILL GIVE AN ERROR RETURN WITH ONE OF 13 ERROR CODES IN AC IF ANY ERRORS ARE DETECTED, RATHER THAN STOPPING THE JOB AND PRINTING A MONITOR ERROR MESSAGE. IN THIS WAY THE USER PROGRAMS CAN ATTEMPT TO RECOVER FROM THE ERROR OR GIVE THE USER A MORE INFORMATIVE MESSAGE ON HOW TO PROCEED. BECAUSE SOME USER PROGRAMS WILL NOT WANT TO GO TO THE BOTHER OF ERROR RECOVERY (SAY DURING CHECKOUT), THE MONITOR WILL NOT GIVE AN ERROR RETURN IF THE LH OF THE ERROR RETURN LOCATION IS A HALT INSTRUCTION. THIS ALSO ALLOWS THE CONSCIENTIOUS USER PROGRAM TO EXECUTE A SECOND RUN UUO WITH A HALT IF THE ERROR CODE IS FOR AN ERROR FOR WHICH THE MONITOR MESSAGE IS SUFFICIENTLY INFORMATIVE AND FOR WHICH THE USER PROGRAM CANNOT RECOVER.

THE ERROR CODES ARE AN EXTENSION OF THE LOOKUP ENTER, AND RENAME UUO ERROR CODES AND ARE DEFINED ON THE S.MAC MONITOR FILE.

| | | |
|---|---|---|
| FNFERR | Ø | FILE NOT FOUND |
| IPPERR | 1 | INCORRECT PROJ-PROG NO. (NON-EXISTENT) |
| PRTERR | 3 | FILE BEING MODIFIED |
| AEFERR | 4* | ALREADY EXISTING FILE |
| NLEERR | 5* | NEITHER LOOKUP NOR ENTER |
| TRNERR | 6 | TRANSMISSION ERROR |
| NSFERR | 7 | NOT A SAVE FILE |
| NECERR | 1Ø | NOT ENOUGH CORE |
| DNAERR | 11 | DEVICE NOT AVAILABLE |
| NSDERR | 12 | NO SUCH DEVICE |
| ILUERR | 13* | ILLEGAL UUO (GETSEG UUO ONLY ON 1 REG SYSTEM) |

*NOT POSSIBLE ON RUN UUO.

THE MONITOR IS CAREFUL NOT TO ATTEMPT TO ERROR RETURN TO A USER PROGRAM AFTER THE HIGH OR LOW SEGMENT CONTAINING THE RUN UUO HAS BEEN OVERLAYED.

IN ORDER TO SUCCESSFULLY PROGRAM THE RUN UUO FOR ALL SIZE SYSTEMS AND ALL CUSPS WHOSE SIZE IS NOT KNOWN AT THE TIME RUN UUO IS CODED, IT IS NECESSARY TO UNDERSTAND THE SEQUENCE OF OPERATION OF THE RUN UUO. (IT IS THE SAME AS FOR GET, R, AND RUN COMMANDS, EXCEPT THAT THE COMMANDS REMOVE THE OLD HIGH SEGMENT FROM THE LOGICAL ADDRESSING SPACE AND REDUCE THE LOW SEGMENT TO 1K BEFORE PROCEEDING WITH THE FOLLOWING.)

ASSUME THAT THE JOB EXECUTING THE RUN UUO HAS BOTH A LOW AND A HIGH SEGMENT. (THE RUN UUO CAN BE EXECUTED FROM EITHER SEGMENT, HOWEVER FEWER ERRORS CAN BE RETURNED TO USER IF RUN UUO IS EXECUTED FROM HIGH SEGMENT.

STEPS 1-44 FOR RUN UUO, 1-31 FOR GETSEG UUO

1. DOES A HIGH SEG ALREADY EXIST BY SAME NAME? (IF YES, GO TO 3Ø)

2. INIT AND LOOKUP FILE NAME .SHR(IF NOT FOUND, GO TO 1Ø).

3. READ HIGH FILE INTO TOP OF LOW SEGMENT BY EXTENDING IT. (HERE THE OLD LOW SEG AND NEW HIGH SEG AND OLD HIGH SEG MAY NOT EXCEED THE CAPACITY OF CORE.)

4. REMAP THE TOP OF LOW SEGMENT REPLACING OLD HIGH SEG IN LOGICAL ADDRESSING SPACE.

5. STORE NAME OF THIS NEW SHARABLE HIGH SEG SO OTHERS CAN SHARE (GO TO 4Ø OR RETURN TO USER IF GETSEG UUO).

1Ø. LOOKUP FILE NAME .HGH. (IF NOT FOUND, GO TO 35 OR ERROR RETURN TO USER IF GETSEG UUO)

11. READ HIGH FILE INTO TOP OF LOW SEGMENT BY EXTENDING IT. (HERE AGAIN THE OLD LOW SEG AND NEW HIGH SEG AND OLD HIGH SEG MAY NOT EXCEED THE CAPACITY OF CORE)

12. CHECK FOR IO ERRORS. (IF YES, ERROR RETURN TO USER UNLESS HALT IN LH OF RETURN) (GO TO 41)

30. REMOVE OLD HIGH SEG FROM LOGICAL ADDRESSING SPACE, IF ANY.

31. PLACE THE SHARABLE SEGMENT IN USER'S LOGICAL ADDRESSING SPACE (GO TO 40 OR RETURN TO USER IF GETSEG UUO).

35. REMOVE OLD HIGH SEG FROM LOGICAL ADDRESSING SPACE, IF ANY. (GO TO 41)

40. COPY VESTIGIAL JOB DATA AREA INTO JOB DATA AREA. DOES THE NEW HIGH SEG HAVE A LOW FILE (LH JOBCOR>137). (IF NO, GO TO 45)

41. LOOKUP FILE NAME.SAV OR .LOW OR USER SPECIFIED (ERROR IF NOT FOUND. RETURN TO USER IF CALL FROM LOW SEG AND NOT HALT IN LH OF ERROR RETURN).

42. REASSIGN LOW SEG CORE ACCORDING TO SIZE OF FILE OR USER SPECIFIED CORE ARGUMENT WHICHEVER IS LARGER. PREVIOUS LOW SEGMENT IS OVERLAYED

43. READ IN LOW FILE. INTO BEGINNING OF LOW SEGMENT

44. CHECK FOR IO ERRORS (YES, PRINT ERROR MESSAGE, DO NOT RETURN TO USER)

45. REASSIGN LOW SEGMENT CORE ACCORDING TO LARGER OF USER'S CORE ARGUMENT OR ARGUMENT WHEN FILE SAVED (RH JOBCOR)

IN ORDER TO ALWAYS BE GUARANTEED OF HANDLING THE MOST NUMBER OF ERRORS, THE CAUTIOUS USER SHOULD REMOVE HIS HIGH SEGMENT FROM HIGH LOGICAL ADDRESSING SPACE (USE CORE

UUO WITH A ONE IN LH OF AC). THE ERROR HANDLING CODE SHOULD BE PUT IN THE LOW SEGMENT ALONG WITH THE RUN UUO AND THE SIZE OF OF THE LOW SEGMENT REDUCED TO 1K. AN EVEN BETTER IDEA WOULD BE TO HAVE THE ERROR HANDLING CODE BE WRITTEN ONCE AND PUT IN A SELDOM USED (PROBABLY NON-SHARABLE) HIGH SEGMENT WHICH COULD BE GOTTEN IN HIGH SEGMENT USING GETSEG UUO (SEE BELOW) WHEN AN ERROR RETURN OCCURS TO LOW SEGMENT ON A RUN UUO.

13. A GETSEG UUO(CALLI 40) HAS BEEN IMPLEMENTED SO THAT JUST A HIGH SEGMENT CAN BE INITIALIZED FROM A FILE OR SHARED WITHOUT AFFECTING THE LOW SEGMENT. THIS UUO IS BEING IMPLEMENTED FOR SHARED DATA SEGMENTS AND SHARED PROGRAM OVERLAYS. IT IS ALSO BEING USED FOR RUN TIME ROUTINES SUCH AS FORTRAN OR COBOL OPERATING SYSTEMS (BUT NOT THE SELECTIVE LIBRARY ROUTINES) THIS UUO WORKS EXACTLY LIKE THE RUN UUO EXCEPT THAT

1) NO ATTEMPT IS MADE TO READ A LOW FILE.

2) NO CHANGE IS MADE TO LOW SEGMENT OR JOB DATA AREA EXCEPT BOTH HALVES OF JOBHRL. (ONLY JOBDAT LOCATIONS DESCRIBING HIGH SEGMENT)

3) IF AN ERROR OCCURS CONTROL IS RETURNED TO THE ERROR RETURN, UNLESS LH=HALT.

4) IF EVERYTHING OK, CONTROL IS RETURNED 2 LOCATIONS FOLLOWING UUO WHETHER IT IS CALLED FROM LOW OR HIGH SEGMENT. (SO GENERALLY IT SHOULD BE CALLED FROM LOW SEGMENT UNLESS THE NORMAL RETURN HAPPENS TO COINCIDE WITH THE STARTING ADDRESS OF THE NEW HIGH SEGMENT.)

5) SEE STEPS 1 THROUGH 31 IN RUN UUO DESCRIPTION.

6) USER CHANNELS 1 THRU 17 ARE NOT RELEASED. THIS IS SO GETSEG UUO CAN BE USED FOR PROGRAM OVERLAYS SUCH AS COBOL COMPILER. CHANNEL Ø IS RESET AND USED BY THE GETSEG UUO.

14. FOR EFFICIENT EXAMINING OF THE MONITOR, A SPY UUO HAS BEEN IMPLEMENTED WHICH PLACES ANY NUMBER OF K OF PHYSICAL CORE IN THE USER'S HIGH ADDRESSING SPACE. THE SO-CALLED SPY SEGMENT CANNOT BE SAVED (NO ERROR IF TRIED), CANNOT BE INCREASED OR DECREASED BY CORE UUO, (ERROR RETURN) AND CANNOT HAVE UWP TURNED OFF (ERROR RETURN).

```
MOVEI   AC, HIGHEST PHYSICAL CORE LOCATION DESIRED
CALL    AC, [SIXBIT /SPY/] OR CALLI     AC,42
ERROR RETURN (DO NOT HAVE PRIVILEGES)
OK RETURN
```

ANY PROGRAM WRITTEN TO USE SPY UUO SHOULD TRY PEEK UUO IF IT GETS AN ERROR RETURN FROM SPY UUO. THIS PROGRAM WILL STILL RUN, ALTHOUGH LESS EFFICIENTLY, ON PDP-6'S AND PDP-1Ø'S WITHOUT KT1ØA OPTION.

15. SUPERCEDING SEGMENTS AND RELEASE UUO

OCCASIONALLY IT WILL BE DESIRABLE TO SUPERCEDE A SHARABLE PROGRAM OR DATA SEGMENT WHICH IS IN THE PROCESS OF BEING SHARED BY A NUMBER OF USERS (SEE DESIGN GOAL 11). WHENEVER A SUCCESSFUL CLOSE OUTPUT UUO OR RENAME UUO IS EXECUTED FOR A FILE WITH THE SAME DIRECTORY NAME AND FILE NAME (PREVIOUS NAME IF RENAME UUO) AS A SEGMENT BEING SHARED, THE SEGMENT'S NAME WILL BE SET TO Ø, SO THAT NO NEW USERS CAN SHARE THE OLDER VERSION WHEN THEY DO AN R, RUN, GET COMMAND OR RUN, GETSEG UUO. INSTEAD THEY WILL START SHARING THE NEWER VERSION WHICH WILL REQUIRE THE MONITOR TO READ THE NEWLY CREATED FILE ONCE TO INITIALIZE THE NEWER SEGMENT. THE OLDER SEGMENT WILL BE DELETED WHEN ALL THE USERS ARE FINISHED SHARING IT.

## 16. USING THE LINKING LOADER

ONE OF THE DESIGN GOALS OF WRITING REENTRANT USER SOFTWARE, IS TO MINIMIZE THE EFFORT REQUIRED TO SUPPORT RE-ENTRANT SOFTWARE WHICH MUST ALSO RUN ON A MACHINE HAVING ONLY A SINGLE RELOCATION REGISTER (PDP-6). TO DO THIS, BOTH THE SOURCES AND RELOCATABLE BINARIES CAN BE THE SAME FOR A PROGRAM WHICH WANTS TO BE REENTRANT ON THE PDP-10 BUT WHICH ALSO SHOULD RUN ON THE PDP-6.

THE DECISION THAT A PROGRAM WRITTEN TO BE RE-ENTRANT IS TO BE ONE SEGMENT INSTEAD OF TWO CAN BE POSTPONED TO LINKING LOAD TIME (RATHER THAN EARLIER AT CODING OR ASSEMBLY TIME). THUS, THE LOADER WILL HAVE A SWITCH (H MEANING NO HIGH SEGMENT), WHICH WILL BE USED ONLY WHEN A TWO SEGMENT PROGRAM IS TO BE LOADED INTO ONE SEGMENT INSTEAD OF THE USUAL TWO. OBVIOUSLY, ONE SEGMENT PROGRAMS WILL CONTINUE TO BE LOADED INTO ONE SEGMENT AND THE H SWITCH WILL NOT BE REQUIRED. IT IS HOPED THAT OUR CUSTOMERS WILL FOLLOW THIS PRACTICE ALSO.

TO FURTHER MINIMIZE THE USE OF THE H SWITCH ON SINGLE REGISTER MACHINES, THE LOADER WILL CHECK TO SEE IF THE SYSTEM HAS THE TWO SEGMENT CAPABILITY. IF THE MONITOR HAS A TWO SEGMENT CAPABILITY, BUT, THE MACHINE DOES NOT, THE SYSTEM WILL BEHAVE AS IF IT DOES NOT. IF IT DOES NOT, THE LOADER WILL AUTOMATICALLY LOAD A TWO SEGMENT PROGRAM INTO JUST ONE SEGMENT, JUST AS IF THE USER HAD TYPED THE H SWITCH. THUS, THE ONLY USE OF THE H SWITCH WILL BE TO LOAD A TWO SEGMENT PROGRAM ON A TWO SEGMENT SYSTEM WHICH IS INTENDED TO RUN ON A ONE SEGMENT SYSTEM. TO FIND OUT IF THE SYSTEM HAS A TWO SEGMENT CAPABILITY, THE LOADER WILL USE THE CALL SETUWP UUO AND ATTEMPT TO SET ITS USER MODE PROTECT BIT TO ONE (EVEN IF IT DOESN'T HAVE A HIGH SEGMENT). AN ERROR RETURN WILL INDICATE THAT THE SYSTEM HAS ONLY A SINGLE REGISTER CAPABILITY. NOTE THAT ON A ONE SEGMENT SYSTEM, THE LOADER WILL NOT BE ABLE TO PRODUCE A TWO SEGMENT PROGRAM AND THE MONITOR WILL NOT BE ABLE TO SAVE IT AS TWO SEGMENTS.

SINCE THE DECISION AS TO WHETHER A RE-ENTRANT PROGRAM IS NOT GOING TO BE LOADED INTO A HIGH SEGMENT WILL BE POSTPONED TO LINKING LOAD TIME, THE CODE EXECUTED (INCLUDING MONITOR UUO'S) WILL BE THE SAME FOR EITHER CASE. THE MONITOR UUO'S HAVE BEEN DESIGNED WITH THIS OBJECTIVE IN MIND; ANY INCONSISTENCIES IN THE MONITOR UUO SHOULD BE POINTED OUT SO THAT THEY CAN BE FIXED.

## 17. ASSEMBLER PSEUDO-OP-HISEG

EACH SUBPROGRAM ASSEMBLED BY MACRO MUST BE EITHER LOADED ENTIRELY INTO THE LOW SEGMENT OR ENTIRELY INTO THE HIGH SEGMENT. TO INDICATE THAT A SUBPROGRAM IS TO BE LOADED INTO THE HIGH SEGMENT, USE THE HISEG PSEUDO-OP ANYWHERE IN THE PROGRAM. (AT THE BEGINNING IS BEST SINCE IT TELLS THE READER THAT THIS IS DESTINED FOR THE HIGH SEGMENT.) MACRO GENERATES BLOCK TYPE 3 NEAR THE BEGINNING OF THE BINARY OUTPUT WHICH TELLS THE LOADER TO LOAD THIS SUBPROGRAM INTO THE HIGH SEGMENT.

## 18. MODIFICATIONS TO LINKING LOADER

THE LOADER IS ITSELF REENTRANT, SO THAT ITS INSTRUCTIONS EXIST IN THE HIGH SEGMENT.
THE LOADER HAS BEEN MODIFIED TO LOAD TWO SEGMENTS INSTEAD

OF ONE. HOWEVER, SINCE BOTH SEGMENTS ARE DATA WITH RESPECT TO THE LOADER, THE TWO SEGMENTS MUST BOTH EXIST IN THE LOW SEGMENT DURING LOAD TIME. THUS, THE LOADER MUST DUPLICATE THE FOLLOWING LOADER VARIABLES, AND HAVE ONE FOR EACH SEGMENT.

ORIGIN (LOW=140, HIGH=400010)
OFFSET
LOCATION COUNTER

THE LOADER LOADS AN ENTIRE RELOCATABLE SUBPROGRAM INTO THE HIGH SEGMENT OR LOW SEGMENT DETERMINED BY WHETHER THEY CONTAIN A HISEG PSEUDO-OP OR NOT. FURTHERMORE, ALL SUBPROGRAMS TO BE LOADED INTO THE LOW SEGMENT MUST BE LOADED BEFORE ANY SUBPROGRAMS ARE LOADED INTO THE HIGH SEGMENT. IF THE LOADER ENCOUNTERS A SUBPROGRAM (OF NON-ZERO LENGTH)WITHOUT HISEG PSEUDO-OP AFTER IT HAS SEEN ONE WITH HISEG PSEUDO-OP, IT WILL PRINT THE FOLLOWING ERROR MESSAGE:

LOW SEG PROG XXXXXX PRECEDED BY HISEG PROG

AN REINITIALIZE ITSELF SO THAT NO PROGRAMS HAVE BEEN LOADED. THE EXCEPTION FOR 0 LENGTH FILES TO BE OUT OF ORDER IS SO THAT JOBDAT CAN BE LOADED DURING USUAL LIBRARY SEARCH AFTER SOME HIGH ROUTINES HAVE BEEN LOADED.

SINCE VERY OCCASIONALLY IT WILL BE DESIRABLE TO LOAD A PROGRAM IN WHICH THE LOW SEGMENT IS LONGER THAN 400000 OCTAL WORDS, THE SWITCH NNNNNNH ALLOWS THE USER TO CHANGE THE ORIGIN OF THE HIGH SEGMENT FROM ITS INITIAL SETTING OF 400000 TO NNNNNN, WHERE NNNNNN SHOULD BE LARGER. RECALL THAT IF NNNNNN IS MISSING, THE LOADER WILL LOAD EVERYTHING INTO THE LOW SEGMENT.

AFTER LOADING IS COMPLETE, THE REENTRANT LOADER WILL:

1) SET LH OF JOBHRL IN THE JOB DATA AREA TO THE NEW HIGHEST RELATIVE USER ADDRESS (RELATIVE TO HIGH SEGMENT ORIGIN) IN HIGH SEGMENT, OR 0 IF NO HIGH SEGMENT.

2) SET THE LH OF JOBCOR TO THE HIGHEST LOCATION IN LOW SEGMENT LOADED WITH NON-ZERO DATA.

3) EXCHANGE THE SYMBOL TABLE WITH THE PART DESTINED FOR THE HIGH SEGMENT

4) USE REMAP UUO TO MAKE TOP PART OF LOW SEGMENT WHICH CONTAINS THE INTENDED HIGH SEGMENT REPLACE THE LOADER AS THE HIGH SEGMENT.

5) CALL EXIT OR START UP PROGRAM.

19. USING DDT

DDT CLEARS AND SAVES UWP USING SETUWP BEFORE WRITING INTO A HIGH SEGMENT WHICH CAN BE SHARABLE (PROVIDED THAT THE USER HAS NOT MEDDLED WITH THE PROGRAM UNLESS HE HAS WRITE PRIVILEGES). HOWEVER, THE BEST WAY TO DEBUG A SHARABLE PROGRAM IS TO MAKE A PRIVATE COPY:

```
GET     SYS     CUSP
SAVE    DSK     CUSP
GET     DSK     CUSP
```

IN FACT IT IS A BAD IDEA TO PUT A CUSP IN SYS DIRECTORY WHICH HAS A DDT IN IT, SINCE A MALACIOUS USER COULD MODIFY THE SHARABLE HIGH SEGMENT USING DDT. DDT WILL RESTORE UWP BEFORE EXECUTION. DDT WILL NOT BE RE-ENTRANT INITIALLY AND SO WILL BE LOADED INTO LOW SEGMENT. THE SYMBOL TABLE WILL CONTINUE TO BE AT THE TOP OF THE LOW SEGMENT.

20. JOB DATA AREA(JOBDAT)

THE SYMBOLIC LIBRARY FILE WHICH DEFINES THE USER'S JOB DATA AREA (FIRST 140 LOCATIONS) HAS ONE NEW ENTRY.

LOCATION JOBHRL=115 IS ANALOGOUS TO JOBREL AND ITS RH CONTAINS THE HIGHEST LEGAL USER ADDRESS IN THE HIGH SEGMENT. OBVIOUSLY IT IS GREATER THAN OR EQUAL TO 401777, UNLESS THERE IS NO HIGH SEGMENT, IN WHICH CASE IT WILL BE 0. THE RH OF JOBHRL IS SET BY THE MONITOR (LIKE JOBREL WHOSE LH IS ALWAYS 0) EVERY TIME THE USER STARTS TO RUN, OR DOES A CORE OR REMAP UUO. THE PROPER WAY TO TEST IF A HIGH SEGMENT EXISTS IN THE ADDRESSING SPACE IS TO TEST IF JOBHRL IS NON-ZERO (BOTH HALVES). NOTE THAT THIS IS A MUCH DIFFERENT TEST THAN WHETHER OR NOT THE SYSTEM HAS A TWO REGISTER CAPABILITY. (SEE SETUWP UUO).

THE LH OF JOBHRL=115 IS ANALOGOUS TO JOBFF AND CONTAINS THE FIRST RELATIVE FREE LOCATION IN THE HIGH SEGMENT.

(RELATIVE TO THE HIGH SEGMENT ORIGIN SO IT IS SAME AS HIGH SEGMENT LENGTH), THE LH OF JOBHRL IS SET BY THE LINKING LOADER AND SUBSEQUENT GETS, EVEN IF THERE IS NO FILE TO INITIALIZE THE LOW SEGMENT.

THE REASON THAT THE LH IS A RELATIVE QUANTITY IS THAT THE SAME SHARED SEGMENT CAN APPEAR AT DIFFERENT USER ORIGINS AT THE SAME TIME. THE SAVE COMMAND USES THIS QUANTITY TO KNOW HOW MUCH TO WRITE FROM THE HIGH SEGMENT (ALL IS WRITTEN IF LH=0 (LIKELY IF USER CREATED HIGH SEG USING CORE OR REMAP UUOS).

THE RH OF JOBERR (=42) WILL BE USED TO PASS THE ACCUMULATED ERROR COUNT FROM ONE CUSP TO THE NEXT DURING A CCL SEQUENCE OF CUSPS. THE LH WILL NOT BE USED AND WILL BE SAVED FOR FUTURE EXPANSION, SO CUSPS SHOULD BE WRITTEN TO LOOK ONLY AT THE RH OF JOBERR, EVEN THOUGH THIS MEANS ANOTHER INSTRUCTION.

THE VERSION NUMBER OF A CUSP WILL BE STORED IN THE RH LOCATION JOBVER(=137) SO THAT THE E COMMAND CAN BE USED TO FIND THE VERSION NUMBER AFTER A GET, R, OR RUN. THE LH WILL CONTAIN THE PROGRAMMER NUMBER OF THE PROGRAMMER WHO LAST MADE A CHANGE (THE PERSON WHO INCREASED THE RH). DIGITAL WILL ALWAYS DISTRIBUTE CUSPS WITH THE LH=0, SO IT IS SUGGESTED THAT CUSTOMERS MAKING MODIFICATIONS TO CUSPS, CHANGE ONLY THE LH, SO THAT THE RH REMAINS AS A RECORD OF THE DEC VERSION

THE LOADER USES 3 CONSECUTIVE LOCATIONS (JOBBLT=45) IN THE JOB DATA AREA WHICH THE USER WOULD NEVER WANT TO LOAD INTO. THE LOADER PUTS A BLT INSTRUCTION AND A CALLI UUO TO MOVE THE PROGRAM DOWN ON TOP OF THE LOADER. THESE LOCATIONS ARE DESTROYED ON EVERY EXEC UUO BY THE EXEC PUSH DOWN LIST.
CHAIN PROGRAM (FORTRAN RUNTIME ROUTINE) NEEDS 6 TEMP LOCATIONS IN JOB DATA AREA FOR ITS OVERLAY. SINCE CHAIN ALWAYS RELEASES ALL IO CHANNELS, JOBCN6=106 IS DEFINED TO BE IN JOBJDA TABLE.

```
        JOBERR=42       ;LH UNUSED AT PRESENT, RH CUSP ACCUMULATED
                        ;ERROR COUNT
        JOBHRL=115      ;LH=FIRST FREE LOC (RELATIVE) IN HIGH SEG,
                        ;RH=HIGHEST LEGAL ADR IN HIGH SEG
        JOBCOR=133      ;LH=HIGHEST LOC IN LOW SEG LOADED WITH
                        ;NON-ZERO DATA, SET BY LOADER RH=USER
                        ;ARG ON LAST SAVE OR GET COMMAND, SET
                        ;BY MONITOR
        JOBVER=137      ;LH=PROGRAMMER NO. MAKING
                        ;CHANG, RH=VERSION NUMBER
        JOBBLT=45       ;3 LOCATIONS WHERE THE LOADER CAN PUT
                        ;INSTRUCTIONS TO MOVE PROGRAM
                        ;DOWN ON TOP OF ITSELF. THESE
                        ;LOCS DESTROYED ON EXEC UUOS.
        JOBCN6=106      ;6 LOCATIONS USED BY CHAIN AFTER
                        ;IT RELEASES ALL IO CHANNELS
```

THERE ARE A FEW "CONSTANT" DATUM IN THE JOB DATA AREA WHICH A TWO-SEGMENT, ONE FILE PROGRAM MIGHT LIKE TO LOAD WITHOUT HAVING TO USE INSTRUCTIONS ON A GET:

```
        JOB41           ;USER LOCATION41
        JOBREN          ;RH IS REENTER STARTING ADDRESS
                        ;LH IS UNUSED (SET TO 0) SAVE FOR
                        ;FUTURE EXPANSION
        JOBVER          ;RH IS VERSION NUMBER, LH IS
                        ;PROGRAMMER NO. OF PROGRAMMER
                        ;WHO LAST CHANGED THE PROGRAM,
                        ;OR CUSTOMER SUB-VERSION NUMBER
```

AND THERE ARE A NUMBER OF LOCATIONS WHICH THE MONITOR MUST LOAD ON A GET.

```
JOBSA           ;LH=FIRST FREE LOC IN LOW SEG

                ;RH=STARTING ADDRESS
JOBCOR          ;LH=LAST LOC IN LOW SEG WITH DATA
                 (SET BY LOADER)
                ;RH=SIZE OF CORE TO BE ASSIGNED
                ;ON GET (SAVE'S THIRD ARG IF ANY
                ;OR SIZE OF CORE NEEDED)
JOBHRL          ;LH=FIRST FREE LOC IN HIGH SEG
                ;RELATIVE TO ITS USER ORIGIN, I.E.
                ;LENGTH
```

IN ORDER TO DO THIS, THE FIRST 10 (OCTAL) LOCATIONS OF HIGH SEGMENT WILL BE RESERVED FOR THESE LOW SEGMENT CONSTANTS. THUS, A HIGH PROGRAM WILL BE LOADED BY THE LOADER INTO 400010 INSTEAD OF 400000.

WITH THE VESTIGIAL DATA AREA IN THE HIGH SEGMENT, THE MONITOR WILL AUTOMATICALLY LOAD THE ABOVE CONSTANT DATA INTO THE JOB DATA AREA WITHOUT REQUIRING A LOW FILE ON A GET,R,RUN COMMAND OR RUN UUO(BUT NOT GETSEG UUO). SAVE WILL WRITE A LOW FILE FOR A TWO SEGMENT PROGRAM ONLY IF THE LH OF JOBCOR (HIGHEST LOCATION LOADED WITH DATA BY LOADER) IS 140 OR GREATER. SEE EXAMPLE BELOW WHICH SETS VERSION NUMBER. SINCE NO DATA WAS LOADED ABOVE 137, SAVE WOULD NOT WRITE LOW FILE. JOBHRL IS SET BY THE LINKING LOADER AND SUBSEQUENT GETS, EVEN IF THERE IS NO FILE TO INITIALIZE THE LOW SEGMENT.

## 21. WRITING REENTRANT USER PROGRAMS

### A. DEFINING VARIABLES AND ARRAYS FOR THE LOW SEGMENT

THE LOADER SIMPLIFICATION MAKES IT SOMEWHAT MORE DIFFICULT TO DEFINE VARIABLES AND ARRAYS. THE EASIEST WAY TO DEFINE VARIABLES AND ARRAYS SO THAT THE RESULTING RELOCATABLE BINARY CAN BE LOADED ON A ONE OR TWO SEGMENT MACHINE, IS TO PUT THEM ALL IN A SEPARATE SUBPROGRAM AS INTERNAL GLOBAL SYMBOLS USING BLOCK 1 AND BLOCK N PSEUDO-OPS. ALL OTHER SUBPROGRAMS MUST REFER TO THE DATA AS EXTERNAL GLOBAL LOCATIONS. THUS, MOST REENTRANT PROGRAMS WILL HAVE AT LEAST TWO SUBPROGRAMS, ONE FOR DEFINITION OF LOW SEGMENT LOCATIONS AND THE OTHER (NEEDS HISEG PSEUDO-OP) FOR INSTRUCTIONS AND CONSTANTS FOR THE HIGH SEGMENT. SINCE PROGRAMS MUST BE SELF INITIALIZING, THEY MUST CLEAR THE LOW SEGMENT WHENEVER THEY ARE STARTED. (EVEN THOUGH THE MONITOR CLEARS CORE WHENEVER IT ASSIGNS IT TO A USER.)

USING BLOCK 1 AND BLOCK N PSEUDO-OPS WILL CAUSE THE LOADER TO LEAVE TRACKS IN THE JOB DATA AREA(LH OF JOBCOR) SO THAT A MONITOR SAVE COMMAND WILL NOT NEED TO WRITE THE LOW SEGMENT, SINCE IT CONTAINS NO INSTRUCTIONS, DATA, OR CONSTANTS. THIS IS ADVANTAGEOUS IN SHARABLE PROGRAMS FOR TWO REASONS, IT REDUCES THE NUMBER OF FILES IN OUR SMALL DECTAPE DIRECTORIES (22 FILES MAXIMUM) AND MORE IMPORTANT IT MEANS THAT I/O MUST BE DONE ONLY FOR THE FIRST USER'S GET (TO INITIALIZE HIGH SEGMENT) BUT NOT FOR ANY SUBSEQUENT USER'S GETS (EITHER HIGH OR LOW SEGMENT.)

AN EXAMPLE OF A REENTRANT PROGRAM:

LOW SEGMENT SUBPROGRAM:

```
TITLE LOW - EXAMPLE OF LOW SEGMENT SUB-PROGRAM

        JOBVER=137
        LOC     JOBVER
        3                       ;VERSION3
        RELOC   Ø
        INTERNAL LOWBEG,DATA,DATA1,DATA2,TABLE,TABLE1

LOWBEG:
DATA:   BLOCK   1
DATA1:  BLOCK   1
DATA2:  BLOCK   1

TABLE:  BLOCK   1Ø
TABLE1: BLOCK   1Ø
LOWEND=.-1              ;LAST LOCATION TO BE CLEARED
        END
```

HIGH SEGMENT SUBPROGRAM:

```
TITLE HIGH - EXAMPLE OF HIGH SEGMENT SUB-PROGRAM

        HISEG
        EXTERN  LOWBEG,LOWEND
        T=1
BEGIN:  SETZM   LOWBEG          ;CLEAR DATA AREA
        MOVEI   T,LOWBEG+1
        HRLI    T,LOWBEG
        BLT     T,LOWEND
        MOVE    T,DATA1         ;COMPUTE
        ADDI    1,1,
        MOVEM T, DATA2
          .
          .
          .
        END     BEGIN   ;STARTING ADDRESS
```

VERY FEW REENTRANT PROGRAMS REQUIRE THAT SOME LOCATIONS IN THE LOW SEGMENT CONTAIN SOME "CONSTANT" DATA WHICH DOES NOT CHANGE DURING EXECUTION. WHILE THIS PRACTICE IS TO BE DISCOURAGED, OCCASIONALLY THERE ARE GOOD REASONS FOR IT (AN INITIAL ASSEMBLER SYMBOL TABLE). SINCE THE INITIALIZATION OF THIS "CONSTANT" DATA NEED HAPPEN ONLY ONCE AFTER EACH GET RATHER THAN AFTER EACH START, THE TEMPTATION IS TO PUT

THESE CONSTANTS INTO THE SAME SUBPROGRAMS AS THE ONE CONTAINING THE DEFINITION OF THE VARIABLE DATA LOCATIONS. HOWEVER, THIS WOULD REQUIRE THAT SAVE WRITE THEM OUT AND GET LOAD THEM BACK IN AGAIN. SO, SUCH CONSTANT DATA SHOULD BE MOVED BY THE PROGRAMS FROM THE HIGH SEGMENT TO THE LOW SEGMENT AT THE SAME TIME THAT THE REST OF THE LOW SEGMENT IS BEING INITIALIZED WHENEVER THE PROGRAM IS STARTED. (THE EXTRA EXECUTION TIME IS NEGLIGIBLE.) OBVIOUSLY THERE IS AN EXCEPTION TO THIS RULE; IF THE AMOUNT OF CODE AND CONSTANTS IN THE HIGH SEGMENT NEEDED

TO INITIALIZE LOW SEGMENT CONSTANTS TAKES UP TOO MUCH ROOM IN THE HIGH SEGMENT, IT IS BETTER TO SUFFER THE PENALTY OF I/O INTO THE LOW SEGMENT ON EACH GET.

A SIMPLE RULE OF THUMB TO DECIDE BETWEEN THIS HIGH SEGMENT CORE SPACE VS LOW SEGMENT GET I/O TIME TRADEOFF IS TO PUT THE CODE IN THE HIGH SEGMENT IF IT DOESN'T PUT THE HIGH SEGMENT OVER THE NEXT 1K BOUNDARY.

B. A MORE CONVENIENT WAY TO WRITE REENTRANT PROGRAMS

V2 A SECOND WAY OF WRITING SINGLE SAVE FILE REENTRANT PROGRAMS HAS
V2 BEEN DEVELOPED IN WHICH THE SOURCE FILE CAN BE A SINGLE FILE
V2 INSTEAD OF TWO SEPARATE ONES AS INDICATED IN THE DISTRIBUTED
V2 RENMON.MAN WRITEUP (100-118-005-01).

V2 THIS MEMO IS BEING PRINTED IN THE SOFTWARE BULLETIN WHICH GOES
V2 TO ALL CUSTOMERS EVERY TWO WEEKS AND WILL BE ADDED TO RENMON.MAN
V2 TO MAKE 100-118-005-02.

V2 THE NEW TECHNIQUE IS MORE CONVENIENT, ALTHOUGH IT DOES INVOLVE
V2 CONDITIONAL ASSEMBLY AND THEREFORE PRODUCES TWO DIFFERENT
V2 RELOCATABLE BINARIES. A NUMBER OF CUSPS HAVE BEEN WRITTEN
V2 THIS WAY (TECO, LOGIN, LOGOUT, SRCCOM, CREF).

V2 THE IDEA IS TO HAVE A CONDITIONAL SWITCH, SAY PURE, WHICH IS 1
V2 IF REENTRANT ASSEMBLY, AND 0 IF NON-REENTRANT. THE DATA AREA IS
V2 PUT LAST IN THE SOURCE FILE FOLLOWING A LIT PSEUDO-OP AND CON-
V2 SISTS ONLY OF BLOCK 1 AND BLOCK N STATEMENTS, ALONG WITH DATA
V2 LOCATION TAGS. IF A REENTRANT PROGRAM IS DESIRED A LOC 140 IS
V2 ASSEMBLED, PLACING THE DATA AREA AT ABSOLUTE 140 IN THE LOW
V2 SEGMENT. BECAUSE OF THE LOC, NO OTHER RELOCATABLE PROGRAM CAN
V2 BE LOADED INTO LOW SEGMENT. THEREFORE THE PROGRAM SHOULD BE
V2 DEBUGGED AS A NON-REENTRANT PROGRAM WITH DDT SINCE DDT IS A LOW
V2 SEGMENT RELOCATABLE FILE. ALSO USE THE /B LOADER SWITCH TO PRO-
V2 TECT THE SYMBOLS. BECAUSE THE USUSAL WAY OF ASSEMBLY IS REEN-
V2 TRANT, PURE IS DEFINED TO BE 1 IF NOT ALREADY DEFINED.

V2 SINCE THE SYSTEM WASN'T DESIGNED TO RUN THIS WAY, THE PROGRAM
V2 MUST FIX UP ONE LOCATION IN THE JOB DATA AREA WHEN IT IS AS-
V2 SEMBLED TO BE REENTRANT SO THAT THE MONITOR WILL START AS-
V2 SIGNING BUFFERS AT THE END OF THE DATA AREA IN THE LOW SEGMENT
V2 RATHER THAN AT LOCATION 140. THIS CAN BE DONE BY CHANGING THE
V2 LH OF JOBSA BEFORE CALLI 0 (RESET) OR CHANGING C(JOBFF) AFTER
V2 CALLI 0. THE CHOICE WILL DEPEND ON HOW THE PROGRAM REINITIAL-
V2 IZES ITSELF ON ERRORS AND UPON COMPLETION. IT SHOULD BE REMEM-
V2 BERED THAT CALLI 0 MOVES THE LH OF JOBSA TO C(JOBFF). THE PRO-
V2 GRAM SHOULD NOT CHANGE THESE LOCATIONS, IF IT IS ASSEMBLED AS
V2 NON-REENTRANT SO THAT THE SYMBOL TABLE CAN BE PROTECTED USING
V2 THE LOADER /B SWITCH WHICH PLACES THE SYMBOLS NEXT TO THE LAST
V2 PROGRAM LOADED AND SETS LH OF JOBSA APPROPIATELY HIGHER. HENCE
V2 THIS CODE IS UNDER CONTROL OF PURE CONDITIONAL ASSEMBLY. NOTE
V2 THAT THE PERSON DEBUGGING DOES NOT NEED TO USE THE /B SWITCH
V2 IF HE DOESN'T WANT TO.

```
V2 TITLE DEMO - DEMO ONE SOURCE REENTRANT PROGRAM -VØØ1
V2 SUBTTL                T, HASTINGS     25 JUN 69
V2 JOBEVER=137
V2       LOC 137
V2       EXP ØØ1                    ;VERSION NUMBER

V2       INTERN JOBVER,PURE
V2       EXTERN JOBSA,JOBFF

V2 IFNDEF PURE,<PURE=1>             ;ASSUME REENTRANT IF PURE UNDEFINED
V2       IFN PURE,<HISEG>           ;TELL LOADER TO LOAD IN HIGH SEGMENT
V2                                  ;IF REENTRANT

V2 BEG:
V2 IFN PURE,<                       ;ONLY NEED IF REENTRANT
V2                                  ;(NOT NEEDED IF TWO FILES)
V2       MOVSI    T,DATAE           ;SET FIRST FREE LOCATION IN LOW SEG,
V2       HLLM     T,JOBSA           ;RESET SETS JOBFF FROM LH OF JOBSA
V2 >
V2       CALLI    Ø                 ;DO CALL RESET
V2       MOVE     T,JOBFF           ;ASSIGN AT LEAST ENOUGH CORE FOR DATA
V2       CALLI    T,11              ;CORE UUO
V2       JRST     ERROR
V2       MOVE     T,[XWD DATAB,DATAB+1]   ;NOW CLEAR DATA REGION
V2       SETZM    DATAB
V2       BLT      T,DATAE-1         ;LAST LOCATION CLEARED
V2       .
V2       .
V2       .
V2       .
V2       LIT                        ;PUT LITERALS IN HIGH SEG
V2 ;DATA AREA:
V2 IFN PURE,<LOC 14Ø>               ;START DATA AREA AT 14Ø IN LOW SEG IF REENTRA
V2 DATAB:                           ;FIRST LOCATION CLEARED EVERY START UP
V2 DATA:         BLOCK 1
V2 TABLE:        BLOCK 128
V2       .
V2       .
V2       .
V2 DATAE:        END      BEG       ;DEFINE FREE LOCATION
V2
```

22. MONITOR USE OF SWAPPING SPACE

THE RE-ENTRANT CAPABILITY IMPROVES SYSTEM THRUPUT BY REDUCING THE DEMANDS ON:

1. CORE MEMORY - SHARING
2. SWAPPING STORAGE - SHARING
3. SWAPPING CHANNEL - READS
4. SWAPPING CHANNEL - WRITES
5. STORAGE CHANNEL - READS (GET)

HOWEVER, 2 COMPETES WITH 5 IN THAT TO REDUCE THE DEMANDS 5, COPIES OF THE SEGMENTS ARE KEPT ON THE SWAPPING DEVICE, THEREBY INCREASING THE DEMAND FOR SWAPPING STORAGE (2). THE QUESTION NATURALLY ARISES, HOW DOES THE SYSTEM DETERMINE THIS SPACE-TIME TRADEOFF?

THE MONITOR ACHIEVES THE BALANCE DYNAMICALLY, AFTER THE OPERATOR ESTABLISHES THE SIZE OF THE SWAPPING SPACE WHEN

THE SYSTEM IS STARTED AND THE DISK IS REFRESHED (ONCE ONLY DIALOG). THE MONITOR ASSUMES THAT THERE IS NO SHORTAGE OF SWAPPING SPACE, AND SO KEEPS A SINGLE COPY OF AS MANY HIGH SHARABLE SEGMENTS IN THE SWAPPING SPACE AS THERE ARE HIGH SEGMENT NUMBERS, EVEN THOUGH NO ONE MAY BE USING THE HIGH SEGMENT. IF THE SEGMENT IS UNUSED, IT IS CALLED DORMANT, (THE MAX. NUMBER OF HIGH SEG. IS ESTABLISHED USING MONGEN AND IS EQUAL TO OR GREATER THAN THE NUMBER OF JOBS (COUNTING NULL JOB)). THUS DEMAND 5 IS MINIMIZED. HOWEVER, IF THE MONITOR CANNOT FIND CONTIGUOUS FREE SPACE ON THE SWAPPING DEVICE, IT WILL FRAGMENT THE HIGH OR LOW SEGMENT ON THE SWAPPING SPACE. THEN IF SWAPPING SPACE RUNS OUT, THE MONITOR WILL TRY DELETING A DORMANT SEGMENT AND WILL CONTINUE FRAGMENTING THE USER. WHEN THE DELETED SEG IS NEXT NEEDED (A GET OCCURRED), IT WILL BE GOTTEN FROM STORAGE DEVICE, INCREASING DEMAND 5.

TO RESTATE: THE MONITOR USES ALL THE SWAPPING SPACE AVAILABLE. IF IT RUNS OUT, IT INCREASES STORAGE CHANNEL READS (GET), IN ORDER TO OVERCOME THE SWAPPING SPACE SHORTAGE.

23. MONITOR USE OF CORE

THE SAME IDEA IS BEING EXTENDED TO PHYSICAL CORE IN BOTH THE SWAPPING AND NON-SWAPPING SYSTEMS. A DORMANT SEGMENT WILL STAY IN CORE UNTIL CORE IS NEEDED. IN SWAPPING SYSTEMS THE MONITOR WILL LEAVE AN ACTIVE WRITE LOCKED SEGMENT IN CORE EVEN THOUGH NO ONE IN CORE IS USING IT (SOME SWAPPED OUT USER IS USING OR ELSE IT WOULD BE DORMANT RATHER THAN IDLE).

IN OTHER WORDS, THE MONITOR ALLOCATES LOGICAL CORE INDEPENDENTLY FROM PHYSICAL CORE.

24. QUESTIONS IN MONGEN DIALOG

REENTRANT SOFTWARE? IF ANSWERED Y MONGEN WILL ASK:

HOW MANY MORE HIGH SEGMENTS THAN JOBS?

AND WILL GENERATE THE REQUIRED MONITOR TABLES TO HANDLE HIGH SEGMENTS. UNTIL A SYSTEM HAS MORE SHARABLE PROGRAMS THAN MAX NUMBER OF JOBS, THIS ANSWER SHOULD BE 0. ONLY WHEN THE MONITOR RUNS OUT OF HIGH SEGMENT NUMBERS WILL IT DELETE A DORMANT SEGMENT. FOUR MONITOR TABLES (JBTSTS, JBTADR, JBTSWP, JBTCHK) ARE LENGTHENED BY THE MAXIMUM NUMBER OF HIGH SEGMENTS (AT LEAST EQUAL TO MAXIMUM NUMBER OF JOBS, COUNTING NULL JOB, SO THAT EACH JOB CAN HAVE A DIFFERENT HIGH SEGMENT). ONE NEW TABLE (JBTSGN) IS GENERATED EQUAL TO THE MAX. NUMBER OF JOBS ALLOWED. TWO NEW TABLES (JBTDIR, JBTNAM) ARE GENERATED WHOSE LENGTH WILL BE EQUAL TO THE MAX. NUMBER OF HIGH SEGMENTS. THESE TABLES HAPPEN TO BE EXTENSIONS OF PRJPRG AND JBTPRG TABLES. SEE MONITOR SUB-PROGRAMS COMMON AND SEGCON FOR DETAILS.

## GLOSSARY OF TERMS (IN ORDER OF APPEARANCE)

**SEGMENT** — A CONTINUOUS REGION OF A USER'S CORE IMAGE WHICH THE MONITOR MAINTAINS IN PHYSICAL CORE AN/OR ON THE SWAPPING DEVICE. MOST FIT IN CORE ALL AT ONCE.

**SHARABLE SEGMENT** — A SEGMENT WHICH IS APPEARING OR HAS POTENTIAL OF APPEARING IN MORE THAN ONE USER CORE IMAGE AT THE SAME TIME. SHARABLE SEGMENTS ALWAYS HAVE NAMES UNTIL THEY ARE SUPERCEDED. THEY ARE ALWAYS INITIALIZED FROM FILES

**NON-SHARABLE SEGMENT** — A SEGMENT FOR WHICH EACH USER HAS HIS OWN COPY. NON-SHARABLE SEGMENTS NEVER HAVE NAMES EVEN IF INITIALIZED FROM A FILE. THEY MAY ALSO BE CREATED BY CORE OR REMAP UUO.

**LOW SEGMENT** — A 1 TO 256K NON-SHARABLE SEGMENT STARTING AT USER 0. ALWAYS REQUIRED.

**HIGH SEGMENT** — A 0 TO 128K SHARABLE OR NON-SHARABLE SEGMENT STARTING AT USER 400000 OR END OF LOW SEGMENT, WHICH EVER IS GREATER. OPTIONAL.

**RE-ENTRANT PROGRAM** — A TWO SEGMENT PROGRAM COMPOSED OF A SHARABLE AND NON-SHARABLE SEGMENT

**NON-RE-ENTRANT PROGRAM** — A ONE OR TWO SEGMENT PROGRAM IN WHICH NEITHER SEGMENT IS SHARABLE

**IMPURE SEGMENT** — A SEGMENT WHICH IS OR CAN BE MODIFIED WHILE PART OF A USER CORE IMAGE

**PURE SEGMENT** — A SEGMENT WHICH CANNOT BE MODIFIED WHILE PART OF A USER CORE IMAGE

**FILE** — A NAMED OR UNNAMED COLLECTION OF 36 BIT WORDS (INSTRUCTIONS AND/OR DATA). LENGTH NOT RESTRICTED BY SIZE OF CORE. ONE OF THE USES OF FILES IS TO INITIALIZE SEGMENTS WHEN THEY ARE CREATED WITH INSTRUCTIONS AND/OR DATA.

**NAMED FILE** — A NAMED COLLECTION OF 36 BIT WORDS (INSTRUCTIONS AND/OR DATA) STORED BY THE FILE SYSTEM AND THE STORAGE DEVICE (BURROUGHS DISK, BRYANT DISK, OR DECTAPE)

**DIRECTORY NAME** — PROJECT, PROGRAMMER NUMBER PAIR WHICH UNIQUELY IDENTIFIES A DIRECTORY; THE DEVICE NAME IN THE CASE OF DECTAPE OR MAGTAPE

**FILE NAME** — 1 TO 6 ALPHANUMERIC CHARACTERS CHOSEN BY THE USER TO IDENTIFY THE FILE

FILE EXTENSION — 1 TO 3 ALPHANUMERIC CHARACTERS USUALLY CHOSEN BY THE PROGRAM TO DESCRIBE THE CLASS OF INFORMATION IN FILE

STORAGE DEVICE — THE DEVICE USED TO STORE NAMED FILES BY THE GET, R, OR RUN COMMANDS. IF THE FILE IS MARKED AS A SHARABLE (EXTENSION = "SHR"), THE MONITOR WILL GIVE THE SEGMENT THE SAME NAME AS THE FILE. THIS IS THE ONLY WAY THAT A SEGMENT CAN BE SHARED.

CREATE — A FILE IS CREATED WHEN IT HAS BEEN OPENED FOR WRITING, WRITTEN AND CLOSED FOR THE FIRST TIME. ONLY ONE USER MAY BE CREATING THE FILE AT A TIME.

RECREATE — A FILE IS RECREATED WHEN IT HAS BEEN OPENED FOR WRITING, WRITTEN AND CLOSED ONE OR MORE SUBSEQUENT TIMES. THE OLDER COPY IS DELETED WHEN ALL READERS ARE FINISHED. ONLY ONE USER CAN BE RECREATING THE FILE AT A TIME.

UPDATE — A FILE IS UPDATED, OPENED FOR READING AND WRITING, ONE OR MORE BLOCKS REWRITTEN IN PLACE, AND CLOSED. ONLY ONE USER MAY BE UPDATING THE FILE AT A TIME.

VESTIGAL JOB DATA AREA — THE FIRST 10 OCTAL LOCATIONS OF THE HIGH SEGMENT USED TO CONTAIN DATA FOR INITIALIZING CERTAIN LOCATIONS IN THE JOB DATA AREA.

CREATE — A SEGMENT IS CREATED BY THE CORE OR REMAP UUO. LOGICALLY, GET, R, AND RUN COMMANDS ALSO DO CORE UUO'S.

MEDDLING — A PROGRAM WITH A SHARABLE HIGH SEGMENT IS SAID TO BE MEDDLED WITH IF THE USER HAS DONE SOMETHING WHICH PREVENTS THE PROGRAM FROM BEING IN COMPLETE AND PREDICTABLE CONTROL OF ITSELF. A PROGRAM WHICH HAS BEEN MEDDLED WITH, CANNOT BE ALLOWED TO TURN OFF ITS USER MODE WRITE PROTECT BIT OR CHANGE ITS HIGH SEGMENT CORE ASSIGNMENT WITH THE CORE UUO, EXCEPT REMOVE IT ENTIRELY. SEE SECTION VI MODIFYING SHARED SEGMENTS DURING EXECUTION

DORMANT — A SHARABLE HIGH SEGMENT KEPT ON SWAPPING SPACE AND POSSIBLY CORE WHICH IS IN NO USER ADDRESSING SPACE.

IDLE — A SHARABLE HIGH SEGMENT WHICH IS IN CORE BUT FOR WHICH NO USERS IN CORE ARE USING. HOWEVER AT LEAST ONE SWAPPED OUT USER IS USING, ELSE IT WOULD BE A DORMANT SEGMENT.

**READER'S COMMENTS**

PDP-10/40 PDP-10/50
TIME SHARING MONITORS
REFERENCE MANUAL
DEC-T9-MTZA-D

Digital Equipment Corporation maintains a continuous effort to improve the quality and usefulness of its publications. To do this effectively we need user feedback: your critical evaluation of this document. Please give specific page and line references when appropriate.

ERRORS NOTED IN THIS PUBLICATION: _____

_____

_____

_____

_____

_____

_____

SUGGESTIONS FOR IMPROVEMENT OF THIS PUBLICATION: _____

_____

_____

_____

_____

_____

_____

_____

_____

DEC also strives to keep its customers informed about current DEC software and publications. Thus, the following periodically distributed publications are available upon request. Please check the publication(s) desired.

☐ PDP-10 User's Bookshelf, a bibliography of current programming documents.

☐ Program Library Price List, a list of available software documents and programs.

Name _____ Date _____

Organization _____

Please describe your position _____

Street _____

City _____ State _____ Zip Code _____

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

**Digital Equipment Corporation**
**Software Information Services**
**146 Main Street, Bldg. 3-5**
**Maynard, Massachusetts 01754**

Digital Equipment Corporation
Maynard, Massachusetts

digital

printed in U.S.A.