TSS/8 SYSTEM MANAGER'S GUIDE FOR THE PDP-8/I TIME-SHARING SYSTEM

DIGITAL EQUIPMENT CORPORATION • MAYNARD, MASSACHUSETTS

1st Edition, August 1970
2nd Printing, July 1971

CONTENTS

CONTENTS (Cont)

CONTENTS (Cont)

ILLUSTRATIONS

ILLUSTRATIONS (Cont)

TABLES

Preface

This Guide is intended for the TSS/8 System Manager - the person responsible for operating and maintaining the TSS/8 software system. The Guide describes how to load and to build the Monitor, how to build a library of system programs, how to define account numbers and passwords, how to copy the newly built system onto DEC-tape, how to maintain the system, and much more.

This information is not required by the TSS/8 user. In fact, the TSS/8 user should not have access to this Guide without the System Manager's consent.

Part One describes the operation of a TSS/8 system. Chapter 1 introduces the Guide and the System. Chapter 2 explains in depth how to build a complete, operational TSS/8 software system. First familiarize yourself with system building and then refer to the summary in Appendix B. Chapter 3 describes how to maintain an operational system - the system library, user files, account numbers, passwords, etc. Chapter 4 describes methods for modifying the TSS/8 Monitor.

Part Two contains a description of the actual TSS/8 Monitor and how it is used to service multiple users simultaneously. It also describes the Monitor's internal tables. This information is useful to installations where Monitor will be modified.

Decimal numbers are used throughout this document except where indicated - 1011_8 is octal, 1011_2 is binary, 1011 is decimal.

Part One

Operating TSS/8

Chapter 1
Introduction

1.1 USE OF THIS MANUAL

This Guide describes those aspects of the TSS/8 system which are of interest only to the System Manager, the person responsible for managing and/or operating the TSS/8 system. This Guide is a companion piece to the Chapter on TSS/8 in Introduction to Programming 1970 which describes in detail the operation of the system and its many features available to the on-line TSS/8 user.

The TSS/8 system consists of a comprehensive collection of system software (computer programs) and a system hardware configuration (computer and peripherals). TSS/8 is available in the various hardware configuration shown in Appendix D. The system software supplied with a configuration is tailored specifically for that configuration. Monitor, however, is structured so that standard loading and operating procedures apply for all configurations; these procedures are explained in this Guide.

1.2 PLANNING A TSS/8 INSTALLATION

TSS/8 is a compact time-sharing system that does not require the refined environment of a big computer. Typically, TSS/8's are installed in the location where they will be used - a classroom or small computer lab. The following sections provide the information needed to plan the installation.

1.2.1 Hardware Requirements

TSS/8 systems are made up of standard DEC cabinets. These are 71-7/16 inches high, 21-11/16 inches wide, and 30 inches deep. TSS/8 systems range from two to six of these cabinets. A small system with one or two disks, 12-16K of core, the PT08 Teletype ® interfacing, and no DECtape, fits in two cabinets. A third or fourth disk requires an extra cabinet. DECtapes require a cabinet. If the DC08 communications equipment is used, it is housed in its own cabinet. Adding the 689 Dataphone® control option to the DC08 adds still another cabinet. Average TSS/8's are three to four cabinets

® Teletype is a registered trademark of the Teletype Corporation; Dataphone is a registered trademark of Bell Telephone Corporation.

1.2.2 Power Requirements

A TSS/8 installation requires two, and possibly three, 30-amp power outlets which will accept Hubbell three-prong twist-lock plugs. All power cords are 25 feet long.

NOTE

Outlets for 50-cycle power should be 20 amps.

The first cord services the processor and options except for the disk. For all but the largest TSS/8s this cord will service the processor and all such options. For large systems, where total power requirements for these items exceed 30 amps, a second cord is required. This power, whether from one cord or two, is controlled by the power on/off switch on the computer console.

The system disk, or disks, is always powered by a separate cord since it should never be shut down, even when the rest of the system is turned off.

1.2.3 Teletypes, Dataphones, and Cables

Although the TSS/8 hardware itself is a single unit, installations will have either terminals or Dataphones located nearby. These must be physically cabled to the machine. Normally, all Dataphones will be together; hence they will require common cable lengths. (For PT08 systems, the Dataphone cable is the PT08F, for DC08Bs it is the BC01C-25, for 689s it is the 689LM.) These cables will be 25 feet long unless specially ordered. Local Teletypes, i.e., those in the computer room hard-wired to the machine, may require varying cable lengths depending on how they are placed around the room. Teletype cables are 12 feet long unless specially ordered. (Local Teletypes plug into PT08s or into DC08Bs.) To be sure that all cables are of appropriate length, a map of the installation should be made. A typical installation is shown in Figure 1-1.

1.2.4 Environment

It is suggested that TSS/8 be installed in an environment where the temperature and humidity can be controlled. TSS/8 does not require a formal computer room, but it should not be subjected to rapid changes in temperature or humidity. Excessive dust or smoke should always be avoided.

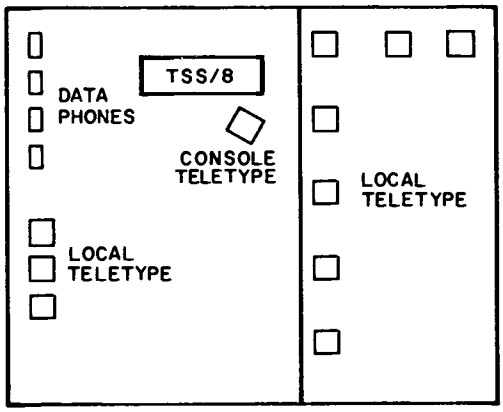


Figure 1-1 Typical TSS/8 Installation

Chapter 2
Building a TSS/8 System

Building a TSS/8 software system takes five steps:

1. Loading Monitor onto the disk.
2. Refreshing the disk.
3. Building the system program library.
4. Defining account numbers and passwords.
5. Dumping the newly built system to DECtape.

Steps 1, 2, and 4 are identical for all TSS/8 systems. Step 5 is done only for systems which include DECtape. Step 3 is done in one of two ways, depending on whether DECtape is available.

a. If the system does not include DECtape, all TSS/8 software is distributed on paper tape. These tapes include a copy of TSS/8 BUILD, five (5) Monitor tapes, a copy of XDDT, and a copy of TSS/8 PIP. These tapes are all BIN format tapes. In addition, the System Library Programs are distributed as SAVE format paper tapes. TSS/8 BUILD is used to load the rest of the BIN format tapes (as explained in this manual). PIP is used to load the SAVE format tapes.

b. If the system does include DECtape, only the BIN format tapes are distributed. Instead of PIP, a copy of COPY, in BIN format, is included. The System Library Programs are distributed on a single TSS/8 format DECtape.

The description of step 3 in this chapter is divided into two parts. Follow the section which applies to your configuration.

The program TSS/8 BUILD tells what to do to build the Monitor (Step 1). Then, if specified, the remaining four steps that build a complete system are printed as instructions. A complete printout of the TSS/8 BUILD dialogue is in Appendices B and C.

2.1 INITIALIZING THE SYSTEM

Before using the computer system, all units should be initialized; that is, all switches and controls should be set as specified below.

1. Main power cord is properly plugged in.

2. Teletype is turned ON.
3. Low-speed punch is OFF.
4. Low-speed reader is set to FREE.
5. Computer POWER key is ON.
6. PANEL LOCK is unlocked.
7. Console switches are set to DF = 000 IF = 000 SR = 0000
   SING STEP and SING INST are not set.
8. High-speed punch is OFF.
9. DECtape REMOTE lamps are OFF.

The system is now initialized and ready for the RIM Loader to be loaded into core using the PDP-8/I console switches.

2.1.1 Read-In Mode (RIM) Loader

When a computer in the PDP-8 family is first received, its core memory is completely demagnetized. The computer "knows" absolutely nothing, not even how to receive input.

The RIM Loader, the first program loaded into the computer, is loaded using the console switches. The RIM Loader instructs the computer to receive and store, in core, data punched on paper tape in RIM-coded format. Table 2-1 contains the RIM Loader Program.

The procedure for loading (toggling) the RIM Loader into core is illustrated in Figure 2-1. Once RIM has been loaded, a good programming practice is to verify that all instructions have been stored properly. Figure 2-2 illustrates how to verify storage as well as how to correct an incorrectly stored instruction.

Table 2-1
RIM Loader Program (High-speed version)

Location	Instruction	Location	Instruction
7756	6014	7767	6011
7757	6011	7770	5367
7760	5357	7771	6016
7761	6016	7772	7420
7762	7106	7773	3776
7763	7006	7774	3376
7764	7510	7775	5357
7765	5374	7776	0000
7766	7006		

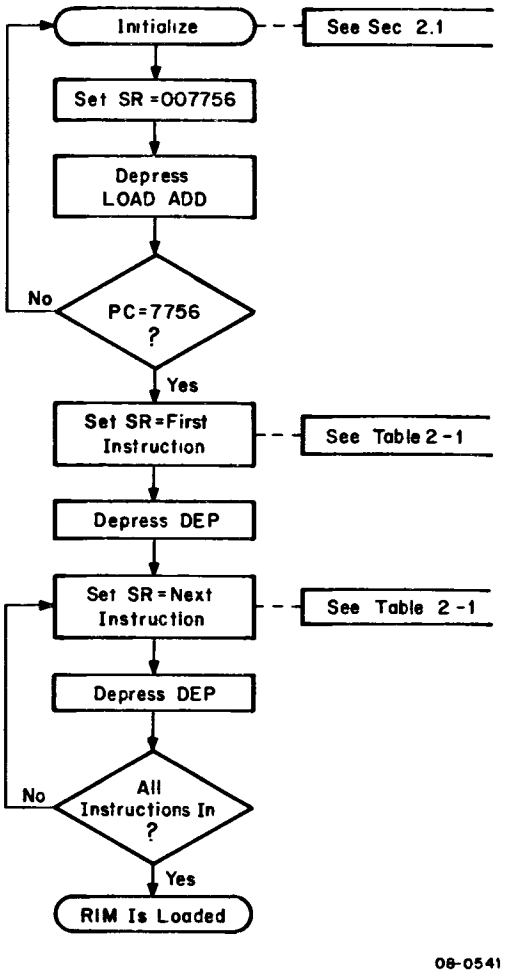


Figure 2-1 Loading the RIM Loader

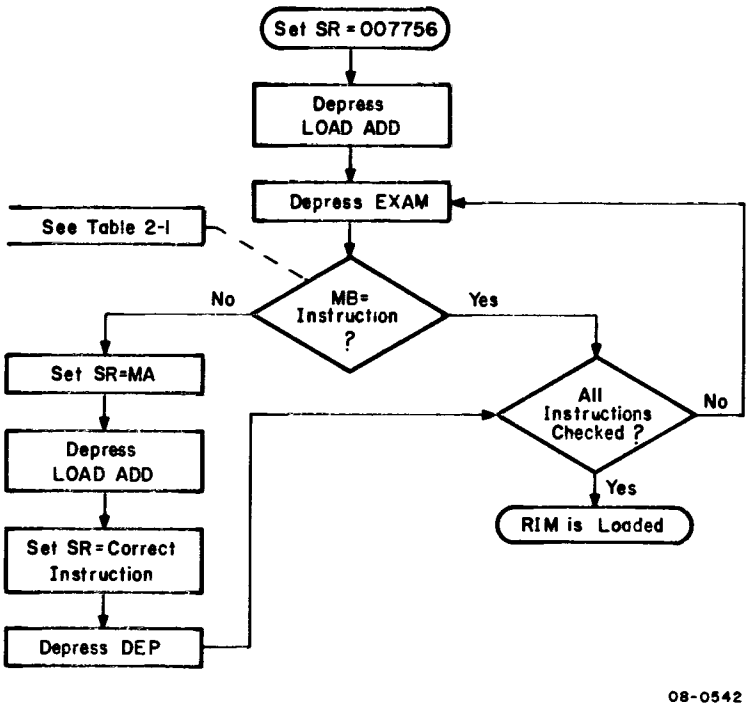


Figure 2-2 Checking the RIM Loader

2.1.2 Binary (BIN) Loader

The BIN Loader is a short utility program which, when in core, instructs the computer to read binary-coded data punched on paper tape and to store data in core memory. BIN is used to load the TSS/8 BUILD program and INIT program (Chapter 3).

BIN is furnished on punched paper tape in RIM-coded format. Therefore, RIM must be in core before BIN can be loaded. Figure 2-3 illustrates the steps necessary to properly load BIN.

2.2 LOADING MONITOR

After initializing the computer and loading the RIM and BIN loaders, you should load and use the TSS/8 BUILD program to load the Monitor subprograms. Monitor is composed of seven subprograms supplied on binary-coded paper tape and identified as listed below.

1. SI System Interpreter
2. FIP File Phantom
3. XDDT Debugging Utility
4. INIT Initializer

5.	TS8	Field 0 Resident Monitor
6.	TS8II	Field 1 Resident Monitor
7A.	PIP	Peripheral Interchange Program*
7B.	COPY	DECtape Copy Program**

BUILD is used to load each of these subprograms into core and then to transfer each onto the disk.

BUILD is loaded into core using the BIN Loader as illustrated in Figure 2-4.

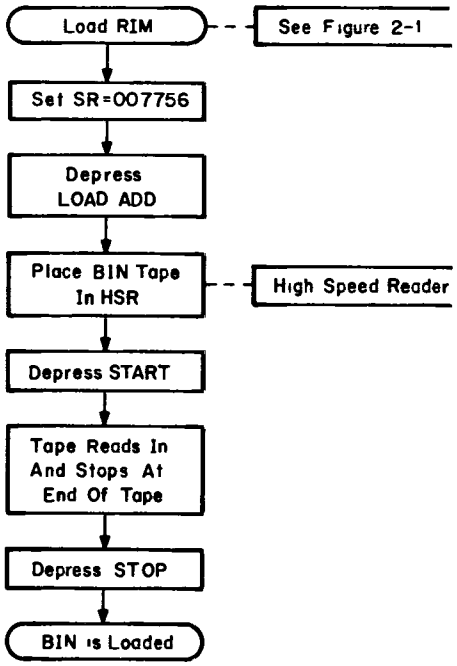


Figure 2-3 Loading the BIN Loader

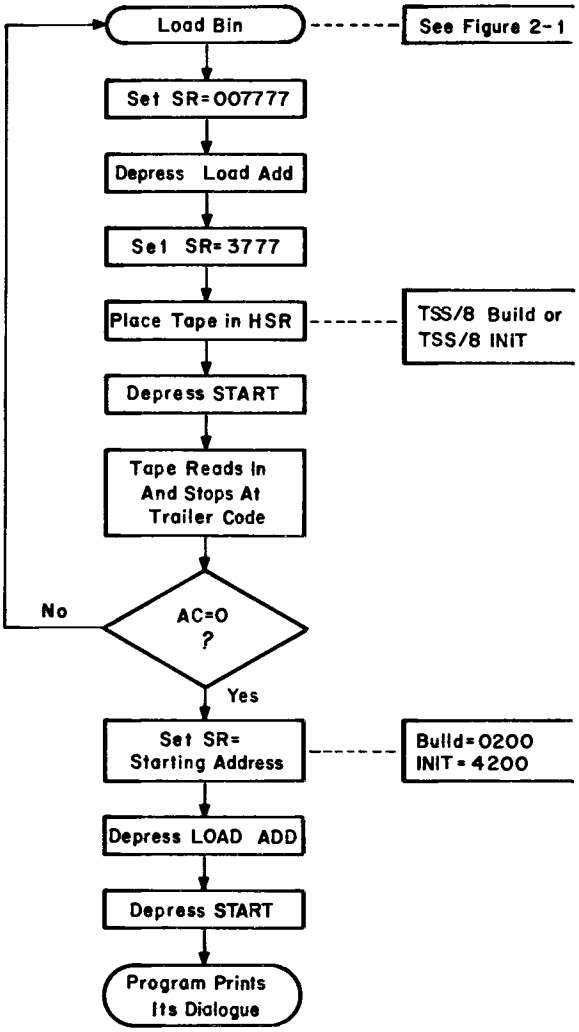


Figure 2-4 Loading and Starting TSS/8 BUILD or TSS/8 INIT

*If system does not include DECtape.
**If system includes DECtape.

When BUILD is started (at location 0200 as illustrated on Page 2-5) it will begin its dialogue by printing:

```
TSS/8 BUILD--(REVISED 2/15/70)

IS DISK AN RS08? (Y IF RS08. N IF DF32): Y
DOES THE SYSTEM INCLUDE DECTAPE? (Y OR N): N
```

Reply by typing Y (for yes) or N (for no) and then terminate each line by typing the RETURN key.

NOTE

All lines are terminated with the RETURN key, unless otherwise specified.

After you type a reply, Monitor prints the following:

```
YOU SHOULD HAVE THE FOLLOWING BIN FORMAT PAPER TAPES:

1) SI    ... TSS/8 SYSTEM INTERPRETOR
2) FIP   ... TSS/8 FILE PHANTOM
3) XDDT  ... TSS/8 DEBUGGING UTILITY
4) INIT  ... TSS/8 INITIALIZOR
5) TS8   ... FIELD 0 RESIDENT MONITOR
6) TS8II .. FIELD 1 RESIDENT MONITOR
7A)PIP   ... PERIPHERAL INTERCHANGE PROGRAM *** IF NO DECTAPE ***
7B)COPY  ... DECTAPE COPY PROGRAM *** IF DECTAPE ***

THE BUILDING PROCESS IS DONE IN FIVE STEPS

1. LOADING MONITOR ONTO THE DISK
2. REFRESHING THE DISK
3. BUILDING UP THE SYSTEM LIBRARY
4. BUILDING UP THE FILE OF VALID PASSWORDS
5. DUMPING THE SYSTEM TO DECTAPE
   (IF DECTAPE IS ON THE SYSTEM.)

ONLY THE FIRST STEP IS DONE UNDER THE CONTROL OF TSS/8 BUILD.
STEP 2 IS DONE UNDER CONTROL OF INIT. STEPS 3 AND 4
ARE DONE WHILE THE TIME-SHARING SYSTEM IS ON-LINE. STEP 5
IS DONE BY STOPPING THE SYSTEM AND RECALLING INIT.

STEP ONE ---

AS EACH TAPE NAME IS TYPED OUT, MOUNT THE CORRESPONDING TAPE IN
THE HIGH SPEED READER AND TYPE CARRAIGE RETURN.

IF, FOR SOME REASON, A TAPE IS INCORRECTLY READ,
BUILD WILL TYPE '?' AND REQUEST THE SAME TAPE AGAIN.

SI:
```

NOTE

To hasten the building process, you may suppress dialogue by starting BUILD at 201 instead of 200.

BUILD is now waiting for the SI (System Interpreter) tape to be placed in the high-speed, paper-tape reader.

Paper tapes are hand-placed in the high-speed reader as explained below (see Introduction to Programming, Chapter 4, for complete details).

1. Place the paper tape in the right-hand bin so that the beginning of the tape will pass over the sensors first.
2. Place several folds of leader tape in the left-hand bin with the tape passing under the tape retainer cover.
3. Close the retainer cover over the tape so that the feed holes are engaged in the teeth of the sprocket wheel.
4. Advance the tape until leader code* is directly over the sensors.

Once the paper tape has been properly placed in the reader with leader code directly over the sensors, type the RETURN key, and the paper tape will pass through the reader and stop at the beginning of trailer code. BUILD ensures that the data on the tape has been correctly received, i.e., that the program is complete, and that no data has been lost during the read in. If the data was not read in correctly, BUILD prints a question mark (?) and the name of the incorrect program (in this case SI), and waits for the same tape to be loaded again. For example, if the SI tape had not been correctly read in, the following would have been printed on your console.

```
SI:
? SI:
```

When the tape is read in properly, BUILD prints the name of the next tape to be placed in the high-speed reader. For example: BUILD prints

```
SI:
FIP:
```

and then waits for the FIP tape to be placed in the reader and the RETURN key to be typed, exactly as when loading the SI tape. When FIP is correctly loaded, BUILD prints XDDT: and waits for the

*See Introduction to Programming 1970

XDDT tape to be placed in the reader and the RETURN key to be typed. This process continues until all Monitor subprogram tapes have been loaded. Then BUILD asks if an explanation of step 2 is required.

If all Monitor subprograms had been loaded without difficulty, the printout on the console would appear as shown below, starting with STEP ONE --

```
STEP ONE ---

AS EACH TAPE NAME IS TYPED OUT, MOUNT THE CORRESPONDING TAPE IN
THE HIGH SPEED READER AND TYPE CARRIAGE RETURN.

SI:
FIP:
XDDT:
INIT:
TS8:
TS8II:
PIP:

EXPLAIN STEP 2? (Y OR N):
```

Monitor is now built and loaded onto the disk. BUILD will continue to print the dialogue explaining the remaining steps of building a complete TSS/8 system. However, BUILD does not interact with the operator in performing the remaining steps as it did when building Monitor.

If you want BUILD to print the dialogue for step 2, type Y and the RETURN key. If you do not want the step 2 dialogue, type N and the RETURN key. BUILD will omit explaining step 2 and respond with

```
EXPLAIN STEP 3? (Y OR N):
```

to which you type Y or N, as with step 2 above. This sequence will continue until you reply to

```
EXPLAIN STEP 5? (Y OR N):
```

If you reply with Y to the above, BUILD will explain the process of step 5. When it has completed execution, the builder types:

```
END OF TSS/8 BUILD
```

It then loads and starts the initializer program, INIT, which prints:

```
LOAD, DUMP, REFRESH, START?
```

and waits for you to specify which of the four operations you intend to perform, as explained in the following sections.

2.3 REFRESHING THE DISK AND STARTING THE SYSTEM

This section on refreshing only applies to systems which have just been built on the disk with TSS/8 BUILD.

After Monitor has been built, the next step is to refresh the disk, i.e., to clear the disk of all data except Monitor so that a new system can be constructed.

The first 20K of disk storage is used to hold an image of Monitor. The next portion of disk for user program swapping area - each simultaneous user on the system has a 4K swap area on the disk which holds his active program while it is not in core. Thus, for a 16-user system, there is 64K of disk dedicated to swap area. All disk storage above the swap area and any additional disks in the system are available for on-line storage of system and user programs (files), and file and system directories, thus all this data is deleted during refresh time.

Monitor's initializer (INIT) subprogram is used to refresh the disk. BUILD transfers control to INIT upon completion of its dialogue so that INIT automatically prints:

```
LOAD, DUMP, REFRESH, START?
```

and waits for you to specify the desired operation. In this case, type the word REFRESH and then the RETURN key. (The other three operations are explained later.)

To ensure that you don't refresh the disk accidently, INIT will print REFRESH? and wait for you to type YES or NO. If NO, INIT will ignore your request to refresh and repeat LOAD, DUMP, REFRESH, START and wait for another operation to be specified. If YES, INIT prints its query for you to assign the new system and library passwords. Refreshing the disk deletes all valid passwords, even the system and library passwords. Therefore, to allow system files to be built up again, these two special passwords must be defined at this time. The first is the system password (INIT automatically assigns it to account number 0001) which allows access to accounting information and allows user passwords to be defined. The second is the library password (INIT automatically assigns it to account number 0002) which allows access to the system program library. These two passwords should be known only to you, unless you wish to provide them to others, e.g., the system operator.

The system and library passwords must be made up of four alphanumerics. After specifying the passwords, your printout to this point might appear as follows.

```
LOAD, DUMP, REFRESH, START? REFRESH
REFRESH? YES
SYSTEM PASSWORD? TSS8
LIBRARY PASSWORD? LBRY

LOAD, DUMP, REFRESH, START?
```

This time you should answer by typing START, and INIT will print

```
LOGIN MESSAGE?
```

Your reply should be Y (for yes) or N (for no), depending on whether you want to have a message printed on each console whenever it is logged in. The message may be a greeting, caution, special instruction, or anything else you desire as long as the message has no more than 128 characters (counting spaces).

When you reply by typing N, INIT will print its next query, LOAD EXEC DDT?. However, when your reply is Y, INIT will respond with

```
END WITH ALT MODE:
```

and position the console paper so that your message can be typed on the next line. After typing the message, you should end by typing the ALT MODE key. (ALT MODE is labeled ESC on some Teletypes.) Your printout to this point and an example message would appear as shown below.

```
LOAD, DUMP, REFRESH, START? START
LOGIN MESSAGE? Y
END WITH ALT MODE:
CONGRATULATIONS. YOU ARE NOW ON-LINE WITH TSS/8.
   REPORT ANY PROBLEMS TO THE SYSTEM SUPERVISOR. $
LOAD EXEC DDT?
```

When you type the ALT MODE key at the end of your message a dollar sign ($) will be printed as shown above. INIT then prints its next query, asking if you plan to load the XDDT program to debug or modify Monitor (explained in Chapter 4). You should answer with N (for no).

The next query asks you to specify the number of core fields available for user programs. Remember that fields 0 and 1 are reserved for Monitor. Therefore, the total number of core fields in the configuration minus two should be your reply. For example, for a 16K (4 core fields) configuration your reply would be 2. INIT then prints:

```
MONTH-DAY-YEAR?
```

and waits for the response of the current month, day, and year separated with colons. Next, INIT asks HR:MIN - You answer with the time of day expressed in military time using a 24-hour clock. separate the hour and minutes with a colon. For example, 9:45 a.m. is entered 9:45, 1:30 p.m. is 13:30, 9:45 p.m. is 21:45, and 12 o'clock midnight is 24:00.

Your printout to this point of step 2 might appear as follows.

```
LOAD, DUMP, REFRESH, START? START
LOGIN MESSAGE? Y
END WITH ALT MODE:
CONGRATULATIONS. YOU ARE NOW ON-LINE WITH TSS/8.
   REPORT ANY PROBLEMS TO THE SYSTEM SUPERVISOR. $
LOAD EXEC DDT? N
# USER FIELDS - 2
MONTH-DAY-YEAR: 9:30:69
HR:MIN - 23:23
```

After entering the time of day and terminating the line with the RETURN key, INIT transfers control to Monitor. TSS/8 is now on-line.

2.4 BUILDING THE SYSTEM LIBRARY

Building up the system library is done while the system is on-line, i.e., operational and running. You should now log in with the system library account number and password and use the PIP (or COPY) sub-program to load system programs from paper tape via the high-speed reader (or DECtape) onto the disk.

2.4.1 Logging In

Any system console may be used when logging in. Set the console LINE-OFF-LOCAL switch to LINE, then type the RETURN key. Monitor will print a dot (period) at the left margin of the console paper.

You should Log in with account number 0002 and the library password specified during refresh time.

The command LOGIN and account number and password will not echo (print) on the console paper if valid. Hower, if they are invalid (not defined), Monitor will echo exactly what was typed, followed by a question mark and then by another dot; for example, when you make a typing error when logging in.

When the login is accomplished, Monitor prints the version number of the TSS/8 Monitor being used, the job number assigned to you when logging in, the number of the console being used, and the current time of day. (These entries are fully explained in the Time-Sharing System User's Guide.)

The login message is printed next, followed by Monitor's dot indicating that the building session has been successful to this point. For example:

```
.
TSS/8.21C  JOB 01   K00      23:23:07
CONGRATULATIONS. YOU ARE NOW ON-LINE WITH TSS/8.
   REPORT ANY PROBLEMS TO THE SYSTEM SUPERVISOR.
.
```

Note that the LOGIN command, account number, and password is not printed on the console.

2.4.2 Loading System Programs from Paper Tape with PIP

TSS/8 PIP is used to load DEC-supplied system program paper tapes (BASIC, FOCAL, EDIT, CAT, PIP, LOADER, etc.) and any other program of general use to TSS/8 users. TSS/8 PIP was loaded as one of Monitor's seven subprograms. PIP is brought from the disk into core and started by typing START 0 (start execution at core location zero) in response to Monitor's dot. START is the first Monitor command to be used. Command lines as well as all input to PIP are always terminated by typing the RETURN key.

When started, PIP prints INPUT: and waits for you to specify input. Now type the RETURN key to indicate that input is to be from paper tape. When PIP prints OUTPUT: you place the system program tape in the high-speed reader (with leader code directly over the sensing holes) and then type the file name of the program being loaded. TSS/8 users will use the file name to request the program. For example, if BASIC is being loaded, the printout would appear as follows:

```
.START 0

 INPUT:
OUTPUT:BASIC
OPTION:
```

Your reply to OPTION: should always be S (and the RETURN key) when loading programs into the system library. The S denotes that the incoming program is coded in TSS/8's SAVE format.

Monitor uses the SAVE format primarily for loading and backing up system library files (see Sections 3.4 and 3.6).

When the OPTION: line is terminated, the program tape begins to pass through the reader and stops at the trailer code. If a program is not correctly received, PIP prints "LOAD ERROR", and repeats its request for INPUT:

```
 INPUT:
OUTPUT:BASIC
OPTION:S
LOAD ERROR

 INPUT:
```

When PIP prints INPUT: without the error message, you should load the next system program tape. Your response to OUTPUT: is the name of the program being loaded.

When all system programs have been loaded, you should reply to the last INPUT: by typing CTRL/B and then the S key to return control to Monitor. Monitor then prints a dot and waits for a command. Some installations may not choose to load all System Library Programs. However, one program, LOGOUT, must be loaded on all systems.

Once the library has been loaded with all desired programs, log out. Until you need to change the contents of the system library, you need not log in with the library password again. See Appendix B for a complete example.

2.4.3 Loading System Programs from DECtape with COPY

TSS/8 COPY is used to load the DEC-supplied system program from the System Library DECtape. This tape includes all TSS/8 System Library Programs (BASIC, FOCAL, EDIT, CAT, etc.) along with a number of sample and demonstration programs which may be useful in testing and exercising the system. COPY has been loaded during step 1. To use it, type

```
•START Ø
```

which commands TSS/8 to bring COPY in and start it. START is the first Monitor command to be used. Command lines as well as all input to COPY are always terminated by typing the RETURN key. At the same time, mount the TSS/8 DECtape on tape unit zero (0). Make sure it is in the on-line (REMOTE) mode.

When started, COPY prints OPTION- and waits to be told what to do. Type LIST to obtain a listing of all programs on the tape. COPY then responds DEVICE-. Type D0 to indicate where the tape is mounted. The whole process will appear on the Teletype as follows:

```
•START Ø

OPTION- LIST
DEVICE- DØ

642. FREE BLOCKS

NAME       SIZE    DATE
BASIC .SAV   66   2-MAR-7Ø
FOCAL .SAV   32  12-MAR-7Ø
COPY  .SAV   2Ø   3-MAR-7Ø
EDIT  .SAV   14   2-MAR-7Ø
etc.
```

COPY continues to list file names.

After printing all file names, COPY again prints OPTION-. The exact contents of the System Library DECtape may change as updates are made. In all cases the directory will start with files whose names are followed by a period and SAV (.SAV). This is the file extension which indicates the type of file. The files with an extension of SAV are the System Library Programs. They are to be loaded on every system. Installations may also wish to load some of the sample programs.

To load a file, type COPY in response to OPTION-. COPY will then print INPUT-. Respond with D0: followed by the file name. For example: D0:BASIC. Do not include the file extensions. Just type the file name. COPY will respond with OUTPUT-. Again type the file name. COPY will load the file from the DECtape. For example:

```
OPTION-COPY
INPUT - DØ:BASIC
OUTPUT - BASIC

OPTION -
```

Continue this process until all desired files are loaded. Some installations may not choose to load all System Library Programs. However, one program, LOGOUT, must be loaded. When done, respond to OPTION- with EXIT. The System Library is now loaded. To complete this step, type LOGOUT.

```
OPTION- EXIT
↑BS
.LOGOUT
```

See Appendix C for a complete example.

2.5 DEFINING ACCOUNT NUMBERS AND PASSWORDS

At this stage of the system building process there are only two passwords defined: the system and library passwords. To open the system up for TSS/8 users, more passwords must be defined. Each such password is actually two elements: an account number (1-4 octal digits) and a password (1-4 characters). Differant users may have the same password, but each must have a unique account number.

The password is the user's private identity and is used only for logging into the system. TSS/8 will not let a user access the system unless he types in a valid account number and password. Users should not divulge their password to other users. The account number is the user's public identity and is

generally not kept secret. The account number is important because it identifies a user's disk library. For each account number, TSS/8 maintains a library of saved files. Each such library is independent of all others. A user may only create files in his own library, i.e., the library associated with the account number and password used to log in. Once he has created them, he may decide whether other users may use these files, or change them.

The account number is actually two numbers, a project number and a programmer number. Account number 5440 is actually project number 54, programmer number 40. Account number 102 is project number 1, programmer number 2. For this reason, account numbers may be specified as two numbers separated by commas (i.e., 1,2) as well as a single number (102). Users may specify that all other users may share their files, only users whose project number is the same, or no other users at all*. Therefore, in defining new account numbers it is useful to group users into projects, giving them account numbers which have a common project number.

The library account number (2 or 0,2) is no different from any other account number. Users logged in as account number 2 may use TSS/8 just as any other user would. The one thing that makes it special is that the R command automatically fetches the specified program from the library of account number 2. In this way users may get programs from this library without knowing specifically its account number. The reason the library password is kept secret is to prevent users other than the system manager from altering its contents.

The system account number (1 or 0,1) is privileged. When logged in with this account number and, password, the user has access to several unique capabilities, such as defining new passwords. There are other capabilities available only to the system account number which are discussed later in this manual. It is, therefore, quite important that the system password remain secret.

LOGID is the program used to create new account number, password combinations. It is only usable by a user logged in with the system password. Therefore, the next step in the system building process is to log in with this password (the account number is 1).

LOGID is then called by typing:

```
.R LOGID
```

LOGID prints opening instructions, then an asterisk, and waits for you to specify the user's account number and password separated by a space. As usual, terminate the line with the RETURN key. After entering the combination, LOGID prints an asterisk and waits for another user account number-password combination.

*Chapter 10 of Introduction to Programming 1970 describes this capability in detail.

If, when entering a combination, you realize that you have made an error (typo or whatever) you need only type the RUBOUT key and that line will be ignored. LOGID will print a question mark and another asterisk so that you may enter the correct combination.

When all desired account numbers and passwords have been defined, you should type CTRL/B and then S. To complete this step, type LOGOUT. For example:

```
.R LOGID

TSS/8 ACCOUNT MAINTENANCE --

* ACC'T # <SPACE> PASSWORD <RETURN TO OPEN/CHANGE, ALT MODE TO CLOSE>

* 732 TOUR
* 1215 JOHN
* 1066 HARO
* 1000 OTTO
* ↑BS
.LOGOUT
```

2.6 DUMPING THE SYSTEM TO DECtape

NOTE

For simplicity, these instructions assume a system with one disk and at least two DECtapes. For other system configurations, see the general instructions in Section 2.7.

To dump the newly completed system onto DECtape, restart INIT at location 4200 of field 0 as illustrated in Figure 2-5. When INIT is started it prints:

```
LOAD, DUMP, REFRESH, START?
```

You should now mount DECtapes on units 1 and 2. Then set units 1 and 2 to WRITE ENABLE (see Introduction To Programming for complete instructions). Then type DUMP. INIT will copy an image of the entire system onto the DECtapes.

When INIT again prints:

```
LOAD, DUMP, REFRESH, START?
```

the entire system has been copied. Remove the DECtapes from the spindles and write some identification on the DECtape spools before filing them. To make the system available for use again, respond by typing START and complete the system startup procedure.

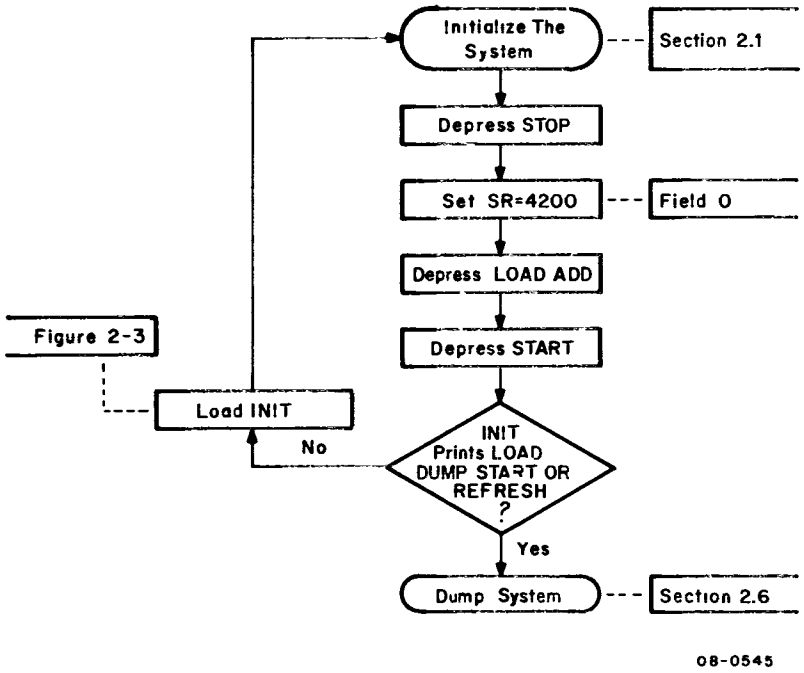


Figure 2-5 Calling and Starting INIT

2.7 GENERAL INSTRUCTIONS FOR DUMPING DISKS TO DECTAPE

The contents of an RS08 disk (256K words) will not quite fit on a single DECtape (200K words). Part of a second tape is required. In general:

Disks	DECtapes
1	2
2	3
3	5
4	6

Thus, for a one-disk system, the LOAD and DUMP process requires two tapes. Loading and dumping always proceeds as follows: The DECtape selected as unit one (1) is used first, then DECtape 2, then, if necessary, units 3, 4, 5, and 6. If the system includes as many DECtape drives as are indicated in the table above, setting up for a LOAD or DUMP is very simple. Select consecutive units, starting with unit 1 and mount the appropriate DECtapes. The LOAD or DUMP routine will access them in order.

If there are not as many tape units as there are DECtapes to be loaded or dumped, it is necessary to use them more than once. The LOAD and DUMP routines work as follows: they use DECtape 1, then look for DECtape 2. If they find it available (i.e., a DECtape unit has been selected as unit 2) the transfer continues on this unit. Then, if a third DECtape is needed, the routines look for unit 3. If at any point a unit is sought but not found, the routines wait for it to be selected. Therefore, it is

possible to load the first tape of the system on unit one, dismount the tape, place the second tape on the same DECtape unit, switch it to unit two, and have the load continue automatically at that point. The following procedure will dump the contents of two disks on a system with two DECtape drives. (Assume that the system has just typed out LOAD DUMP START OR REFRESH?) First set the DECtapes to units 1 and 2 and write enable. Mount two scratch tapes on these units labeled TAPE ONE and TAPE TWO. Now type DUMP. The system will completely write DECtape 1, then automatically go on to DECtape 2. At this point, switch DECtape 1 to LOCAL and rewind it. Now mount a third DECtape on this unit, labeled TAPE THREE, set the unit select to three, and then, as the last step, switch the unit to REMOTE. Normally, this procedure can be done in the time it takes for the system to write DECtape 2. It will then go on immediately to write DECtape 3. However, there is no need to hurry. If unit 3 is not ready when it is needed, the system will wait for it. The same procedure is followed for a LOAD.

This same general procedure is followed for any system where there are not enough DECtapes to select them all simultaneously.

Chapter 3
Operating the TSS/8 System

TSS/8 will function with a minimum of operator control. In a typical installation, the operation might consist of a morning startup, a system dump at night, and occasional type-outs of accounting information. In addition, you may need to clean out old disk files occasionally to prevent the disk file storage from being exhausted. If remote users wish to make use of the assignable devices (Section 3.7), an operator will be needed to mount DECtapes and paper tapes. Finally, there will be the normal system updating - adding and deleting account numbers and perhaps adding new programs to the system library.

NOTE

For simplicity, these instructions assume a system with one disk and at least two DECtapes. For other system configurations, see the general instructions in Section 2.7.

3.1 LOADING THE SYSTEM

For an installation without DECtape, loading the system is the same as that of building, described in Chapter 2. For an installation with the system copies on DECtape, loading is a simple matter of calling INIT and reading in the system from DECtapes. INIT is the one paper tape which is used continuously even after the system has been built.

Using the Binary Loader (Figure 2-3), load INIT into core field zero (see Figure 2-4) and start at location 4200. When started, INIT prints

```
LOAD, DUMP, REFRESH, START?
```

You should now mount on units 1 and 2 the DECtapes on which the system was copied, and then type LOAD. The tapes will spin as the system is being loaded into the configuration. (If the system includes multiple disks, there will be more than two tapes.) When the system is completely loaded, INIT will again print.

```
LOAD, DUMP, REFRESH, START?
```

Loading the system is just that simple once it has been dumped onto DECtapes.

3.2 STARTING SYSTEM

If the system is to be started after loading it from DECtape, reply to INIT's option query

```
LOAD, DUMP, REFRESH, START?
```

by typing START and replying to the startup queries as explained in Section 2.3.

When INIT is not in core, INIT must be loaded from paper tape using the Binary Loader (Figure 2-4) and start at location 4200.

In either case, you should specify the START option and reply to the startup queries as you did in Section 2.3. When the HR:MIN? query has been answered, type the RETURN key to transfer control to Monitor.

3.3 RESTARTING THE SYSTEM

The TSS/8 INIT program is used to perform all system starts and restarts. It always begins its execution by typing:

```
LOAD, DUMP, REFRESH, START?
```

During normal system operation, INIT is stored on the disk. To bring it into core, stop the system and start at location 4200 of field zero. This executes a special load routine in field zero resident Monitor which reads INIT in from the disk and starts it running. If TSS/8 (and hence the load routine) is not in core, it is necessary to load the paper tape of INIT by means of the BIN Loader. (See Sections 2.1.1 and 2.1.2 for details.) Once INIT is in core, start it at location 4200.

3.4 SYSTEM BACKUP

Since TSS/8 on-line files change from day to day, you should maintain system backups on DECtapes. To save the state of the system, first check, using SYSTAT, that all users are logged out. It is imperative that all users be logged out when the system is stopped for dumping. Depress STOP and start at location 4200 in field zero.

When INIT prints its option query

```
LOAD, DUMP, REFRESH, START?
```

mount two scratch (empty) DECtapes, one on unit 1 and the other on unit 2, and set for WRITE ENABLE. Now type DUMP, and the entire system, including all user files, will be loaded onto the DECtapes. You can restore the system to this state at any time by using INIT's LOAD option.

NOTE

For simplicity, these instructions assume a system with one disk and at least two DECtapes. For other configurations, see the general instructions in Section 2.7.

You may decide to perform a dump each night and to restore the system the next morning so that should a fatal malfunction occur, you can restore the system.

3.5 PASSWORDS AND ACCOUNTING

When the system was first built, a number of passwords were defined. As the system is used, you will need to add and change passwords and obtain records of system usage. These functions are possible only when you log in using the system password.

LOGID is used to update the file of valid passwords. To use it, type

```
•R LOGID
```

LOGID prints an asterisk to indicate that it is ready to accept input. Type the account number (1 to 4 octal digits), a single space, and the password (1 to 4 alphanumerics). To define the password (open the account) close the line by typing the RETURN key. To delete the password (close out the account) type the account number and password as above, but terminate the line with the ALT MODE key (ESC key on some Teletypes). The account number and password will be deleted, and any files in that user's library will be deleted. LOGID will print DELETED and another asterisk.

It is also possible to change the password for a given account number. To do so, type in the account number and password and close with the RETURN key. When LOGID requests it, type in the new password.

```
•R LOGID
TSS/8 ACCOUNT MAINTENANCE --
* ACC'T # <SPACE> PASSWORD <RETURN TO OPEN/CHANGE, ALT MODE TO CLOSE>
* 1215 JOHN $ DELETED
* 1066 HARO
   CHANGE PASSWORD TO:  WILL
* 1517 LUTH
* ↑BS
•LOGOUT
```

The System Library Program, CAT, is used to obtain system accounting information. CAT is a dual-purpose program: when called by a regular TSS/8 user, it prints the contents of his library. When used by the system manager, logged in under the system password, it prints a report of the accounting information for each user. This report consists of the accumulated time (in hours, minutes, and seconds) for central processor usage and connect time as well as the number of disk segments currently being used. This accounting information is continually being updated by TSS/8 Monitor.

System accounts may be obtained at any time. However, eventually the system manager will wish to record the accumulated usage and reset the accounting clocks to zero. Most systems will do this once a day. (TSS/8 can accumulate about a week of heavy usage before the clocks overflow.) At the end of each accounting, CAT asks if the manager wishes to reset the clocks. If not, respond NO or type the RETURN key. To reset the clocks, type YES. The accounting clocks should not be reset while other users are on the system.

NOTE

Logging in with the system password gives the user complete control over the system files and directories, including the ability to change (or destroy) them. Therefore, the password must be used with caution. CAT and LOGID are the only programs which should be used while logged in this way. Running other programs, such as BASIC, may alter critical files. Be sure to log out after running CAT or LOGID to prevent this possibility.

3.6 MAINTAINING THE SYSTEM LIBRARY

Generally, once the system library has been built there is no need to alter it. If you wish to add programs to the library, log in using the library account number and password and then use PIP, or COPY to load the programs into the library. To load new versions of DEC-supplied system programs, use PIP or COPY just as when building the library (see Section 2.4.2).

DEC-supplied system programs are TSS/8 binary programs stored in SAVE format. The system library is not restricted to just this one format. A user's BASIC, FOCAL, or FORTRAN program or other file of interest to many users may and should be made available from the system library.

3.7 ASSIGNABLE DEVICES

The assignable devices (reader, punch, or DECtape) are among the most powerful features of TSS/8. For consoles physically near the configuration, there is no problem in accessing these devices; each user handles his own paper tapes or DECtapes. If the user is remote from the configuration, an on-site operator must be present to handle the user's tapes or DECtapes. The on-site system operator should maintain a library of DECtapes for all users. When a user wishes to have one of his tapes mounted, he communicates with the system operator via the TALK command. For example, he might type

```
.TALK OPR MOUNT DECTAPE #1002 ON UNIT 3
```

This message, together with the number of the console on which the message was typed, is printed on the system operator's master console. The operator may then use the same TALK command to acknowledge the request and to indicate that the tape has been mounted. The TALK command may be used to call any console. A remote user may request any local user to help him mount and dismount DECtapes or paper tapes. See Chapter 10 of Introduction to Programming 1970 for more information.

3.8 CONTROLLING DISK USAGE

It will always be necessary to monitor disk usage to prevent the disk from filling up. The system accounting routine (CAT) provides a convenient method of checking on disk usage. For each account number/password, CAT lists the number of disk segments. If a single user has a disproportionate amount of disk, the system manager may then list his directory to see what has been stored. To do this, type R CAT:L instead of just R CAT. CAT will ask for the account number of the user whose directory is to be listed. Only the system manager may list another user's directory.

```
.R CAT:L

USER NUMBER- 5440

DISK FILES FOR USER 54,40 ON 1-JUN-70

NAME        SIZE PROT    DATE
BTYPE .BIN     1   12   1-JUN-70
PEACE .BAS    14   12   1-JUN-70

TOTAL DISK SEGMENTS:    15


USER NUMBER-  ↑BS
.LOGOUT
```

If it becomes necessary to delete some files for a user, the system manager may log in with that user's password and delete them. The LOGID program also provides a convenient way of deleting all of a user's files without having to log in under his password. To do so, delete the password, thereby deleting all his files. When this has been done, redefine the password.

The actual means of controlling disk storage will depend on the particular installation. Some installations may keep DECtape copies of deleted files; others will simply delete them. Some installations will have regularly scheduled times when the disk will be cleared.

3.9 CONTROLLING SYSTEM USERS

The system manager has complete control over on-line users. If necessary, he may interrupt or even log out a user. For example, a user may forget (or refuse) to release a device which is needed by another user. The system manager may force its release with the FORCE command;

FORCE keyboard number command

The FORCE command allows the system manager (logged in under the system password) to connect to any other Teletype long enough to issue a command. For example, if the user sitting at keyboard #10 has the reader and will not release it, the system manager may type

```
.FORCE 10 RELEASE R
```

RELEASE R will be printed on the user's console (just as though the user had typed it) and the reader will be released.

The FORCE command works exactly like typing on the affected console. Commands entered by FORCE are treated as Monitor commands only if that console is in Monitor mode. The user at his own console makes use of the CTRL/B (echoed ↑B) to put the console in Monitor mode. Within a FORCE command, ↑ (SHIFT/N) is used to mean CTRL/B. For example, the above command could by typed

```
.FORCE 10 ↑S;RELEASE R
```

The ↑S (SHIFT/N followed by S) acts just like ↑BS (CTRL/B followed by S) and assures that whatever the user at console 10 is doing is terminated, allowing the RELEASE command to be executed. In general, when forcing a Monitor command, precede it by ↑S; (SHIFT/N followed by S and semicolon) as shown above.

Only the actual command (FORCE) and the keyboard number need to be properly formatted for the command to be considered valid. If they are not correct, Monitor will return an error message to the console which issued the command. For example,

```
.FORCE 9 ↑S;RELEASE R
FORCE 9 ↑S;RELEASE R?
```

Monitor requires that keyboard numbers be octal numbers. If the rest of the command, i.e., the part to be forced, is invalid, the command is executed anyway. No error message is typed back. However, when Monitor attempts to execute the command for that console, it detects the error and prints an error on the console being forced, not the console which issued the FORCE command. For example,

```
.FORCE 10 ↑S;RLEASE R
.
```

The job connected with keyboard 10 is stopped, but the reader is not released. The following message is printed on console 10:

```
↑BS;RLEASE R
.
RLEASE R?
.
```

FORCE may be used for input to system programs as well as commands to Monitor. For example, the following sequence of commands would log in a user at console 7, call BASIC, load a BASIC program ABC from the System Library, execute it, then log out.

```
.FORCE 7 ↑LOGIN 57 XXX
.FORCE 7 ↑R BASIC
.FORCE 7 OLD
.FORCE 7 ABC*
.FORCE 7 RUN
.FORCE 7 BYE
.FORCE 7 ↑LOGOUT
.
```

FORCE may be used only by a user logged in with the system password.

3.10 COMMUNICATING WITH USERS

Normally, the system manager communicates with users via the normal TALK command. However, there are times when it is necessary to send a message to all users. The BROADCAST command is used for this purpose. BROADCAST may only be used when logged in with the system password:

.BROADCAST MESSAGE

The message indicated is sent to all Teletypes unconditionally. Active Teletypes are interrupted. For example,

```
.BROADCAST THE SYSTEM WILL BE SHUT DOWN IN 5 MINUTES
```

will cause the following message to be printed on all consoles:

```
*** THE SYSTEM WILL BE SHUT DOWN IN 5 MINUTES
```

3.11 SPECIAL IOTS

Chapter 10 of Introduction to Programming 1970 documents the IOTs which are available to programs running under TSS/8. In addition, there are two special purpose IOTs. Since they are not useful to ordinary users they are documented here.

PEEK Octal Code: 6423

PEEK allows a user program to examine Monitor core fields zero and one. It is used by the SYSTAT program to determine the status of individual users. PEEK is also used by LOGOUT.

To use PEEK, load the accumulator with a pointer to a four-word block where:

Word 1 = Monitor field in bits 6-8
2 = Starting Monitor address
3 = Starting user address
4 = Two's complement of number of words to transfer.

LOGOUT Octal Code: 6615

LOGOUT logs a user off of TSS/8. All assigned devices are released and the user's Teletype becomes inactive. LOGOUT is used as the last instruction of the System Library Program LOGOUT. To use LOGOUT, load the AC with the job number of this job.

There is an alternate use of LOGOUT. Load the AC with 0. LOGOUT then returns the number of jobs logged in with the same account number as the job executing the LOGOUT IOT. The job is not logged off the system.

Chapter 4
Modifying TSS/8

The information in this chapter is not necessary to operate TSS/8. Most system managers will use the TSS/8 software exactly as it is supplied. Other users, however, will want to make minor modifications or, in some instances, major system changes. This chapter describes the tools available for making such changes.

4.1 MODIFYING SYSTEM LIBRARY PROGRAMS

Modifying library programs is an on-line process. Users who are familiar with TSS/8's advanced Monitor commands will find it a straight forward procedure. Log in with the library password, load the program into core, deposit the patches, then save the program again. For example, a user may wish to modify EDIT so that it considers every sixth character position to be a tab stop. The process is as follows for the 1970 version of TSS/8 EDIT:

```
.LOAD EDIT
.DEPOSIT 2 7772
.SAVE EDIT
.LOGOUT
```

EDIT will now be changed on the disk. If the system includes DECtape, dump the whole system so that the changed version will be captured on the backup tape. If the system does not include DECtape, but does have a high-speed punch, a new SAVE format paper tape should be punched with PIP. Otherwise, the change must be made every time the system is built.

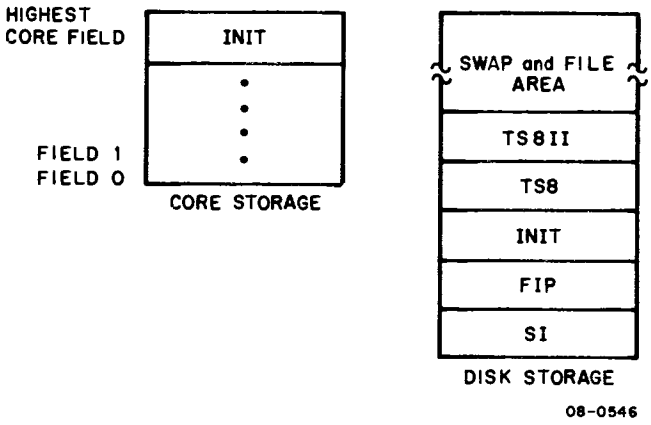
Other System Library Programs may be modified in a similar manner.

4.2 MODIFYING TSS/8 MONITOR

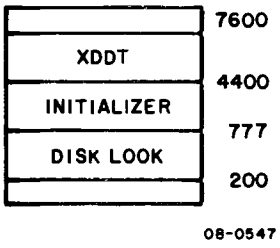
A formal procedure exists for making patches to Monitor. In order to understand how this procedure works, it is necessary to understand how TSS/8 is stored on the disk. The five pieces of Monitor (SI, FIP, INIT, TS8, TS8II) are kept on the first 20K of the disk. Their respective disk addresses are

SI	00000-07777
FIP	10000-17777
INIT	20000-27777
TS8	30000-37777
TS8II	40000-47777

Although the third section is referred to as INIT, it is actually made up of three programs: the TSS/8 initializer, a debugging routine (XDDT), and a disk patch routine (DISKLOOK). To patch the system, it is necessary to bring these routines into core. To do so, stop the system and start at 4200. INIT is brought in and prints LOAD, DUMP, REFRESH, START? At this point the layout of core and disk is as follows:



Within INIT, the three programs are positioned as follows:



Starting at 4200 always brings INIT (plus XDDT and DISKLOOK) into the highest core field in the system. Thus, it will come into different fields for different systems.

When INIT comes in, it prints LOAD, DUMP, REFRESH, START? To start the patching procedure, type XDDT. This will start the XDDT program. XDDT is a powerful debugging tool which is available through DECUS (order DECUS 8-127). It allows the user to control execution of his program while debugging it. (A complete description of XDDT is available from DECUS.) Before doing anything, XDDT needs to know what core field it is in. To provide this information, type the core field number (the highest one on the system, followed by a number sign (#)). Thus, for a 16K system, type:

```
3#
```

At this time the data field lights should equal the instruction field lights. Now use XDDT to transfer control to DISKLOOK by typing:

```
200'
```

DISKLOOK is now running, allowing the user to examine and modify single disk registers. To examine a register, type its address (in octal) followed by a colon. DISKLOOK will print the present content of that register on the disk and wait for a new value to be typed. Enter the new value by typing 1 to 4 octal digits. Type the RETURN key to close the line. If a register has been opened but need not be changed, type the RETURN key. To automatically open the next sequential register, type the LINE FEED key instead of RETURN. Remember that disk locations are actually 5-digit addresses. For example, location 2104 in TS8 is stored in disk location 32104. Location 10 in FIP is 10010, etc.

When all desired patches have been made, type CTRL/X to return to XDDT. To restart the system from XDDT, type

```
4200'
```

An example of the usage of DISKLOOK:

```
LOAD, DUMP, REFRESH, START? XDDT
3#
200'
42601: 7000 5254
6100: 6636 1220
```

```
21241: 0300 320
21250: 0310 330

4200'                                          CTRL/X typed by user
LOAD, DUMP, REFRESH, START?
```

Location 2601 in TS8II is changed from a NOP to a JMP. This change allows the system manager to examine selected Monitor registers by entering an address in the switches. If this patch is made, user programs may not use EAE instructions. The pointer in location 4465 of SI is changed to point to an error return. This patch disables the TALK command. Finally, locations 1241 and 1250 of INIT are changed. This patch changes the device code of a PT08 from 30, 31 to 32, 33. (Note, the exact locations may differ in future Monitors. These examples are for illustration only.)

All changes to Monitor are made on the disk. In this way they will become effective at the next startup. Starting the system brings TS8 and TS8II into core from the disk. SI and FIP are swapped in by the system as needed. The exception is INIT. Since it is already in core when the patching is done, the core image, rather than the disk image will be used to start the system. Therefore, patches made to INIT will not take effect immediately. To get the patched INIT, it is necessary to start the system, then stop it and start at 4200 of field zero, thus booting in the disk copy of INIT.

Once patched, the system should, of course, be dumped to DECtape to preserve the patches. Systems without DECtape must be repatched every time they are built.

4.3 CONTROLLING MONITOR EXECUTION

The XDDT program is very useful for testing any modifications to Monitor. (Information on XDDT is available from DECUS, order number 8-127) XDDT is always in core with INIT. Thus, it is possible to use that XDDT. When the system is started up, specify one fewer user fields to protect XDDT. If the core field is not available, you may request EXEC DDT. EXEC DDT is XDDT loaded into Monitor's buffer area in field one. This restricts Monitor's capacity to three to four simultaneous users, but otherwise does not affect it.

In either case, the starting address is 7000 in the appropriate field. Once Monitor is running, you may stop it and start XDDT only while Monitor is running the null job. You may then restart Monitor at location 4201. If you stop Monitor outside of the null job, press CONTINUE and try again. Never stop the system if a device is active.

Part Two

TSS/8 Monitor

Chapter 5
Introduction and Basic Execution of User Programs

5.1 INTRODUCTION

Although it is not necessary to successfully operate TSS/8, knowing how TSS/8 Monitor words can be helpful to managers. This section provides a brief overview of the major system components and how they interact. It is assumed that the user is familiar with assembly language programming in general and PAL-III in particular. Also the reader should have a detailed knowledge of how to use TSS/8 at the assembly language level and be familiar with software systems.

Some users will want to modify Monitor. For this reason a listing of Monitor is provided to each installation. Careful study of this listing should precede any attempts to change the code. (Comments in the listings will aid this study.) For users who are considering system modifications, this section will serve as an introduction to the listings.

5.2 AN OUTLINE OF THE SYSTEM

The basic design goal for TSS/8 was to provide each of 16 (or more) simultaneous users with the capability of a 4K PDP-8. TSS/8 users should be able to write programs for the system which look just like those written for a stand-alone machine. Existing programs should be able to be run without significant modification, the differences being limited to what is necessary to make the program run efficiently in a time-sharing environment. These programs should be executed directly by the processor rather than being interpreted. In addition, TSS/8 would give the user convenient access to a disk. Disk storage would be available to all users on a first-come-first-served basis. Finally, the system should make other I/O devices available to the user on an exclusive basis. Thus, a DECtape could be assigned to a user for as long as he needs it, then be released for use by other users.

The TSS/8 system which does all this based on a PDP-8 or PDP-8/I with extended memory. The first 8K of core holds a resident Monitor program used to control the system. Additional fields (from 1-6) are used to hold user programs, one user to each 4K field. To allow Monitor to control the execution of the user programs, a processor modification, the KT8/I, is added to all TSS/8s. This modification

defines a new one-bit state register, the User Mode bit. When this bit is cleared, the modification is disabled and the system is said to be in Exec mode. (The bit is cleared by the load address key; therefore, its presence does not affect other programs which may be run on the system on a stand-alone basis.) As long as the bit is disabled, the processor operates exactly like any other PDP-8. When the user mode bit is set (by an IOT), the time-sharing hardware is enabled. This hardware inspects each instruction as it is fetched from core for execution. If it is an IOT instruction (6XXX) or a HLT (7402) or an OSR (7404), execution is inhibited. Instead, a flag, the User IOT flag, is set. This flag is connected to the interrupt bus. Thus, when the processor is in User mode, any attempt to execute these privileged instructions is prohibited, and control goes, via the interrupt, to field zero where Monitor is resident. All other instructions execute normally. See Appendix E for more detail.

Monitor uses this hardware modification to control the running of user programs. It always runs in Exec mode. However, when it starts up a user program which resides in one of the upper core fields, it first sets the Data and Instruction fields to that field, then puts the processor in User mode. Only then does it jump to the user program. At this point, it is guaranteed that execution of the user program will not affect any other core field. The only way to do that would be to execute an IOT, which traps back to Monitor. TSS/8 systems also include a real-time clock to assure that control will eventually return to Monitor even if the user program does not execute an IOT.

Although all IOT's are trapped, they are by no means illegal. In fact, one of Monitor's most important functions is to execute the valid user IOT's. It does this by simulating them in the software, then returning control to the program (which never knows that the IOT wasn't executed by the hardware). Invalid IOT's are ignored.

TSS/8 is designed to accommodate more users than will fit into core simultaneously. Therefore, it requires an external storage device for those users not in core. The high-speed RS08 disk is used for this purpose. Each on-line user has a dedicated 4K area on the disk where his program is stored when it is not in core. Programs are swapped in and out of core as needed for execution. The disk also contains three nonresident portions of Monitor. These are infrequently used routines which are brought into core only when needed. The remainder of the disk is made available to the users for on-line storage.

All TSS/8's must also have interfacing for multiple Teletypes. Depending on the number of terminals in the system, one of a variety of interfaces may be used. Typically, PT08's are used for very small systems, DC08's for all others.

These are the primary elements of the system: 1) a modified PDP-8 or 8/I with extended memory, 2) a disk and controller, and 3) interfacing for multiple Teletypes. In addition systems may have high-speed, paper-tape equipment and DECtape.

5.3 SHARING TIME

The most fundamental job of a time-sharing monitor is the sequential execution (generally for short bursts, or quanta, of time) of a number of user programs. This implies that Monitor has available to it a place where a user program can be brought in order to execute it, and a place to put user programs when they are not being run. TSS/8 reserves one or more core fields within the PDP-8/I as areas in which to execute user programs. A user program, and hence a user area, is 4K words long. TSS/8 may have from 1 to 6 user areas, depending on the amount of core available. Similarly, TSS/8 reserves a portion of its disk as a place in which to keep programs which are not being executed. These "swap areas" are also 4K each. The number of user cores is not necessarily dependent on the number of simultaneous users. Monitor simply uses as many as it has available. The number of swap areas, on the other hand, is directly related to the number of simultaneous users for which the system is configured. There is one dedicated 4K swap area on the disk for each simultaneous user.

User programs are executed by TSS/8 by bringing them into a user core from their swap area, executing them, then returning them to their swap area on the disk, so that the next user program may be brought in. User programs may be brought into any available user core, but when they are swapped out, they always return to their assigned swap area on the disk.

TSS/8's swapping algorithm may be best illustrated by assuming a very simplified situation. TSS/8 has a number of user programs running within it; each is doing hard compute, none is engaged in any input/output. Monitor first decides which user to run next. It chooses the user who has waited to run the longest. It schedules this user to be brought into a user core. However, it can only bring this user into a user core which is unoccupied. Therefore, it must empty one by swapping its present inhabitant (another user program) out. Before doing this, Monitor saves the running state of the program to be swapped out. This information, the AC, PC, LK, MQ, and SC is stored in Monitor core. It then writes the user program (whose state is saved) out onto its respective swap area. Now the user program selected to run next may be brought into core. Once it is in, its run state is restored (the AC, PC, LK, MQ, and SC of each user is stored in Monitor when they are stopped running) and the program is started up. This procedure is continued as long as the user programs need to run.

Obviously, Monitor has to maintain status information about each user program, whether it is in core or not. Indeed, it must maintain more information than just a user program's run state. It must maintain all the information it needs in order to decide if a user program needs to be run or not. In actual

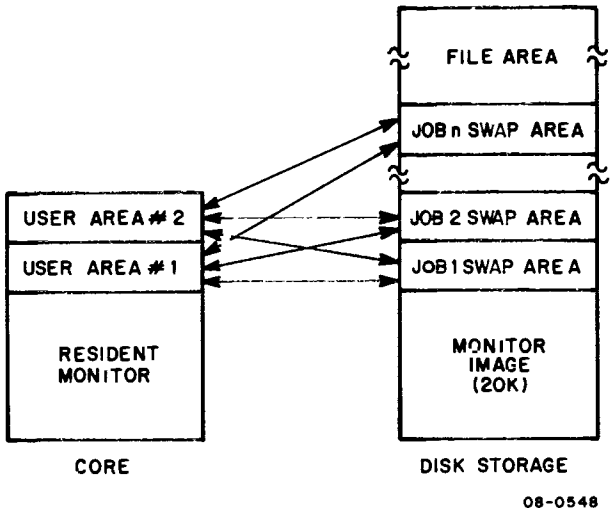


Figure 5-1 16K TSS/8 Configured for 16 Users.

operation, most of this status information deals with the state of a user program's input and output. In our simplified case, where no user is doing any I/O, the only information that needs to be maintained is whether he is done or not. If a user program completes its run, Monitor remembers this fact. The program is swapped out and stays out. If a program does not complete its run at the time when it must be swapped out to allow another user to run, it is remembered to be still runnable. When its turn comes up again, this user will be swapped in to run some more.

The process of deciding which user to run next (scheduling) is an important function of the Monitor. The scheduling algorithm of TSS/8 Monitor is "round-robin". Monitor cyclically scans a table which contains the status information for each user. If the user being checked by Monitor does not have to be run (is not runnable) he is skipped and Monitor goes on to look at the next user. When it finds a user who needs to be run, it goes through the process of swapping out a user who has just been run but who is still in core in order to free up a core field. It then brings in the user who was selected to be run, starts him up and allows him to run for a fixed time quanta. At the end of this time quanta (as indicated by a clock interrupt) Monitor goes on to the next user to see if he is runnable. When it has looked at all the users, the Monitor schedular returns to look at the first job again. It then continues to cycle through the table of users.

In a system with a single user field, the scheduling algorithm is just that simple. The previous user must always be swapped out to make room for the next. Once a user is brought in and started up, there can be no further scheduling activity until he has completed his quanta, or time slice. Similarly, once the user in core has started to be swapped out, the system must wait until the next user is completely swapped in before it can do anything. (A user program may only be run when it is completely in core.) The only special scheduling case for a one-core system comes when only one user program is active in the system. User programs are not automatically swapped out when they complete a time slice. They are only swapped out when another user program must be brought into core to be run. On the other hand, when the schedular decides that a given user should be run, it does not blindly swap it in. It first checks to see if it is already in core. Thus, if only one user program is running, no swapping takes place. When the program has been run for a quanta, its run state is saved but it is not swapped out. The schedular scans through the table of user programs looking for one to run. Since no other program needs to be run, it gets right back to the program just run as the proper one to run next. Finding this program still in core, schedular simply restores its state and restarts it. Thus, except for these periodic checks, the lone user job runs continuously.

The scheduling gets more complicated, and more efficient, when there is more than one user core available. The schedular maintains, besides its table of all user jobs, a table of all user jobs which are in user cores. (A job may be in core, on the disk, or halfway in between when it is being swapped.) It actually scans the former table to decide who to swap next and the latter table to determine what to do in the meantime (while it is waiting for the swap to be complete). The swapping, once set up, happens asynchronously with respect to the scheduling. Once it has set up the swap, Monitor always goes to its table of in-core programs looking for one to work on. When a user program is scheduled to be swapped out, it disappears from the list of in-core programs. Eventually, the program scheduled to be swapped in next is brought in. It then appears in the table of in-core programs and is subsequently run.

In the case of a system with two user fields (16K system) the table of in-core programs has two entries. Entry one indicates which, if any, user program is in field 2; entry two indicates which is in field 3. In actual operation, there will seldom be user programs in both core fields at once. In a two-user-field system (again assuming our case of several running, compute-bound user programs) one field will always be swapping while a program is running in the other. This is because the quantum of time that a user program is allowed to run is (roughly) equal to the time it takes to do a swap (a write followed by a read). It works out as follows.

A user who has just been run is scheduled to be swapped out. In the table of in-core programs, he is marked as no longer in core. The schedular then goes to see if there is anyone in core to be run. The

only candidate is the other user core. If the timing is right, a user program will just have finished being swapped in. Schedular then sets up and runs it. (Note that if this swap is not completed until after the second swap was started, Monitor must wait for it to come in. This situation would occur if a transient error delayed the swap. On the other hand, if latencies on the disk were minimal, the swap might be completed before the other program completed its run quanta. In general, however, these two events will be almost simultaneous.) At this point, a user program has been started at about the same time another one is to be swapped out. At the end of its run quanta, the swap should be complete and a new program in and ready to run.

Thus, at any given time, one of the user cores is being swapped while a user program is being run in the other. The data break capability of the PDP-8/I allows these two operations to occur simultaneously. Cycles are stolen from the running program to allow transfers to occur in the other field. There is (in theory) no time lapse between the running of user programs. The next one is always ready at the time the user program being run finishes its time slice. Using the standard time slice of 200 milliseconds, this allows five users a second to be run. This situation is in strong contrast to the situation with a single-user core. Again assuming a 200-millisecond time slice, only half as many users may be run in the same time. This is because the system cannot run one user while swapping another. During the 200 millisecond swap time, the system must simply wait for the swap to be completed. In the one-user core system, swaps and runs alternate; in a two-user core system they are simultaneous. It is a foreground-background operation.

Schedular depends on various interrupts to keep this process going. Specifically, the scheduling is driven by the clock and disk completion interrupts. After every successful swap and after every third clock interrupt (i.e., 50 milliseconds) the schedular is run. If schedular is run because of a clock interrupt, it checks to see if this is the fourth such clock interrupt it has seen since it started to run the user it is presently running. If not, then this user has not been given his full quantum of run time. He is therefore restarted. When the fourth schedular clock call occurs, indicating that the user program has run for a full 200 milliseconds, it is marked as having been run. Schedular then looks through its table of in-core user programs until it finds one to run. If no other programs are in core, it sees if a swap is in progress. If one is, schedular knows that eventually a new user program will come into core. It goes back and runs the same program. Eventually, the program being swapped will come in and be run. Even if there is another program in core, schedular checks to see if a swap is in progress. If one is, it simply starts up the next resident user program and runs it.

Whenever schedular finds that no swapping is going on, it checks to see if a swap is necessary. A swap is necessary if a user program which needs to be run is out on the disk. Thus, when schedular finds that no swapping is going on, it checks its table of user programs to see if it can find a runnable,

swapped-out user program. If it finds one, it schedules this program to be swapped in. (Generally, this means swapping someone else out.) Once it has set up the swap (if one is called for) schedular finds the next resident user program and starts it up. (Note: the check for swapping activity actually occurs every 50 milliseconds to make sure that the swapping rate is kept up.)

A swap is scheduled by putting a swap request in the disk queue. If the disk is active at the time the swap is scheduled, it is not initiated immediately. However, if the disk is inactive, the transfer is initiated (by setting up and executing a DMAR or DMAW) immediately. Either way, the user program to be swapped is removed from the table of in-core users. It is considered to be no longer in core at the time it is scheduled to be swapped, even though it may not actually be written out until some time later.

Every time the disk completion interrupt occurs, the schedular is run to see if there are any swap requests pending in the disk queue. If there are, the next one is started immediately. If the disk transfer just completed was a swap in, meaning a new user program is now in core, the table of in-core programs is updated to reflect the new arrival.

Thus, the scheduling process consists of two asynchronous processes. Disk handlers, running off the interrupt, are continually swapping users in and out of core areas. As they do this, they update a table which indicates which user programs are in user cores. These routines work on a queue of disk requests. As soon as a transfer is complete, as indicated by a disk completion flag, the disk routines immediately start the next transfer on the queue. While the disk handlers are processing the requests on the disk queue, other schedular routines are deciding what swaps, if any, to do next. Once they have made that decision, and queued the appropriate disk request, they scan the table of in-core user programs, in order to select the next user program to be run. This table is updated by the disk-swap-handling part of the schedular. Thus, a user program which schedular selects to be swapped in will eventually be swapped into a core and hence appear in the table of in-core user programs. Schedular, scanning this table looking for a resident job to run, will find it and run it.

It is important to system efficiency that, at the time schedular goes to the table of in-core users to look for someone to run, that it find someone. If it doesn't, schedular schedules a nonjob, the "null job" to be run. This null job is run until a valid user program comes into core. (The null job is a tight loop in Monitor core which increments the accumulator. It does not take up a user core; it is not swapped.) Clearly, in a one-user core configuration, the system spends a great deal of time in the null job. From the time a swap is initiated until it is completed, Monitor can do nothing but run null job. In a two-user core system, the efficiency is much greater. The background swapping assures that a new user program will be in core about the time the currently active program finishes its time slice. More than two user cores virtually assure that time will not be wasted running the null job.

The previous discussion of scheduling is based on some radically simplified assumptions. We assumed a steady number of compute-only jobs. With a more normal mix of programs, scheduling becomes much more complex. User programs are being continually started and programs are being continually started and halted. Those that are running may need to be interrupted for input/output. All this increases the complexity of the scheduling. How these additional complexities are handled is discussed further on in this manual.

5.4 SOME DEFINITIONS

In the preceding discussion, we have referred to the programs running in TSS/8 as "user programs". In fact, in the system documentation, they are referred to as "jobs". Job in this sense means something slightly different than user program. It also means something different than "job" as it is used in batch processing systems such as the IBM 7094.

A job in TSS/8 is the capacity, or capability to run a program. A user, when he logs in, is assigned a job. He keeps this job, which has an associated job number, until he logs out. A 16-user system is thus a 16-job system. At startup, it has a pool of 16 available jobs which it assigns to individual users as they log in. Once it has assigned all its available jobs, Monitor cannot accept more users until one logs out and releases his job.

The distinction between a job and a user program is clearest right after logging in. The just-logged-in user has a job. He has been assigned a Teletype with which to intercommunicate and a 4K swap track in which to store his program. However, as yet he has no user program. In short, the job is not the program, it is the capability to run a program.

Once logged in, users are known to the system only by their job numbers. Monitor simply schedules and runs jobs. The job numbers for a 16-user system are 1 through 16. The null job is assigned job number zero. Users who are trying to log in are assigned job numbers (since a job number is required internally even to get through the login procedure). If the login is successful, the user retains his job number; if it is not, he is forgotten.

Monitor maintains a table, JOBTBL, which indicates the status of each job. It has a one-word entry for each possible job. If the job is unassigned, this word is zeroed. While the job is defined, its word in JOBTBL contains a pointer to the complete status information for this job. Monitor also maintains a table of in-core jobs. This table is called CORTBL. It is made up of a one-word entry for each available user core. Each entry contains the job number (and some other status bits) of the job which occupies that particular core. Finally, there is a single register, JOB, which indicates which job is being run at a given moment. JOB is updated at the end of a job time slice, CORTBL is updated with each swap; JOBTBL is updated on log-in and log-out.

5.5 TALKING TO THE USER PROGRAM

The preceding discussion was limited to compute-bound jobs, those that do no input or output. Obviously, this situation is rare. Most user jobs do a great deal of console I/O. For Monitor to process this console I/O, it must solve a number of immediate problems. First, it must be able to handle multiple consoles. This is largely a hardware problem, solved by PT08's or the DC08 hardware. All TSS/8 configurations have one or the other, thereby allowing the system to input characters from any given console and output them to any console. However, it must also keep track of which console is which, and which characters came from which console. User programs on TSS/8 are regular PDP-8 programs; as such they input characters from "the" console and output them to "the" console. In a stand-alone system, which has only one console, there is no ambiguity. In a TSS/8 system, where many jobs are each outputting to "the" console, the potential for confusion is considerable. The TSS/8 Monitor must maintain a table telling which console belongs to which job. Thus, when a job does console I/O, Monitor knows the individual console involved.

These are the immediate problems which Monitor must solve. However, in order to be useful, it must also be efficient. Normally, a PDP-8 program which is doing I/O spends virtually all its time waiting for the device. It is Monitor's responsibility to recover this time and use it to run another job. Finally, Monitor should smooth out the I/O. TSS/8 is a swapping system; user programs are in core only for short bursts, then they are swapped out to the disk. If a user program can only output when it is in core, it's typeout would come in choppy bursts. Input would be worse still. If a user could only input when his program happened to be in core, he would never get any input done.

This problem of smoothing out I/O is solved by maintaining buffers within Monitor. There is a Teletype input buffer and an output buffer for each job in Monitor. On input, as characters are received from the console, they are put in the console input buffer for the job which is associated with that console. Thus, the program which is to receive these characters need not be in core. The same is true of output. Characters are taken from a job's console output buffer and sent to his teleprinter whether the associated job is in core or not.

This console character handler may be thought of as the asynchronous part (asynchronous in the sense that it happens independently of the running of individual user programs). Each user's input and output buffers are being filled and emptied (by Monitor) whether the user's program is in core or not. It is essentially an overhead function. A little bit of processor time is stolen from whatever program is currently running and used to keep up the I/O for all active users.

This regularly scheduled (every 90 milliseconds) Teletype handler solves the problem of shuttling characters between console and buffers. There remains the problem of passing them between these buffers

and actual user programs. This character passing occurs via the TSS/8 hardware trapping capability. On input, the key instruction is KRB, the keyboard read IOT. A user program, when it wants to input a character, executes this KRB. The hardware modification causes a trap to Monitor, preventing hardware execution of the instruction. Monitor, on identifying the trapped IOT as a KRB, gets a character from the input buffer which corresponds to this job, puts it in the accumulator, and returns to the user program at the instruction following the KRB. The user program need never know that the KRB was simulated. It acts exactly as it does on a stand-alone PDP-8. The same procedure applies to output. Execution of a TLS is prevented by the hardware; a trap to Monitor occurs instead. Once it has identified the trapped IOT as a TLS, Monitor takes whatever was in the accumulator at the time of the trap and puts it in the appropriate output buffer. Once again, the IOT has been precisely simulated. (The asynchronous Teletype routine assure that characters placed in the output buffer are typed out eventually.)

However, the ability to simulate KRB and TLS is only half the battle. There remain all the timing and synchronization problems which are normally solved by the skip IOT's: KSF and TSF. In a stand-alone PDP-8, KSF means "Is there a character in the input buffer?" In TSS/8 the one-character hardware keyboard buffer is effectively replaced by a multicharacter software input buffer. Thus, in TSS/8, KSF means, "Are there any characters in the input buffer?" KSF, being an IOT, traps from a user program. Monitor, upon identifying the trapped IOT as a KSF, checks that user's input buffer. If there are any characters in it, Monitor simulates a skip by returning to the instruction in the user program which is two registers beyond the KSF. Similarly, on output, TSF asks, "Is the output buffer completely full?" If it is not, Monitor simulates the TSF by causing a skip in a user's program.

This arrangement allows for efficient user program I/O. It allows many characters to be passed between a user program and Monitor quickly. In a stand-alone system, it is impossible to input characters at more than 10 cps, the Teletype speed. Under TSS/8, many KRB's or TLS's may be executed in a hundred milliseconds. For example, consider the following typical sequence of code:

```
LOOP,   TAD I   AX      /AUTOINDEX
        TLS
        TSF
        JMP     .-1
        CLA
        JMP     LOOP
```

The first TLS puts a character in the output buffer but (assuming it is empty to start with), does not fill it. Therefore, the first execution of the TSF skips and causes an immediate loop back for another TLS. In this manner, a whole series of characters may be output in a few milliseconds. (By output, we mean moved to the output buffer. It may be many seconds before the asynchronous Teletype handlers get them all typed out. This, however, is of no concern to the user program.) Similarly, if there are many characters waiting in the input buffer, they could all be picked up at the same time by a KRB loop in a user program.

If the timing of these functions can be manipulated favorably, the system can handle input and output efficiently. The object is to separate the character I/O from the waiting. Rather than waiting 1/10 of a second between each character, put out 80 or 90 characters at once, then wait 8 or 9 seconds. By bundling the I/O wait times into usable amounts, like 8 or 9 seconds, Monitor can use them to run other jobs. This timing is handled partly by the schedular and partly by the routines which handle KSF's and TSF's. It is important that the user never hang in a KSF; JMP .-1 or TSF; JMP .-1 loop as he does on a stand-alone system. This is the code normally used to wait until more I/O can be done. On TSS/8, these waits are 8 to 9 seconds. The user job cannot be left to waste processor time in this loop. Therefore, when Monitor detects a KSF or TSF which is false (i.e., the program must wait for the device) it stops the program just as it would have if the time slice were up. The state of the program is saved. However, the program is stopped in a special way. It is marked as not runnable and the reason it is not runnable (it is waiting for input or output) is remembered. Since it is not runnable, it is dropped from the run queue and, when schedular comes to it the next time, it will not be run. However, Monitor continues to keep track of the state of the I/O. At the time when the device being waited for is again available (about 8 seconds later for the Teletype), Monitor changes the job's state back to being runnable. Now the next time the schedular looks at this job, it will run it. The job will be started up right where it left off (at the skip instructions) but this time the skip will be true, allowing the program to continue. Thus, by trapping the skip IOT, Monitor has salvaged the wait time from a job and used it to run other jobs.

In order to make the running of user programs even more efficient, Monitor exercises control over the keyboard and teleprinter flags. These flags are part of the status information for each user. The object is to turn the flag on, thus starting up the user program only when it is possible to process many characters. On input, this is done by setting up a "break mask". This break mask tells Monitor what characters are important delimiters. For example, BASIC-8 considers carriage return and rubout to be delimiters. When BASIC-8 is ready for keyboard input, it executes a KSF to see if there is any. Typically, there is not. (The user has not yet started to type in the next line in his program.) Therefore, this user is put in the I/O wait state and is marked as not runnable. He stays in this state until a delimiter appears. His keyboard flag is then set, he is returned to the runnable state, is scheduled, and

run. As soon as the program starts up, it executes KRB's to read input characters. BASIC-8 can thus process a whole line of input in a single, 200-millisecond time slice. Since this line probably took several seconds to input, this user is actually taking up very little of the system's time. The same situation applies to output. As the program outputs characters, these characters are placed in his output buffer. As long as it doesn't fill the buffer, the program is allowed to continue running. However, when the buffer does fill, this clears the program's teleprinter flag, thus suspending execution of the program (it moves into I/O wait). As characters from this buffer are subsequently typed out, ending the buffer-full condition, the teleprinter flag is kept down. It is held down and the user kept in the nonrunnable state until the buffer is almost emptied. At this point, the program is restarted so that it can put more characters into the output buffer, thus keeping up continuous output. Programs like BASIC-8 can fill an 80-character output buffer in 1 to 2 time slices. Therefore input, like output, is accomplished without substantial processor time.

Thus, it is the combination of the two parts of the I/O handlers; those which are driven by IOT traps and hence operated synchronously with respect to the user program, and those which are driven by clock interrupts (every 90 milliseconds) and hence asynchronously with respect to user programs which accomplish I/O. The common communications areas for these two routines are the console input-output buffers and their associated flags. The problem of efficient scheduling is solved by prudent manipulation of these flags. On input, this is done by means of the delimiter, or break mask. On output, it is done by detecting buffer-full and buffer-almost-empty conditions.

Chapter 6
Monitor: A More Detailed Look

So far, we have reviewed some of the operations of TSS/8 Monitor and how it responds to various simplified situations. This chapter discusses these operations in greater detail: the various subsystems within Monitor, the full scheduling algorithm, and the data base.

6.1 MONITOR AS INTERRUPT HANDLER

The fundamental task of the time-sharing system is to run user programs. Time spent running Monitor is nonproductive overhead. Therefore, Monitor must restrict its activities to the minimum time necessary to keep the flow of user jobs going smoothly. In order to meet this goal of minimal overhead, TSS/8 Monitor is used as an interrupt processor only. Monitor is never run except in response to an interrupt. The interrupt trap address in field 0, location 1, is its only entry point. It always exits by dismissing the interrupt. When it completes the handling of an interrupt, Monitor dismisses back to the user job. The job is allowed to run until the next interrupt; this being the only way which Monitor can regain control once a user job has been started. Since the "steady state" of the system is the running of a user job, the interrupt handling technique assures that the system will be doing that as much of the time as possible.

Interrupts to Monitor are divided into three levels: level 0, level 1, and level 2. The clock is the only level 0 interrupt. The workings of the clock routines are dependent on the TSS/8 configuration. With a PT08 system, there will be a line frequency clock. With a DC08, there will be just the DC08 baud clock. This clock then serves both as the system clock and the signal to enter the DC08 service routines. Monitor does not take action on every clock interrupt. It waits for 50-millisecond intervals (3 ticks of a line frequency clock - many ticks of a DC08 clock). Thus, when the clock interrupt occurs, the clock interrupt handler simply increments a counter to see if 50 milliseconds have elapsed. If not, the interrupt is dismissed. (On a DC08 system, the DC08 service routines are run to scan the lines for incoming characters and to keep up output.) The level 0 clock interrupt is just that simple. Only at 50-millisecond intervals does it involve more Monitor processing. In this case, it is treated as a level 2 interrupt. Level 0 has its own register save area and hence may interrupt any other process.

The level-1 interrupts are the device interrupts; reader, punch, disk, DECtape, etc. If the system has PT08s, the console teletype interrupts are also level 1. In the case of the paper-tape reader and punch, the interrupt processing generally consists of transferring a character between the device and a Monitor character buffer. DECtape and disk error flags are also disposed of immediately; the transfer is retried. In all these cases, the interrupt is dismissed immediately.

Since they are all brief, none of the interrupt processors above reenable the interrupt before dismissing. Therefore, they have no problems in protecting themselves against being reinterrupted. This is not the case with any of the other interrupt processors. These, which are considered to be level-2 interrupts, reenable the interrupt before they start processing. The level-2 interrupts may be best characterized as those which take a long time to process. The level-2 interrupts consist of the 50-millisecond clock, a reader character which fills the buffer, a punch character which empties the punch buffer, a disk or DECtape completion flag, or a trapped user IOT interrupt. Level-0 and level-1 interrupt handlers take up a miniscule amount of code. Therefore, Monitor may be thought of as a large level-2 interrupt processor.

Since level-2 interrupts are serviced with the interrupt reenabled, there is the possibility that they themselves may be reinterrupted. Level-0 and level-1 interrupts present no problem. The level-2 interrupt code does its own register saves (AC, PC, LK, and location 0) to assure that interrupts from the other levels do not interfere. A second level-2 interrupt, on the other hand, causes problems. It makes no sense to suspend the handling of one level-2 interrupt to go off and start on another. Therefore, Monitor checks for, and prevents, this situation. Whenever a level-2 interrupt is detected, Monitor checks to see whether it was a user mode program which was interrupted. (The state of the user mode bit is automatically saved when an interrupt occurs.) If the processor was interrupted out of the user mode, indicating that a user program was running at the time of the interrupt, then it is permissible to process the level-2 interrupt. Monitor proceeds to do so. If, on the other hand, the processor interrupted out of exec mode, this means Monitor was in the process of handling a previous level-2 interrupt (the only condition under which an interrupt out of exec mode could occur). In this case, processing of the new level-2 interrupt is deferred. It is placed on the level-2 queue. Entries in the level-2 queue are addresses of the routines to handle the specific interrupt. Once this is done, the interrupt is dismissed, since it will dismiss back to a location within Monitor. At the completion of each level-2 interrupt, Monitor checks this level-2 queue. If it is empty, it dismisses back to the user program. If it is not, Monitor is reentered to process the next request on the queue. Only when the backlog on level-2 queue is exhausted does an exit from Monitor occur.

In the case of 50-millisecond clock interrupts, the level-2 handlers save the whole state of the user job in his job status registers. When finally dismissing this interrupt, the saved state of the job whose

number is in core register JOB is restored. If, in the meantime, schedular has changed the contents of JOB, it will in fact be a new job which is started up. Thus, even the system scheduling is accomplished by means of the interrupt handlers.

It is important to keep this concept of Monitor as interrupt handler in mind since the system is incomprehensible when viewed in any other light. All actions by Monitor may eventually be traced to some interrupt. Swapping occurs in response to a disk completion flag. When the disk completion is detected, Monitor, via a level-2 interrupt, looks to see what the next swap should be and, having found it, initiates it. Scheduling occurs as a response to the 50-millisecond clock level-2 interrupt. If it is the fourth such interrupt since a user job has been started, Monitor looks for a new job to run. Even in the interim level-2 clock interrupts, Monitor tries to do some advance scheduling. If there is no swapping going on, it sees if it should start some. Thus, while a job is running, schedular tries to get the next job ready, so that it may be started up immediately after the current job completes its time slice. When a job does complete its time slice, schedular's task is set up and start the next one.

I/O is also maintained as a function of the clock interrupt. Every 90 milliseconds a Teletype I/O level-2 interrupt is processed. At this time, all input characters (typed within the last 90 milliseconds and now sitting in the PT08 or DC08 service input buffers) are gathered up and placed in the appropriate job's multicharacter input buffer. Output characters are also passed from Monitor output buffers to Teletype buffers at this time.

All interaction between the jobs and the system take place through the medium of the IOT traps. The schedular is heavily dependent on the state of each job's input and output. For now, we will just look at the IOT trap handling in general, indicating how various classes of IOTs are handled.

Once it has identified an IOT trap interrupt, Monitor tries to identify the IOT that caused it. At the time of the interrupt, the PC, which is stored in location 0, is set at the address following the IOT. This pointer is backed up and the IOT is fetched from the user's core. This IOT is then tested against a dispatch table of all valid IOTs. If the trapped IOT is found, Monitor dispatches to the appropriate routine. If it is not, the IOT is undefined. Control is returned to the user program. The IOT is treated as a NOP.

Some valid trapped instructions do not return to the user program at all. HLT is the obvious example. HLT means, quite specifically, do not return control to the program. Control of this job passes to the system (how this all operates is discussed in the next section). Other IOT always cause control to be returned to the user program immediately. Among these are all the "phony" IOTs such as TOD, USE, etc., which have nothing to do with the actual input/output. IOTs, as they are used by programs running under TSS/8 do not necessarily mean instructions used to drive I/O devices. They are actually

instructions which allow a job to talk to the outside world, whether it be a peripheral or just TSS/8 Monitor. Those IOTs which communicate just with Monitor return to the user program immediately.

The IOTs which correspond to the actual devices, such as the Teletype IOTs, may or may not return to the user program immediately. A KSF or TSF which is true, i.e., the keyboard has one or more characters in it, or the teleprinter buffer is not yet full, allow control to be returned immediately to the user program (with the skip simulated). Similarly, a KRB which successfully gets a character and a TLS which does not fill the output buffer allow control to be returned to the user program. In these cases, the user program will be able to do more useful running. After a true KSF, the program can do a KRB to pick up the character. After the KRB, it can process the character, then look for more input. Similarly, after a true TSF, the user program can do its next TLS. If the TLS does not fill the buffer, it can continue processing, or outputting.

6.2 I/O WAIT CONDITION

The user program is only allowed to run again after one of these IOT's if it is free to do some useful work. In the opposite cases, where the output buffer is full or the input buffer empty, there is no expectation that the user program can continue processing. It is an I/O wait state if it is looking for input which isn't there (false KSF or unsuccessful KRB) or trying to output where there isn't any room (false TSF). On a stand-alone PDP-8, the program goes into a wait loop until it can do more I/O. Under TSS/8, user programs which must wait for I/O are not allowed to loop. They are stopped until the wait condition has ended. (Note that this prohibits programs from overlapping I/O and processing within themselves. Time spent in I/O wait is used to run other jobs rather than the job which is in the I/O wait. Note also that the wait condition does not occur on a character-by-character basis. All I/O is done on a buffer-by-buffer basis to allow programs to keep up full I/O rates, even though they spend much of their time in I/O wait states.) All other user job I/O is handled in a manner analogous to that of the Teletype. In all cases, buffers of characters are passed between Monitor and user programs. The programs go into an I/O wait until Monitor has successfully completed the transfer of that buffer.

Scheduling is highly dependent on the state of the I/O. Therefore, the IOT trap handlers keep a status register (the "wait mask") to indicate what I/O device the user is waiting for. The mask, which corresponds exactly to the user's status register (STR1 and STR2), has a dummy bit, the "job is not waiting" bit that is set when the user program is not in an I/O wait. Whenever an IOT trap occurs and the user is to be stopped, the bit corresponding to the device which he is waiting for is set. Thus, if the user program executes a KSF when its input buffer is empty, the bit in the wait mask which corresponds to the keyboard flag is set. The user is not restarted and control goes to schedular so that another user

may be run. Thus, whenever a user is in an I/O wait, a single bit in the wait mask indicates the device it is waiting for. (Some transfers, such as file reads and writes, always put the user into a wait state. Others, like the console do so only when a buffer fills.)

The schedular makes use of the wait mask to decide which jobs need to be run. First, for each user, schedular keeps a run bit in the job status register. A user's run bit is on if he has a program in progress. The run bit is set when the user starts his program. It remains set until the program is halted. Those users whose run bits are not set are never scheduled to be run. Among those jobs whose run bits are set, only those which are not in an I/O wait are actually scheduled to be run.

In deciding what user to run next, schedular scans through the list of active jobs looking for one whose run bit is set. Finding such a job, it sees if the wait mask ANDed with the job status flags, is nonzero. If it is, the job is runnable and is scheduled to be run. (Note: if the job is not in I/O wait, the dummy bit, the "job is not waiting" bit, is set to assure that the job will be runnable.) If the job is in an I/O wait, the wait mask ANDed with the status bits will be zero. Only one bit in the wait mask will be set - the bit corresponding to the flag which the job is waiting for. This flag is zero at the time the wait mask bit is set (otherwise, the job would not be in an I/O wait). In this way, jobs which are in an I/O wait are prevented from being scheduled.

A job breaks out of an I/O wait when the flag corresponding to the bit in the wait mask comes on. For example, assume that a job is waiting for the keyboard. Eventually, the user will type a delimiter on the keyboard. This will cause the delimiter bit to be set. The next time schedular checks this user's status, the wait mask ANDed with the status bits will be nonzero. The job is then runnable again. In general, flags are cleared by IOT trap-handling routines. Clearing a flag means a wait condition; at the same time a flag is cleared, the corresponding bit in the wait mask is set. Flags are generally reset by level-1 interrupts, i.e., those that do the data transfers. They are detected by the level-2 schedular when it looks for the next runnable job.

This mode of operation characterizes the whole Monitor. Monitor is made up of a number of asynchronous elements which communicate via status registers and request queues. Schedular, which is the heart of Monitor, is guaranteed to be run every 50 milliseconds. Therefore, it is not necessary for another routine, such as the disk handlers, to jump directly to the schedular in order to indicate that a swap is complete. All that the disk routines need to do is set the appropriate status bit to indicate the new system status. The next time the schedular is run, it will find this updated status and will act accordingly. Similarly, if the schedular decides that a swap is needed, it may simply queue this request if the disk is active. When the completion flag for the present transfer is processed, the disk queue will be checked and the queued transfer initiated. However, if there is no disk transfer in progress when the schedular decides to do a swap, it cannot just queue the request. In this case, the schedular itself must initiate the transfer.

6.3 OTHER PARTS OF MONITOR

The Monitor code which performs the functions discussed so far is permanently resident in field zero while TSS/8 is operating. Field zero contains almost all the resident code. Monitor also occupies field one. About 1K of field one is used for code, most of it for device handlers. The remainder is for tables and buffers. Nearly all of the resident Monitor data base is in field one.

In addition, there are three nonresident sections of Monitor code. They are the System Interpreter (SI), the file handler (FIP), and the error handler (ERP). These are the routines which are not frequently used and hence do not need to be core resident. FIP is a 4K block of code which resides in the second 4K block of the disk (disk locations 10000-17777). SI and ERP together make up a 4K block of code which resides at the bottom of the disk (disk locations 0-7777). When needed, these routines are brought into field two for execution. They do not overlay resident Monitor; they go into the first user fields. In fact, schedular sets them up so that they look just like a user program. They run in the place of the user program which called them. For this reason, they are referred to as "phantoms." (FIP = File Phantom, ERP = Error Phantom.) They are not, however, identical to user programs because they are run in exec mode. This means they may read and write physical disk segments (in the case of FIP) and get at field 0 and 1 data and subroutines.

6.4 THE MONITOR DATA BASE

Some mention has been made of the tables and buffers used by Monitor. Diagrams of these tables may be found at the end of this manual. These should be referred to as specific tables are mentioned. A brief discussion of the tables follows.

Monitor does a great deal of dynamic storage assignment. It makes use of a pool of 8-word blocks known as the free list. At system startup, the unused area in field 1 is divided into these 8-word blocks and linked together by a list structure. The first word in each block is a pointer to the next block. The last block contains a zero pointer. A fixed core register, FRELST, always contains a pointer to the top of free list; a second register, FRECNT, indicates the number of available free core blocks. This free core is used by Monitor for a variety of purposes. Teletype buffers are made up of linked blocks of free core; device and job status information is also stored in free core. Free core is also used for temporary scratch storage in a number of instances.

The device handlers for Teletypes and the assignable devices make extensive use of free core. Both are based on a single, fixed-length table of devices, DEVTBL. DEVTBL contains a one-word entry for each system device (a console counts as two devices: keyboard and teleprinter). If the device is unused, the entry is zero. If it is active, the entry contains a pointer to a block of free core known as the device data block (DDB). This block contains the status information for that device. In addition,

there is a buffer for each device. In the case of the assignable devices, this buffer is a fixed, dedicated, core buffer. In the case of Teletypes, the buffers are dynamically allocated from free core. As characters come in from the keyboard and are put in the buffers they are put into 8-word blocks of free core. As one block fills up, another is fetched from free core and linked to it. As characters are fetched from the buffer and passed to the user program (via trapped KRB's), blocks at the other end of the buffer are emptied and returned to free core. Within the DDB are pointers to the head of the buffer (the "fill pointer"), which indicates where the next character to be put into the buffer is to go and to the tail of the buffer (the "empty pointer") which indicates the next character to be pulled out of the buffer. Input buffers and output buffers work the same way.

This console input and output operate independently from the rest of the system. As characters come in, they are put in input buffers (up to 80 characters). If the character is one designated as a delimiter, the user's keyboard status bit is set. As characters appear in the output buffer, they are typed. Buffers expand and shrink to meet the needs of the moment. This is the limit of the responsibility of the Teletype handlers. They merely pass characters and adjust the appropriate flags.

Just as each active console is marked in DEVTBL, so each active job is marked in a job status table, JOBTBL, which is a fixed table with a one-word entry for each possible system job. Nonexistent jobs are marked by zero entries. Existing jobs have an entry which is a pointer to an assigned free-core block which is its first job status block. Each job actually has several blocks of status information, all linked together. In these status blocks are kept all information about this job's running state. If there are open files, blocks containing their status also exist.

Finally, there are tables Monitor keeps which indicate the status of the system. CORTBL, which indicates where jobs are in user cores, is the most important of these.

Chapter 7
System Storage and Communication

7.1 TALKING TO THE SYSTEM

Up until now, we have assumed that jobs running in the system either did no I/O or simply did console I/O. In doing this console I/O, characters were passed in a manner analogous to a stand-alone PDP-8. No mention was made of how the program was started up in the first place, much less how it was loaded and otherwise controlled. These are functions which, on a stand-alone machine, are not performed through the Teletype at all - they are done through the switches on the console. When talking to TSS/8, however, there is only one physical device, the user's Teletype, through which to perform these two kinds of communication: communicating with the TSS/8 system and communicating with a user program running as a job within that system.

TSS/8 makes a careful distinction between these two modes. A user is always uniquely in one mode or the other, depending on the state of his job. Whenever a user starts executing a program, his console is put in program communication mode. It stays in that mode until the program is interrupted or terminated. If the program is terminated, the console automatically returns to system communication mode. It is also possible to make one-shot inputs to the system without halting the user program.

In order to minimize confusion, TSS/8 has some conventions to distinguish between system and user mode. The system always types a period (.) at the margin to indicate that a Teletype is in system mode and that the system is ready to accept a new command. The control B (B with the control key depressed, abbreviated to a ↑B in all TSS/8 documentation) character tells the system that, regardless of the mode the Teletype is in, the characters following the ↑B are to be treated as though the Teletype were in system mode. Thus, even if the Teletype is in user program mode, all characters following a ↑B, up to the next carriage return, are input to the system.

When the user walks up to a TSS/8 console, he finds it in system mode. If he types carriage return, thereby entering a null command, TSS/8 responds with a period at the margin. The user may then type a command to the system. At this point, the Teletype is actually in a special system mode - it is logged out. This means 1) input is not echoed to the teleprinter, and 2) only two commands, LOGIN

and TIME are considered valid. The system will call all other commands illegal. Thus, the first thing a user does is type a LOGIN command, which consists of the command LOGIN followed by an account number and a password. If the account number and password are valid, the user is logged in. His Teletype remains in the system mode, but his input is now duplexed and all system commands are now valid. (If the login is invalid, he remains in the unlogged in system mode and must try again.)

The user remains in the system mode until he types a command which causes a program to be started for him. He does this by means of the START command which takes an octal address as an argument. (He may also start a program with an R or RUN command.) The START command both starts the program and puts the Teletype in user program mode.

Once a program has been started, there are two ways to stop it, thereby returning the console to system communication mode. One is for the program to execute a HLT. This stops the program. The other way is to type an S (for STOP) command to the system. However, since the Teletype is in user program mode, it is necessary to preface this S by a ↑B in order to get the attention of the system. Notice that by typing ↑B while a program is running, any command may be entered to the system. Only S, however, will send the Teletype back to system mode. For the others, the program will continue to run and hence the Teletype will return to user program mode.

So far we have only talked about three Monitor commands: START, STOP, and LOGIN. There are, however, a great many more. (They are described in the TSS/8 System User's Guide.)

The set of commands enumerated there is designed to give the user convenient and comprehensive control over his program. He may do debugging tasks with commands such as EXAMINE and DEPOSIT. He may store and retrieve programs by SAVEs, LOADs, and RUNs, he may control additional peripherals by means of ASSIGNs and RELEASEs, etc.

The handling of all these system commands is accomplished by means of a nonresident system phantom called the System Interpreter. SI's task is to scan and interpret system input strings and either execute them directly or reduce them to a concise coded form to be executed by another part of TSS/8 Monitor. It is called by the Teletype handlers (part of resident Monitor) whenever a system command (often referred to in the documentation as an SI string) is input.

Characters being input to the System Interpreter are handled by the Teletype input routines exactly as are characters being input to a user program. In either case they are placed in the multicharacter Teletype input buffer until a delimiter is detected. (Delimiters for SI strings are CR, LF, and RO.) It is only when the delimiter is seen that the two types of input strings are treated differently. In the one case, the characters are passed to the user program; in the other, they are passed to SI.

A bit in the input Device Data Block, the "route characters to SI" bit is used to remember that an input string is actually an SI string. This bit is always set when the Teletype is in system mode. It is also set whenever a ↑B is input. A command to start running a user program clears the bit.

If the "route characters to SI" bit is set, input characters are checked against the System Interpreter delimiter mask (carriage return, line feed, and rubout). If the input character is a delimiter, a second DDB bit, the "SI command delimited" bit is set. Also, a schedular register, COMCNT, is incremented.

COMCNT, at any given instant, reflects the number of users who have typed in a whole command to the system and are waiting for a response. Schedular checks COMCNT every time it runs. As long as it is zero, everything is up to date. However, if COMCNT >0, this means that someone has an SI string waiting. In this case, the System Interpreter is scheduled to be swapped in and run. It is brought into field two and started up just as any other user program. The principal difference is that SI, being a system phantom, is run in exec mode. This means it can execute IOTs without trapping back to field zero. Specifically, it may do a CDF into Monitor core in order to inspect the DDBs. When it finds an "SI command delimited" bit set, SI knows who called it.

Once it has found who called it, SI reads the command string to find out what the basic command is. SI has a dispatch table for all valid commands. For commands which take arguments, the string is scanned to pick these up. If an error is detected anywhere along the line, SI exits back to Monitor after typing an error message back to the user. If the command is valid, SI must decide what to do with it. SI is capable of executing many commands on its own. For the rest, it calls for help. For all file operations, it must call still another nonresident subsystem, the File Phantom. For these commands, SI reduces the input string to a concise command code which is then passed on to the appropriate portion of Monitor.

SI itself is essentially reentrant. It gets its input, the command string, from Monitor core, operates on it, and puts any output, either a response string to the teleprinter or a concise command to be executed by some other part of the system, back into Monitor core. SI may be thought of as the interface between the user and the system. It allows the user to enter commands in a simple format. These input strings are translated by SI into a form which the rest of the system can understand. It resides on the disk and is called in to perform this interpretation and translation function whenever a user requires it.

7.2 DISK STORAGE AND FILES

Up to now, we have mentioned the TSS/8 disk only in its capacity as a swapping device. For each job, there is a dedicated 4K area on the disk in which it is stored when it is swapped out. This is not, however, the only way in which the disk is used. The low-order tracks of the first disk are used to hold an image of the system. 0-7777 contains the System Interpreter Phantom. 10000-17777 contains the File Phantom, part of which is tables and part of which is code. The entire 4K of FIP is brought in whenever it is called. If FIP updates any of its tables, these are written back out to their place within the disk image. SI, which contains no internal tables, is never written back after it is called. The next 4K of the disk contains an image of the system Initializer. It is brought in only at system startup time, it is not used while the system is up. It is kept on the disk to allow for easy system restarts. The next 8K is used to hold an image of resident Monitor. It is brought into fields zero and one by the initializer at system startup time. It is not accessed by the running system. Like the image of INIT, it is kept on the disk to allow rapid recovery from crashes.

The area of the disk immediately above the system image is used for the swap areas. There is a 4K for each possible system job. (A 16 user system thus uses 64K of the disk for swap tracks. This plus the 20K of system image totals 84K of disk which is taken for system usage.)

All remaining space on the disk is devoted to on-line file storage. If the system has more than one disk, the additional surfaces are completely devoted to file storage. The file area is allocated in 256 word segments.

TSS/8 provides users and user programs with the capability of setting up files in this area of the disk and of reading and writing them. These files may be of arbitrary size; they are, however, made up of an integral number of disk segments. The user may create a file. Creating a file reserves a segment of file space on the disk and associates with it the symbolic name specified in the create command. The user may open this file, thereby allowing it to be manipulated. He may extend the file a given number of segments, thereby reserving more segments of disk for the file. Extending a file tacks the new segments onto the "end" of the already allocated segments. Reducing a file returns one or more segments from those reserved for this file to the pool of available segments. He may also rename a file. These four basic functions of creating, extending, renaming, and reducing (deleting is accomplished by reducing a file until there is nothing left) have nothing to do with the contents of the file. They merely define and reserve a certain amount of space on the disk.

As far as the user is concerned, these segments are contiguous. He addresses, and therefore manipulates, the file as though it were one big long disk area. The actual size of the file, as determined by creates, extends, and reduces, is important only in that a user cannot write off the end of his file.

The file itself is considered to be made up of 12-bit data words. There are no control words in the file; all the space within a file which a user has defined for himself is available to him for program storage. The user addresses a file by internal file number and an address within that file. The first word of the file has address zero. Using this file address, the user may transfer data between a selected part of his 4K core area and the addressed point in his file. Although only 4K may be transferred between core and a file in one transfer, the size of files is by no means limited to 4K. 18 bits are allocated for disk file addressing. It is worth noting that there is no distinction made between types of files. All files are made up of 12-bit data words whether these 12-bit words contain single ASCII characters (or, indeed, characters of any other code), pairs of trimmed characters, numbers, or whatever, is immaterial to the system as a whole. How the data of a given file is interpreted by a program is, of course, what matters.

The fact that segments of a file appear to the user to be contiguous is, of course, an illusion. Disk segments are, in fact, allocated at random. TSS/8 maintains directories in order to remember which segments are allocated to which files. As mentioned above, the actual segments which make up a disk file are pure data area. Segments of a file are not chained together; there are no header words attached to a segment.

For each user TSS/8 maintains a User File Directory (UFD) which holds the names of all files which a given user is maintaining and the disk segments of which it is comprised. (Note: The diagrams at the end of this manual will help in understanding the TSS/8 file structure.) The UFD is divided into 8-word entries. For each file there is a single file name entry. The first three words contain the file name (6 characters packed in TSS/8 internal format).

Words 4-6 contain information about this file. Words 3 contains a pointer to the next name block in this user's UFD. This pointer is used to chain through the UFD name blocks. The final word of the name block (Word 7) contains a pointer to a File Retrieval Information Block. Each name block is associated with it one or more of these retrieval blocks. They are also 8-word blocks and are interspersed with the name blocks in the UFD file (hence the need to chain the name blocks). The first word of the retrieval block is a pointer to the next retrieval block for this file (or zero if this is the final block). The next seven words contain a list of segment numbers of the segments which comprise the file. The file is considered to run from the first segment in the file to the last. (A zero segment number terminates the list.) The algorithm for associating addresses within a file (the means by which a user addresses his file) and physical disk addresses (the system's ways of addressing) is straightforward. The file address is divided by the segment size. The quotient is the logical file segment number. Counting down the file retrieval block's list of segment numbers to this number yields the physical segment number. (If the list runs out too soon, the user has run off the end of his file.)

In the actual implementation, the UFDs are themselves files. They are made up of disk segments just like any other file. (The 8-word blocks into which the UFDs are divided are merely a software division.) In order to keep track of these UFD files, there is still another directory, the Master File Directory. In format, it is virtually identical to a UFD. It is broken down into 8-word name and retrieval information blocks. The 3-word names in the name block are, however, login IDs rather then file names. The first word contains the account number as a 12-bit binary number, the next two words contain the four character password, packed in internal code. Taken altogether, these three words constitute the "name" of the UFD. (The MFD is, of course, also used at login to see if the account number and password are valid.) The file retrieval information block linked to the name block (in the case of the MFD, only one retrieval block per UFD is allowed) contains the segment numbers of the segments which make up the UFD for the user.

To complete the symmetry, the MFD is in turn a disk file made up of segments. It, however, always starts with segment 1.

The MFD and UFDs take care of the problem of allocated disk segments. There is one further table, the Storage Allocation Table (SAT), which keeps track of unallocated segments. SAT is a bit table which is set up when the system is refreshed. It contains a bit for each segment ...the bit is cleared if the corresponding segment is available, it is set if that segment is allocated. All requests for disk segments get them from the SAT table routines. Similarly, no longer needed segments are returned to the SAT. For example, if a file is to be extended a segment, the SAT routines are called. They return with the number of an available segment, which is added to the list of segments in the retrieval blocks for that file. Files are reduced by deleting the last segment number or numbers from the list and clearing the corresponding bit(s) in the SAT table.

7.3 TALKING TO THE DISK: THE FILE PHANTOM

Most of the tasks described in Section 7.2 are accomplished by a second nonresident section of Monitor, The File Phantom (FIP). FIP handles all disk manipulations except actual reads and writes. Like the System Interpreter, it resides on the disk. It is called by Monitor to perform functions which cannot be handled by resident routines. All tables relating to the disk files are kept within the 4K which FIP occupies. They are swapped in with FIP whenever it is called. Whenever they are updated, the tables are immediately written back to the disk by FIP. In this way, the disk always contains all information about itself. The disk is thus protected against loss in most system crashes.

FIP's primary task is to do the file handling. It maintains the UFDs, the MFD, and the SAT. It performs all the needed searches of these tables. It executes the basic file commands of CREATE, EXTEND,

REDUCE, and RENAME as discussed above. They all happen independently of resident Monitor; they result in changing the status of the disk only. PROTECT is similar. It allows the protection code on a file to be altered but nothing more. OPEN and CLOSE, however, are somewhat different in nature.

OPEN and CLOSE do not alter the disk in any way; they simply establish a link between resident Monitor and a disk file. (The fact that OPEN and CLOSE do not affect the disk is important. Newly created files exist even if they have not yet been closed out.) Each job may have up to four files open simultaneously. There are four registers in the last job status block which record the status of these four internal files. If there is no file open on an internal file number, its corresponding job status block word is zero. (See diagrams of job status blocks.) When a file OPEN command is given, FIP sets up a new status block in free core. This block is used to hold pertinent information about the open file. A pointer to this file information block, which remains set up as long as the file is open, is placed in the job status block register for this internal file. At the same time, FIP sets up a second block in free core for this file. This block, the file window, contains one of the file retrieval information blocks from the UFD. At the time of the OPEN, the first file retrieval information block is put in the window. At the same time, the fact that this is the first window is recorded in a register of the file information block. Once all this is done, the OPEN is complete. CLOSE merely dismantles all this and zeroes the register in the last job status block which corresponds to this open file. Opening a file automatically closes any file which was open on that internal file at the time.

CREATE is the only file command which does not have to be preceded by an OPEN. All other file commands operate on internal file numbers rather than file names. In the case of EXTEND, REDUCE, and PROTECT, this is to allow for file protection. The file protection apparatus is part of the OPEN routines. Files which are read-protected against a user cannot be opened by him. If a user is allowed to read but not write, he is allowed to open but a write-protected bit is set in his file information block in free core. EXTEND and REDUCE are considered to be the same as writing. They are prohibited if write-protect is indicated. The PROTECT command, which sets these various modes of protection, is illegal except for the file owner. Finally, there is an implied protect on files which are open to more than one user. If a file to be opened is already open to another user, it is write-protected to prevent confusion.

RFILE and WFILE, the file read and write commands, require the file to be open, because they need the information in the open file information blocks. RFILEs and WFILEs do not, in general, require FIP to be called. Resident Monitor attempts to execute them itself. It takes the file address given as a parameter to the command and compares it against the state of that file's window. It sees if the segments in the window correspond to the part of the file involved. If so, it goes ahead and executes the transfer. (Note, that if it is a write, the write-protect bit in the file information block is

checked first.) If the window is not properly set, resident Monitor calls FIP to move the window so that it is looking at the specified part of the file. FIP then returns to Monitor so that it can do the transfer.

FIP is called whenever Monitor discovers a request that it cannot handle. Before calling, it must set up the appropriate command and parameters so that FIP will know what to do. This command is always in the form of an IOT; one of the TSS/8 IOT's. If parameters are involved, they are passed in precisely the format that they are specified for the IOT itself. Thus, CREATE takes three words of parameters, OPEN 5, etc.

Whatever the IOT, it and all its parameters are placed in a block of free core. A pointer to this block is placed in the job status block register referred to as JOBLNK. FIP is then called. If it is to return parameters, it does so in this same block. As soon as the block is no longer needed, it is returned to free core. Some IOTs do not take parameters. The AC itself is the only parameter. In this case, no IOT Parameter Block is needed. The IOT itself goes into JOBLNK. (The AC itself is, of course, stored in another job status block.)

FIP maintains the Storage Address Table (SAT) which is located in the high end of FIP's 4K. Whenever the SAT is changed (a segment is allocated or deallocated), it is written back to the disk so that the next time FIP is brought in, an updated version of the SAT will come in with it. The SAT is the only permanent table that FIP maintains. It is never changed by a system restart. (Refreshing, of course, clears the SAT.) All other tables and data areas maintained within FIP are kept only as long as individual users are logged in. They are cleared on a system restart.

FIP handles all the open-file information which is linked into job status blocks. These are set up on an OPEN, cleared out on a CLOSE, and suitably updated whenever a file is changed. FIP also maintains some internal tables which make its operation more efficient. For example, when a user logs in, FIP opens that user's UFD. It gets the retrieval information block from the MFD and stores it in a table. By doing this, FIP doesn't have to scan the MFD every time it wants to find a UFD. FIP also remembers how many users are logged in under this account number or are using a file belonging to the account.

Finally, FIP does all updating of the directories, the MFD and UFD's. It has a 256-word buffer into which it can read directory segments. FIP scans directories by reading them in one segment at a time until the desired entry is found. If it is changed, this segment is then written back out to the disk. If the directory is extended or reduced, FIP updates the appropriate retrieval information block in the MFD.

See the diagram section for a more detailed discussion of the FIP tables.

7.4 DISK TRANSFERS

All disk transfers, whether they are swaps, user program I/O requests, or FIP table or directory transfers, are handled by a common disk routine. All disk transfers go between user fields; resident Monitor never does transfers into field 0 or 1. The common disk routine takes a standard set of parameters which are stored in a block of free core. They are: direction of the transfer, the field involved, the disk address (physical), the core address, the number of words to be transferred, and the address of the routine to go to when the transfer has been completed. The disk routine sets up the transfer, does it, and then dispatches. If it tries three times and fails, it dispatches to an error handler instead.

Since requests to do disk transfers can pile up, there needs to be some place to queue them. In the case of swaps, there is a single register SWREQ. If it is zero, no swap is pending. If it is nonzero, it points to a parameter block for the next swap, in or out. Swaps get first priority. When the current transfer is done, this swap will be done next.

All other transfer requests are held in DSUTBL (often referred to as the disk queue). DSUTBL has a four-word entry for each core field. A nonzero entry indicates that a transfer is pending for that core field. (The entry points to the parameter block.) Within the four-word entry, each word corresponds to an open file. Thus, if the job in field 3 wishes to read open file 2, it executes an RFILE. Resident monitor uses the retrieval window for that file (calling FIP to move it if necessary) to figure out the physical disk address, builds a parameter block in free core, and puts a pointer to it in the third word of the DSUTBL entry for field 3. The program is then put into the wait state until the transfer is complete. It is, however, prevented from being swapped while this transfer is taking place. This is done by setting the LOCK bit in CORTBL to lock the user into core. This bit is cleared when the transfer is completed. (Disk transfers, which are not buffered in Monitor core, are the only I/O operations which require that the program remain in core.) Even FIP, when doing directory transfers in and out of its own area, or writing out its internal tables, uses the DSUTBL device for queuing requests.

7.5 ASSIGNABLE DEVICES

All TSS/8 systems include a high-speed, paper-tape reader. Some include a high-speed punch and/or DECtape. These devices comprise the assignable devices for the system. They may be used exclusively by individual on-line users.

Assignable device handling breaks down into three sections: assigning and releasing the devices, a device handler, and code to pass data between Monitor buffers and the user program. Assignable devices have their slots in DEVTBL just as the Teletypes do - the last 10 registers correspond to reader punch and DECtapes 0 through 7. If the device is not assigned, the corresponding register in DEVTBL

contains zero. When a user requests a device (and it is available) a Device Data Block is set up and linked into DEVTBL. Within the DDB, is stored the number of the job which now owns the device. Whenever a reference is made to this device, the referencing job is checked against this job number to assure that it is the right one. No error checking is done at assignment time. Thus, all eight DECtapes could be assigned even though only two transports exist. When a user releases the device again, the DDB is freed up and a zero is returned to the DEVTBL entry. Also, the amount of time that the device was assigned is added to the user's device time. In this way, use of assignable devices is reflected in the accounting information.

All assignable devices have fixed, one-page buffers in Monitor core (the DECtape buffers are actually 129 words). They are too fast to use the linked free-core buffers used for Teletype I/O, but they are too slow to hold the user in core and transfer directly into the user field as is done with the disk. When the device is started up, it remains active until the buffer is completely filled, then stopped until it is completely emptied (or vice versa for output). No attempt to double-buffer is made.

For example, the paper-tape reader is activated by a RRB IOT. Finding the buffer empty, Monitor puts the user job into an I/O wait state, clearing its reader flag and setting the corresponding bit in the wait mask. It then sets up the reader service routine to read 128 characters into the reader buffer. When the buffer has been duly filled, the user's reader flag is reset, making him runnable again. The program then executes successive RRB IOT's to pick up individual characters. When the buffer empties again, the process repeats. The user may initialize a read, and clear the buffer by executing an RFC instruction.

Operation of the high-speed punch is very similar. The running program passes characters to Monitor, via trapped PLS instructions. These go into the punch buffer. If the buffer fills, the job goes into the wait state until it is emptied again. One difference is that punching is begun whenever any characters are in the output buffer. Monitor does not wait for the buffer to fill. Thus, a user program may overlap execution with punching as long as the buffer doesn't fill.

DECtape handling is similar. DDBs are set up when they are assigned and returned when released. Since there are eight possible DECtapes, Monitor reserves eight words in DEVTBL. DECtapes are the only read/write devices in the system (except, of course, for the disk). The same DDBs and buffers are used for reads as for writes. Since the DECtape controller allows access to only one transport at a time, there is no point in having a Monitor buffer for each one. In fact, there are two, regardless of the number of units. At the time a user program requests a transfer, Monitor assigns one of the two buffers to be used for that transfer. The job is then put into the wait state until the DECtape block is found and the transfer made. At this point, the buffer is available to another job. Thus, if more than two DECtapes are active, the jobs compete for the two buffers.

Although these are the only peripheral devices supported by TSS/8 Monitor, they provide a good model for users who may wish to incorporate their own special devices. In all cases, three software modules are involved: one to handle device assignment, one to handle data transfers between the user program and Monitor, and one to do the actual device handling. Space in Monitor is available but not in large quantities. Therefore, high-speed devices are probably inappropriate. There simply isn't room for a buffer big enough to keep up the data rates.

7.6 ERROR HANDLING

TSS/8 Monitor allows the user program a great deal of freedom in the way it utilizes system resources. Therefore, system error checking is kept to a minimum. Any job is free to do anything which does not affect another job, or the system as a whole. For example, a program may wipe itself out without interference from the system.

The first level of error handling comes when a user program requests Monitor to do something it cannot do, for example, opening a file that does not exist, or reading from an internal file number for which no file is open. For all such logical errors, Monitor returns an error code to the user program. (See the System User's Guide.) Not all of these errors are simple logic problems. For example, trying to create or extend a file when the disk is full returns an error. Running the same program some other time would give no error. Another nonlogic error is the parity error or directory error on a file read or write. This is the result of a physical malfunction of the disk. A transfer error occurred either within the file itself or within one of Monitor's directories.

The second level of error handling comes when a user program requests something which Monitor cannot do. For example, it requests service from the high-speed reader when someone else owns it, or when it is assigned properly, but there is no tape in it. Another example is a physical disk error when trying to swap this job in or out. In these cases, it is impossible for these jobs to continue. Therefore, Monitor terminates them, typing out an error message and the state of the active registers. User programs may, however, request that they be allowed to handle such problems. They do this by executing an SEA command, which gives Monitor an address to jump to when such an error occurs. This routine is responsible for finding out what the error was (the error code is in job status word 1 where it may be fetched by a CKS IOT), and responding to it.

Monitor also does internal error checking which is not apparent to the user. All disk transfers are tried three times. Only after the third try is a disk transfer error actually reported. When it starts up the reader or punch, Monitor puts a timer on them. If there is no interrupt from the device within a certain amount of time, Monitor signals a hung device.

Chapter 8
Details of Monitor's Data Base

8.1 INPUT/OUTPUT DATA BASE

All I/O, except for the disk, is controlled from a single, fixed-length table, DEVTBL. Actual data about the status of each device is held in a Device Data Block (DDB). DDBs are dynamically assigned blocks of free core. The actual data to be transferred is contained in buffers. In the case of Teletype I/O, these buffers are dynamically assigned blocks of free core. One to eight (linked) blocks of free core make up a Teletype buffer. Teletypes are considered to be two devices, a keyboard and a teleprinter. Each has a DDB, and each has its own buffer. The assignable devices, which have higher data rates, do not use dynamic core buffers. They have fixed, one-page (129 words for DECtape) buffers in Monitor core.

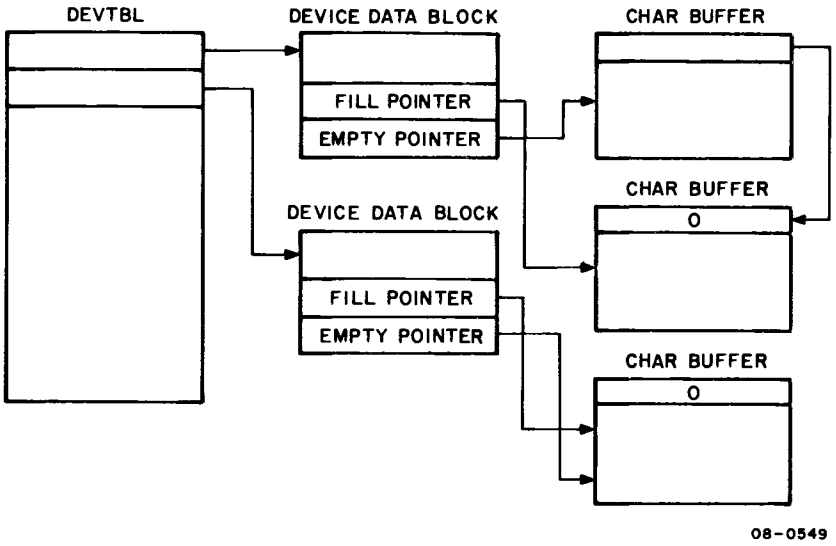


Figure 8-1 Relationship of Tables, DDBs, and Buffers

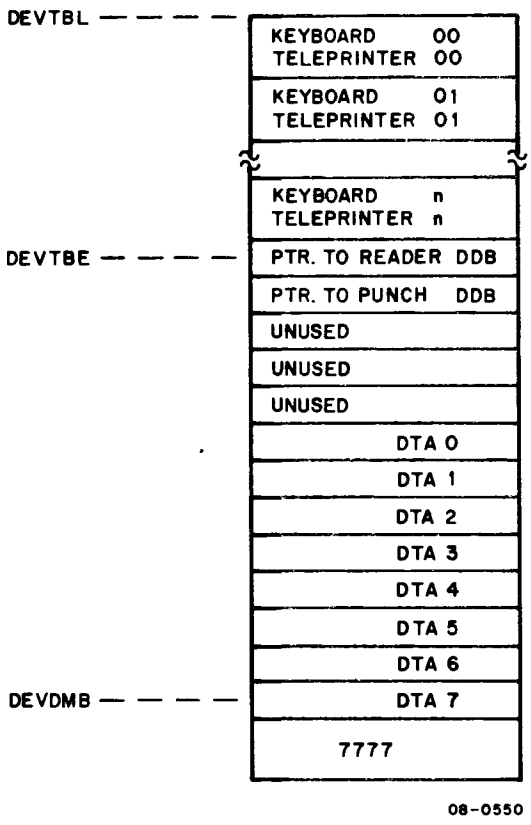


Figure 8-2 DEVTBL

The tables, DDBs, and buffers are linked together by pointers. DEVTBL is, in fact, a table of pointers. If a device is inactive (a Teletype not logged in or other devices not assigned) the corresponding table entry is zero. If the device is in use, the table entry is a pointer to its DDB. The DDB for each device also contains pointers, the fill and the empty pointer. The fill pointer points to where the next character to be put into the buffer should go; it points to the "head" of the buffer. The empty pointer points to the next character to be taken from the buffer. Each buffer block contains in its first word a pointer to the next block. The last block in the buffer contains a zero pointer.

DEVTBL is set up with the Teletype entries first, then the entries for the reader and punch, then three unused entries, then eight entries for DECtape, and finally a 7777 terminator. The number of entries for Teletypes, and hence the size of the table, is dependent on the configuration parameter NULINE,

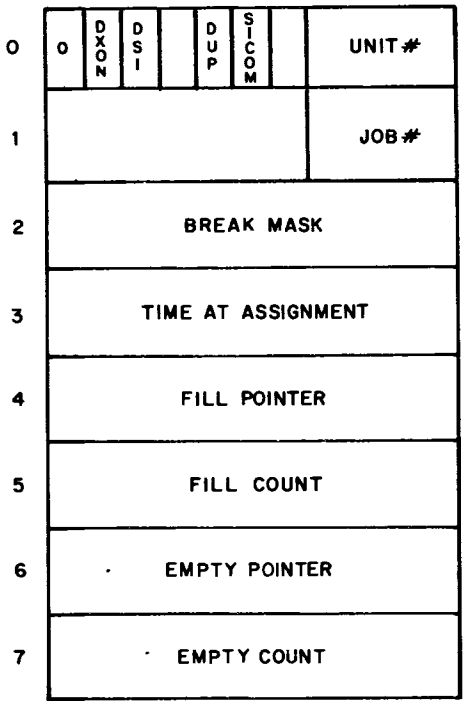


Figure 8-3 Teletype Device Data Block

the number of terminals. DEVTBE marks the beginning of the assignable device section of DEVTBL, which always contains 13 entries even though all these devices may not be included in the system. (At initialization, all slots in DEVTBL which correspond to nonexistent devices are filled with dummy pointers to prevent assignment.)

Device Data Blocks are always 8-word blocks assigned from free core. Bits 7 through 11 of word zero contain the unit number, bits 7 through 11 of word 1 contain the number of the job which owns that device, and word 3 contains the time at which the device became active. This 12-bit time is taken from bits 3 through 11 of CLK2, bits 0 through 2 of CLK1.

Bit 0 of word zero is a zero if the device is a Teletype, otherwise it is a 1. The use of the remainder of the DDB depends on the particular device.

The status bits in word zero of the Teletype DDB are used in the keyboard DDB only. The DXON bit is set when a buffer is almost full and hence an XOF must be sent. When the buffer is emptied, XON must be sent. DSI is set to indicate that the keyboard is in Monitor mode. DUP is set to indicate that the Teletype is in duplex mode; SICOM when set, indicates that the user has just typed a delimiter to a keyboard string. The Break Mask, or delimiter mask, is the value specified by the last KSB IOT. The fill pointer points to the leading block of the buffer. The fill count is the character position (in 2s complement) within that block. The same is true for the empty counter and empty pointer. If no buffer exists, the pointers are zero.

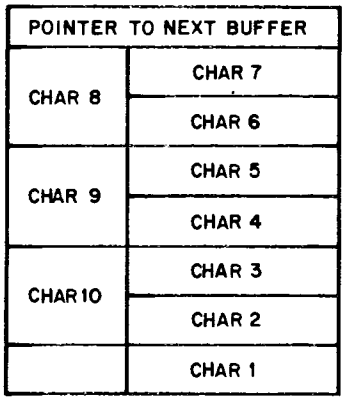


Figure 8-4 Teletype Character Buffer

Teletype characters are packed 10 characters to a block. Characters 1 through 7 go in bits 4 through 11 of words 1 through 7. Characters 8, 9, and 10 are split and packed into the high-order bits of words 1 through 6. Bits 0 through 3 of word 7 are unused.

The DDBs for the assignable devices exist for as long as the device is assigned. They have bit 0 of word 0 set. Bit 1 of word 0 is set when the device is active (initialized).

If a user program IOT or an SI command requires FIP to be called, an IOT parameter block is set up to hold the IOT and its parameters. A pointer to this block goes into JOBLNK. If a FIP IOT is to be executed which requires no parameters, the IOT itself goes into JOBLNK. No IOT parameter block is set up.

8.2 USER PROGRAM STATUS

All job status information is based on a single, fixed-length table in Monitor core, JOBTBL. JOBTBL has a one-word entry for each possible job. If the job does not exist, i.e., no one is logged in on that job, the corresponding entry in JOBTBL is zero. If that job does exist, the entry contains a pointer to the first of three (linked) Job Status Blocks. These contain complete information about the running state of that job. For each file which is open to that job, there are two additional blocks, one of which contains information about the file, the other of which indicates where it is on the disk. While a file transfer is in progress, still another block exists which contains parameters for the transfers. Finally, when executing an IOT which requires a FIP call, a block may be set up to pass the parameters.

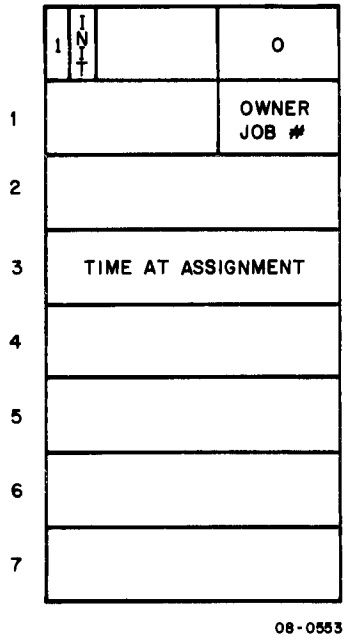


Figure 8-5 Device Data Block - Reader

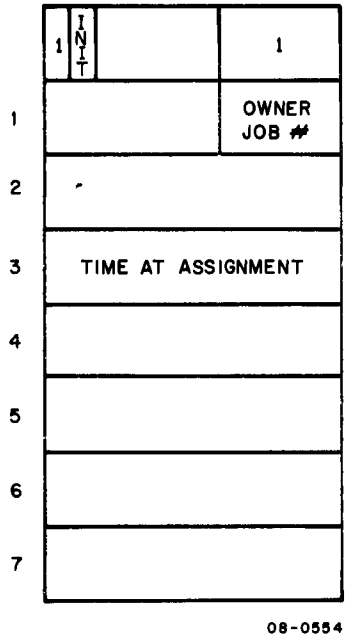


Figure 8-6 Device Data Block - Punch

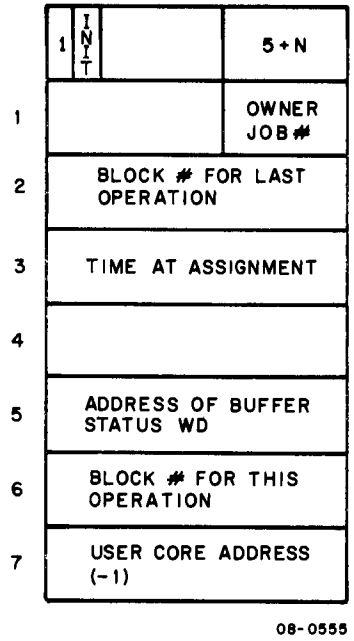


Figure 8-7 Device Data Block - DECtape

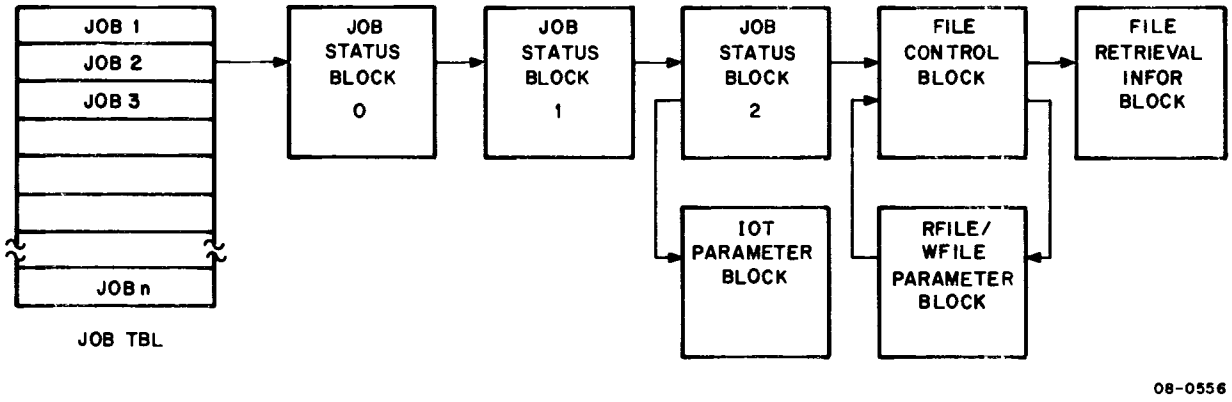


Figure 8-8 Job Status Information

The three job status blocks exist for all jobs. They contain the saved state for the job, AC, PC, LK, MQ, and SC. They also contain the status of the job's I/O. This information is stored in bits of the three status words: STR0, STR1, and STR2.

STR0 contains status bits which are not directly associated with I/O. STR1 and STR2 contain bits which may be considered flags. They are set and cleared according to whether the associated device is ready or not ready. The two words of the wait mask STR1 and STR2. When a job is waiting for a device, a single bit in the wait mask, corresponding to the device bit in STR1 and STR2, is set.

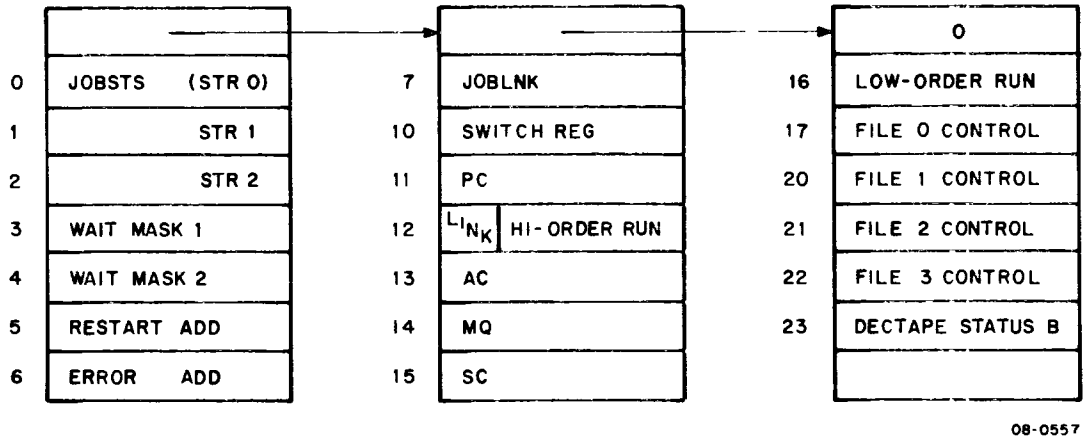


Figure 8-9 Job Status Blocks

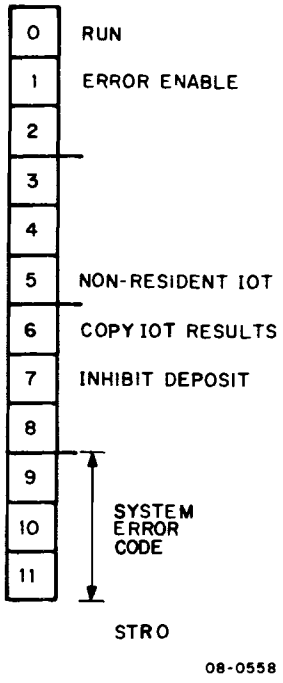


Figure 8-10a STR0

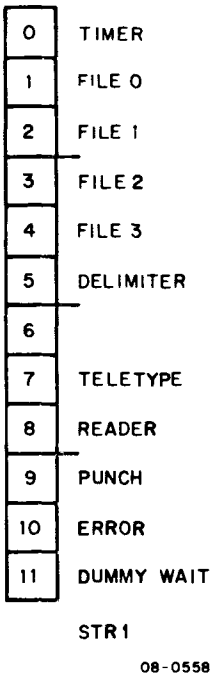


Figure 8-10b STR1

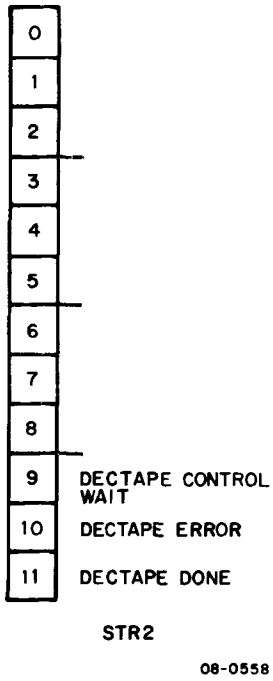


Figure 8-10c STR2

Within Job Status Block 2 are four registers which correspond to the four possible internal files. If a register is zero, no file is open on that internal file. When the file is opened, a file control block is set up and a pointer to it is put in Job Status Block 2. At the same time, the first 8-word File Retrieval Information Block for that block is fetched from the UFD and is set up in another block of free core. Referred to as the file window, this retrieval block is used to calculate addresses far file reads and writes. If a part of the file is being accessed which does not correspond to this window, FIP is called to move the window to the appropriate area. Word 1 of the control block remembers which retrieval information block is in the retrieval window.

When a user program executes an RFILE or WFILE, the transfer parameters (word count and file address) are stored in the file control block. The file address is an address within the logical file. The address of the transfer parameters in the user program is also saved. Then, using the file window, the logical file address is reduced to a physical disk address. A parameter block is set up which contains these physical addresses. A pointer to where in Monitor to go when the transfer is complete is also stored. This block is also linked into the disk queue (DSUTBL).

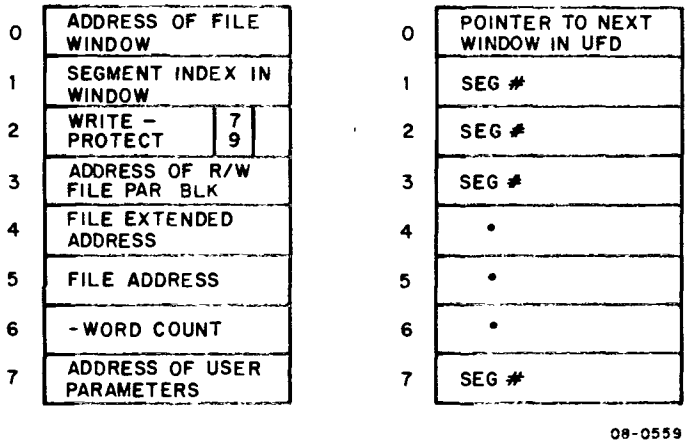


Figure 8-11 File Retrieval Information Block

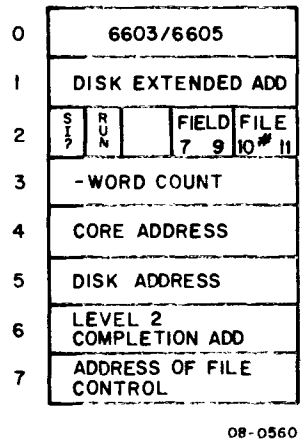


Figure 8-12 Read/Write File Parameter

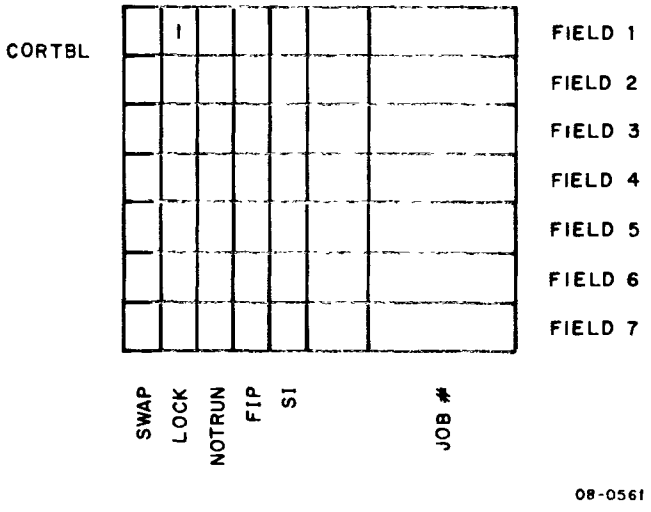


Figure 8-13 CORTBL

CLKTBL is used to execute the STM instruction. It has a one-word entry for each job. If that job is not waiting out on STM, its entry is zero. If it is, the entry contains the number of seconds left to wait (in 2s complement). When the counter goes to zero, the timer flag for that job is set.

The TTYTBL table has a one-word entry for each possible system job. Each entry contains the number of the Teletype associated with that job.

8.3 MONITOR SCHEDULING DATA BASE

DEVTBL, JOBTBL, and their related status blocks maintain some of TSS/8's status information relating to individual jobs. Monitor also maintains some of its own tables. These are used primarily to do scheduling.

CORTBL contains the status of the user fields. It is a seven-word table in Monitor core, each word corresponding to a core field. Within each entry, bits 7 through 11 contain the job number of the job which occupies that field. If the field is empty, a zero is stored there. If the job that occupies a field is not completely there, bit 0 is set to indicate a swap in progress. A job is considered to be in a field from the time it is scheduled to be swapped in until the time it is completely swapped out.

Bit 1 is set if the job in that core field cannot be swapped out. This is the case while a disk transfer is pending for that field. Bit 2 is set if the job in that field has not been run. It cannot be swapped out until it has been run. FIP and SI are called phantoms in the sense that they run in place of a user job. Therefore, when either is running, the calling job number is stored in CORTBL. Bits 3 or 4 is set to remember that it is actually a phantom. Phantoms can run only in field 2. CORTBL has an entry for every core field but field 0, whether it is available or not. At startup time, Monitor's fields and nonexistent fields have their lock bits set to prevent their use.

PRGTBL maintains information on what program each user is running. It has a three-word entry for each possible job. When a user types a LOAD or RUN command, the file name (one to six characters packed in internal format) is stored in PRGTBL. This information is used solely by TSS/8 SYSTAT.

DSUTBL is the disk request queue. It contains a four-word entry for each core field in the system. A 7777 word terminates DSUTBL. Within each four-word entry there is a register for each of the four possible files open to the user currently in that core field. If the entry is zero, there is no file transfer pending for that internal file for that user in that core field. If the entry is nonzero, it is a pointer to a parameter block (the RFILE/WFILE parameter block) which describes the transfer that is to take place. A pointer, DSKPTR, cycles through DSUTBL looking for transfers to be done.

8.4 DISK FILE DATA BASE

For each TSS/8 account number there is a separate disk file library, which contains named files. Controlling this library is the User File Directory which contains, for each file, the file name (and some associated information) and information about where the file may be found. The name is in an 8-word name block; the retrieval information is in one or more 8-word file retrieval information blocks. The UFD itself is stored in disk segments, up to a maximum of seven.

The first 8-word block of the UFD is a dummy block. It contains all zeros except for a pointer to the next block.

The MFD is identical in form to a UFD. The only difference is in the contents of the name block. Where the UFD has six file name characters packed into three words, the MFD has the account number in the first word, then two words of password. Altogether, these three words are the name of the associated UFD.

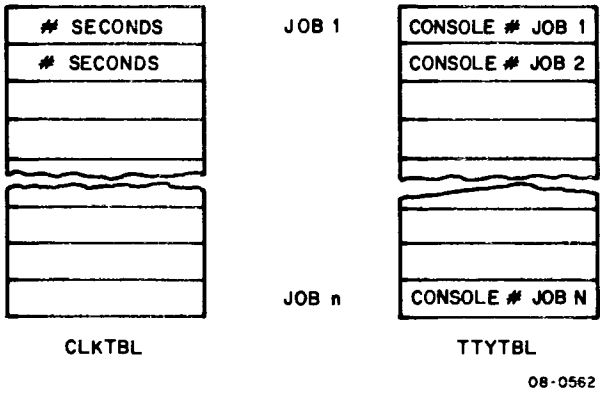


Figure 8-14 PRGTBL

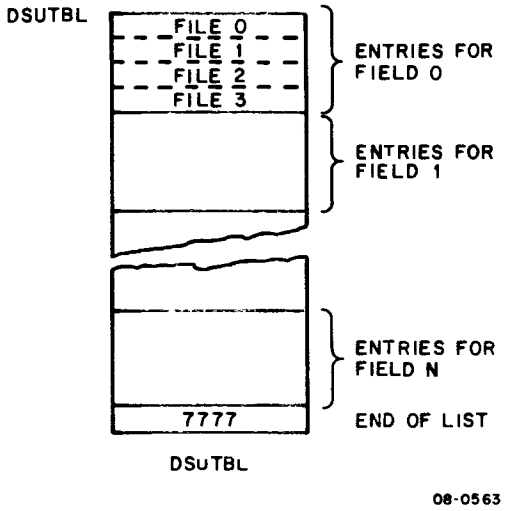


Figure 8-15 DSUTBL

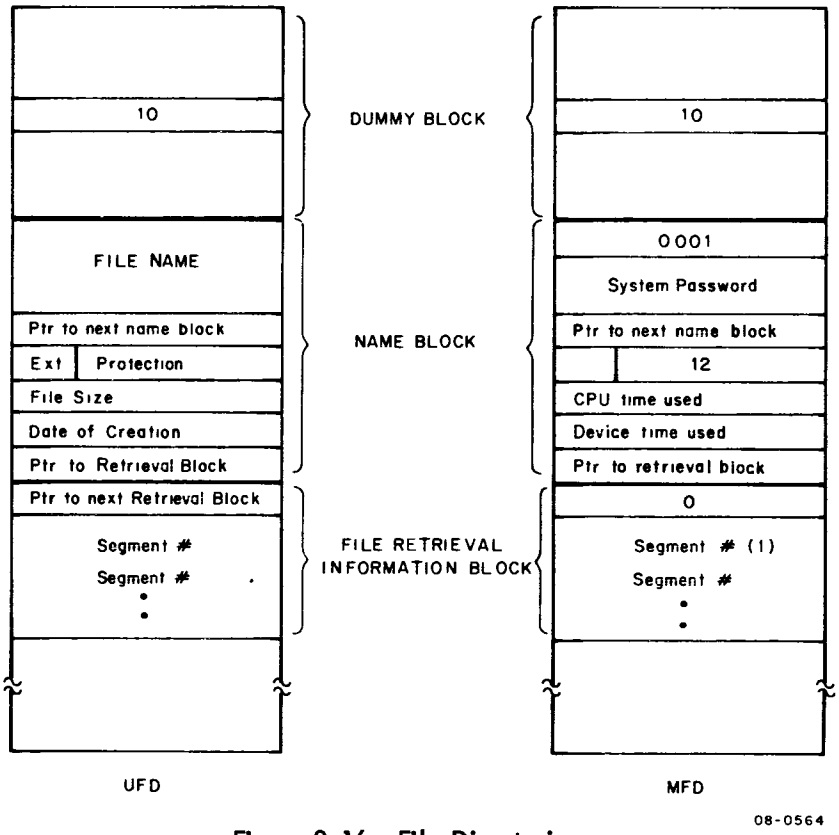


Figure 8-16 File Directories

8.5 FILE PHANTOM DATA BASE

The primary data base of the File Phantom is the directories, the MFD and UFDs. Although they may be accessed as files by a user logged in with the system password, these directories are normally used only by FIP. In addition, to keep track of disk usage, FIP maintains a Storage Allocation Table (SAT). The SAT is a bit map of the disk file space. The 12 bits in each SAT word correspond to 12 disk segments, 1 if the segment is used, 0 if it is available. At refresh time INIT sets all bits which correspond to nonexistent disk to 1s. The SAT is located at the top of FIP's 4K. It is therefore swapped into core with FIP. If the SAT is updated, it is written back to the disk. Below the SAT are two registers, SATCNT and SATBOT which record the number of free disk segments and the place within the SAT where segments are currently being allocated.

FIP also maintains some convenience tables within its own 4K area. These tables allow FIP to get at frequently used information quickly. For example, when a user logs in, the retrieval block, which indicates where his UFD is located, is fetched from the MFD and stored in a table. FIP need not then scan the MFD for this user every time he opens a file.

JOBTAB contains a one-word entry for each possible system job. If no one is logged in for that job, the entry is zero. If there is a user logged in, his account number is stored. (Do not confuse FIP's JOBTAB with resident Monitor's JOBTBL.)

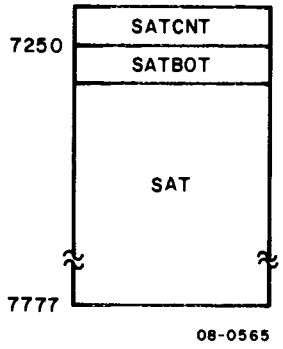


Figure 8-17 Storage Allocation Table

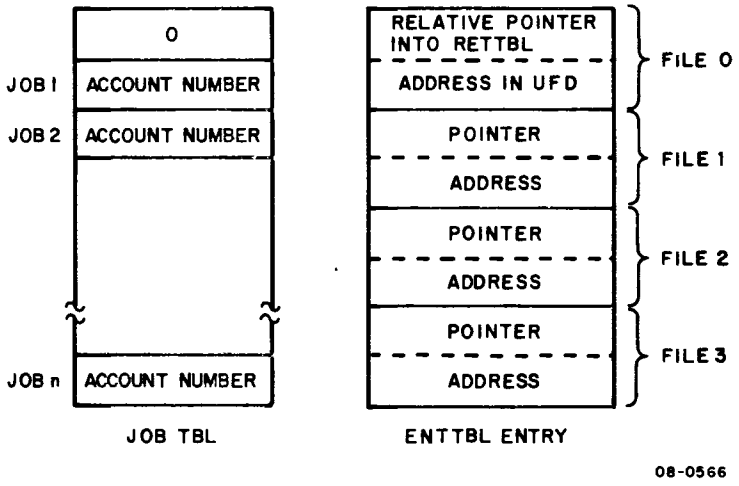


Figure 8-18 FIP Tables

ENTTBL contains an eight-word block for each possible system job. Within these eight words are four two-word entries, one for each possible open file for that job. If the entry is zero, the file is not open. If the file is open, the first word points to the entry in RETTBL for this file. The second word points to the location within the user's UFD where the File Retrieval Information Blocks for this file begin.

UFDTBL and RETTBL work together to maintain retrieval information for all UFDs which are in use within the system. A UFD is in use if one or more users are logged in with that account or if a user

has opened a file from the library of another user. There is only one entry in UFDTBL and RETTBL for each UFD, even if more than one user is using it.

UFDTBL is a table of two-word entries. The first is the account number of the UFD which is open, the second is the number of users who have access to it. (This number is decremented each time a user stops using that UFD. If the count goes to zero, the entry is removed from UFDTBL and RETTBL.) The access count is in 2s complement form.

RETTBL contains the File Retrieval Information Block for the UFD which corresponds to the account number in UFDTBL. There are no pointers between the two tables. Corresponding entries correspond positionally. The number of entries in these tables is at least the number of on-line users. The number of additional entries depends on the amount of file sharing going on. For instance, the library UFD is invariably open to several users.

UFDTBL and RETTBL are initialized to have the system account (#1) open as the first entry with an access count of 1 (actually -1). This allows FIP to get at the MFD while processing a LOGIN request.

All FIP tables except the SAT are cleared at system startup time. SAT is cleared at refresh time.

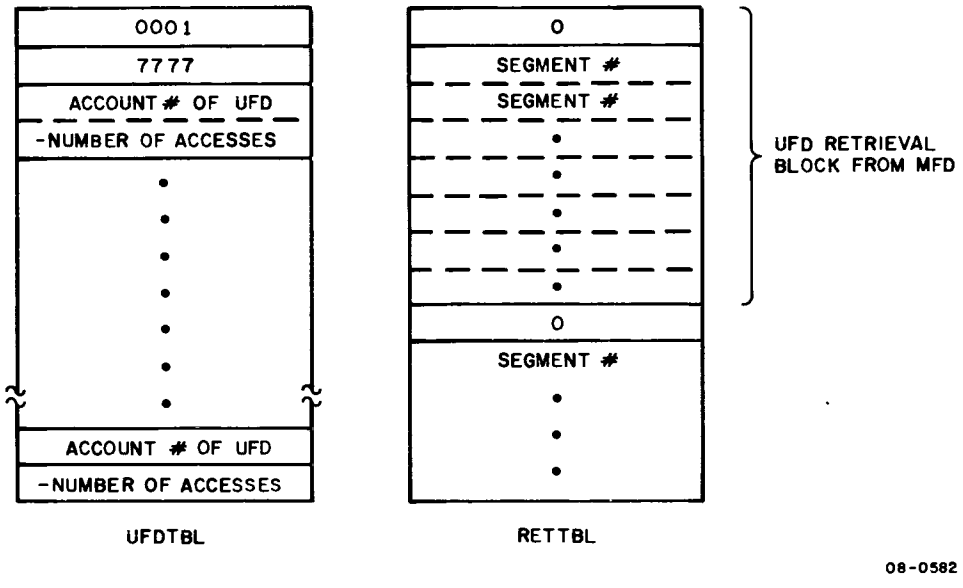


Figure 8-19 UFD Retrieval Data

Appendix A
TSS/8 Character Set

TSS/8 accepts 8-bit ASCII characters only. ASCII is an abbreviation for USA Standard Code for Information Interchange. The acceptable characters and their 6- and 8-bit octal equivalents are listed below.

Character	6-Bit* Octal	8-Bit Octal	Character	6-Bit* Octal	8-Bit Octal
Space	00	240	@	40	300
!	01	241	A	41	301
"	02	242	B	42	302
#	03	243	C	43	303
$	04	244	D	44	304
%	05	245	E	45	305
&	06	246	F	46	306
'	07	247	G	47	307
(	10	250	H	50	310
)	11	251	I	51	311
*	12	252	J	52	312
+	13	253	K	53	313
,	14	254	L	54	314
-	15	255	M	55	315
.	16	256	N	56	316
/	17	257	O	57	317
0	20	260	P	60	320
1	21	261	Q	61	321
2	22	262	R	62	322
3	23	263	S	63	323
4	24	264	T	64	324
5	25	265	U	65	325
6	26	266	V	66	326
7	27	267	W	67	327

* Used to store passwords and filenames only.

Character	6-Bit* Octal	8-Bit Octal	Character	6-Bit* Octal	8-Bit Octal
8	30	270	X	70	330
9	31	271	Y	71	331
:	32	272	Z	72	332
;	33	273	[	73	333
<	34	274	\	74	334
=	35	275	]	75	335
>	36	276	↑	76	336
?	37	277	←	77	337

* Used to store passwords and filenames only.

Appendix B
Building a TSS/8 System from Paper Tape

Load TSS/8 BUILD using the Binary Loader and start at location 0200, as illustrated in Figure 2.4.

```
TSS/8 BUILD--(REVISED 2/15/70)

IS DISK AN RS08? (Y IF RS08. N IF DF32): Y
DOES THE SYTEM INCLUDE DECTAPE? (Y OR N): N

YOU SHOULD HAVE THE FOLLOWING BIN FORMAT PAPER TAPES:

1) SI    ... TSS/8 SYSTEM INTERPRETER
2) FIP   ... TSS/8 FILE PHANTOM
3) XDDT  ... TSS/8 DEBUGGING UTILITY
4) INIT  ... TSS/8  INITIALIZER
5) TS8   ... FIELD 0 RESIDENT MONITOR
6) TS8II .. FIELD 1 RESIDENT MONITOR
7A)PIP   ... PERIPHERAL INTERCHANGE PROGRAM *** IF NO DECTAPE ***
7B)COPY  ... DECTAPE COPY PROGRAM *** IF DECTAPE ***

THE BUILDING PROCESS IS DONE IN FIVE STEPS

1. LOADING MONITOR ONTO THE DISK
2. REFRESHING THE DISK
3. BUILDING UP THE SYSTEM LIBRARY
4. BUILDING UP THE FILE OF VALID PASSWORDS
5. DUMPING THE SYSTEM TO DECTAPE
        (IF DECTAPE IS ON THE SYSTEM.)

ONLY THE FIRST STEP IS DONE UNDER THE CONTROL OF TSS/8 BUILD.
STEP 2 IS DONE UNDER CONTROL OF INIT. STEPS 3 AND 4
ARE DONE WHILE THE TIME-SHARING SYSTEM IS ON-LINE. STEP 5
IS DONE BY STOPPING THE SYSTEM AND RECALLING INIT.

STEP ONE ---

AS EACH TAPE NAME IS TYPED OUT, MOUNT THE CORRESPONDING TAPE IN
THE HIGH SPEED READER AND TYPE CARRIAGE RETURN.

SI:
FIP:
XDDT:
INIT:
TS8:
TS8II:
PIP:
EXPLAIN STEP 2? (Y OR N): Y
REFRESHING THE TSS/8 SYSTEM DELETES ALL FILES,
ACCOUNT NUMBERS, AND DIRECTORIES FROM THE DISK.
```

```
THEN TWO PASSWORDS, THE SYSTEM PASSWORD (ALWAYS
ACCOUNT NUMBER ONE) AND THE LIBRARY PASSWORD (ALWAYS ACCOUNT
NUMBER TWO) ARE DEFINED. THESE PASSWORDS MAY BE ANY COMBINATION
OF 4 CHARACTERS.

THE PROCEDURE IS AS FOLLOWS:
NOTE: ALL USER RESPONSES ARE BRACKETED HERE BY <> FOR CLARITY
        DO NOT INCLUDE < OR > IN THE ACTUAL RESPONSES

LOAD, DUMP, REFRESH, START? <REFRESH>
REFRESH? <YES>
SYSTEM PASSWORD? <MAGI>
LIBRARY PASSWORD? <LIBR>

LOAD, DUMP, REFRESH, START? <START>
LOGIN MESSAGE? <NO>
LOAD EXEC DDT? <NO>
# OF USER FIELDS? <->
MONTH-DAY-YEAR? <12:24:84>
HOUR-DAY? <23:56>

AS PART OF THE INITIALIZATION PROCESS, TSS/8 ASKS HOW MANY
USER FIELDS ARE TO BE USED. FOR NORMAL RUNNING, THIS IS THE NUMBER
OF CORE FIELDS ON THE SYSTEM MINUS TWO (FOR RESIDENT MONITOR)
THUS, FOR A 16K TSS/8, RESPOND WITH THE NUMBER 2

AS SOON AS THE TIME OF DAY HAS BEEN ENTERED, THE SYSTEM IS UP
AND RUNNING. THIS COMPLETES STEP TWO.

EXPLAIN STEP 3? (Y OR N): Y
TSS/8 SYSTEM LIBRARY PROGRAMS ARE DISTRIBUTED ON SPECIAL SAVE
FORMAT PAPER TAPES. SAVE FORMAT TAPES ARE
READ (AND PUNCHED) ONLY BY TSS/8 PIP.

PIP HAS BEEN PRE-LOADED DURING STEP 1. TO USE IT TO LOAD
THE LIBRARY PROGRAMS, LOG IN WITH THE PASSWORD YOU DEFINED
FOR THE LIBRARY (THE ACCOUNT NUMBER IS TWO.) TO START PIP
TYPE:

.<START 0>


 PIP RESPONDS BY TYPING ' INPUT:' RESPOND BY TYPING THE
RETURN KEY. WHEN PIP REQUESTS OUTPUT, TYPE IN THE NAME
OF THE PROGRAM BEING LOADED AND MOUNT IT IN THE HIGH SPEED
READER. WHEN PIP REQUESTS OPTION, TYPE  THE LETTER S TO INDICATE A
SAVE FORMAT TAPE. PIP WILL THEN READ IN THE TAPE AND REQUEST
MORE INPUT. EXAMPLE:

 INPUT:<CR>
OUTPUT:<FOCAL>
OPTION:<S>

 INPUT:
```

```
REPEAT UNTIL ALL DESIRED PROGRAMS HAVE BEEN LOADED. REMEMBER
TO LOAD PIP ITSELF. WHEN DONE, YOU MAY RUN CAT TO VERIFY
THAT ALL PROGRAMS HAVE BEEN PROPERLY LOADED. THEN TYPE  CTRL/B AND S
AND THEN LOGOUT. THIS COMPLETES STEP 3.

NOTE: SAVE FORMAT TAPES INCLUDE A CHECKSUM. IF PIP DETECTS A
CHECKSUM ERROR, IT WILL TYPE 'LOAD ERROR'.

EXPLAIN STEP 4? (Y OR N): Y
PASSWORDS MAY ONLY BE DEFINED BY A USER WHO IS LOGGED IN WITH THE
SYSTEM PASSWORD.  THIS IS THE PASSWORD DEFINED IN STEP 2. LOG IN USING
THIS PASSWORD. (THE ACCOUNT NUMBER IS 1).

THE LIBRARY PROGRAM 'LOGID' IS USED TO DEFINE NEW ACCOUNT
NUMBER/PASSWORDS. IT SHOULD HAVE BEEN LOADED INTO THE LIBRARY DURING
STEP 3. TO USE IT, TYPE

.<R LOGID>

WAIT FOR LOGID TO TYPE AN ASTERISK(*).  TO DEFINE NEW PASSWORDS,
TYPE THE ACCOUNT NUMBER (1 TO 4 OCTAL DIGITS), A SINGLE
SPACE, AND THE PASSWORD (1 TO 4 ALPHANUMERICS), THEN THE RETURN
KEY. WAIT FOR THE NEXT ASTERISK BEFORE ENTERING THE NEXT LINE.
WHEN ALL DESIRED ACCOUNT NUMBERS HAVE BEEN DEFINED, TYPE
CTRL/B AND S AND THEN LOG OUT. THIS COMPLETES STEP 4.

NOTE: IF YOU MAKE A TYPING MISTAKE, TYPE THE RUBOUT KEY. THIS DELETES
        THE LINE BEING ENTERED. TO DELETE AN ALREADY DEFINED
        PASSWORD, TYPE IT IN AGAIN, BUT TERMINATE WITH ALTMODE
        INSTEAD OF THE RETURN KEY. EXAMPLE:

.R LOGID

TSS/8 ACCOUNT MAINTENANCE--

* ACCT #<SPACE>PASSWORD <RETURN TO OPEN/CHANGE, ALT MODE TO CLOSE>

* 10 DEMO
* 277 XYZ
* ↑BS
.LOGOUT

EXPLAIN STEP 5? (Y OR N): N

END OF TSS/8 BUILD

LOAD, DUMP, REFRESH, START? REFRESH
REFRESH? YES
SYSTEM PASSWORD? TSS8
LIBRARY PASSWORD? LBRY

LOAD, DUMP, REFRESH, START? START
LOGIN MESSAGE? NO
LOAD EXEC DDT? NO
# USER FIELDS - 2
MONTH-DAY-YEAR: 6:3:70
HR:MIN - 10:14
```

```
.
TSS/8.21C       JOB 01  K00     10:14:17
YOUR MESSAGE IN THIS SPACE


.START 0


 INPUT:
OUTPUT:FOCAL
OPTION:S

 INPUT:
OUTPUT:BASIC
OPTION:S

 INPUT:
OUTPUT:PALD
OPTION:S

 INPUT:
OUTPUT:EDIT
OPTION:S

 INPUT:
OUTPUT:FORT
OPTION:S

 INPUT:
OUTPUT:FDCOMP
OPTION:S

 INPUT:
OUTPUT:FOSL
OPTION:S

 INPUT:
OUTPUT:FOSSIL
OPTION:S

 INPUT:
OUTPUT:LOGOUT
OPTION:S

 INPUT:
OUTPUT:CAT
OPTION:S

 INPUT:
OUTPUT:PIP
OPTION:S

 INPUT:
OUTPUT:SYSTAT
OPTION:S

 INPUT:
OUTPUT:LOADER
OPTION:S
```

```
 INPUT:
OUTPUT:ODTHI
OPTION:S

 INPUT:
OUTPUT:LOGID
OPTION:S

 INPUT:↑BS
.R CAT
DISK FILES FOR USER     0,2     ON 3-JUN-70

NAME    SIZE    PROT    DATE
FOCAL   16      12      3-JUN-70
BASIC   33      12      3-JUN-70
PALD    16      12      3-JUN-70
EDIT    7       12      3-JUN-70
FORT    7       12      3-JUN-70
FDCOMP  16      12      3-JUN-70
FOSL    7       12      3-JUN-70
FOSSIL  10      12      3-JUN-70
LOGOUT  6       12      3-JUN-70
CAT     5       12      3-JUN-70
PIP     10      12      3-JUN-70
SYSTAT  5       12      3-JUN-70
LOADER  4       12      3-JUN-70
ODTHI   2       12      3-JUN-70
LOGID   2       12      3-JUN-70

TOTAL DISK SEGMENTS: 146

↑BS
.LOGOUT


JOB  1, USER [  0,2] LOGGED OFF K00 AT 10:44:27 ON   3 JUN 70
RUNTIME 00:01:09 (   11. CPU UNITS)
ELAPSED TIME 00:27:18
.
TSS/8.21C  JOB 01  K00    10:46:11

YOUR MESSAGE IN THIS SPACE

.R LOGID


TSS/8 ACCOUNT MAINTENANCE --

* ACC'T # <SPACE> PASSWORD <RETURN TO OPEN/CHANGE, ALT MODE TO CLOSE>

* 1066 HARQ
* 1215 JOHN
* 732 TOUR
* ↑BS
.LOGOUT


JOB  1, USER [  0, 1] LOGGED OFF K00 AT 10:47:58 ON  3 JUN 70
RUNTIME 00:00:00 (   0. CPU UNITS)
ELAPSED TIME 00:01:42
.
```

Appendix C
Building a TSS/8 System from DECtape

```
TSS/8 BUILD--(REVISED 2/15/70)

IS DISK AN RS08? (Y IF RS08. N IF DF32): Y
DOES THE SYTEM INCLUDE DECTAPE? (Y OR N): Y

YOU SHOULD HAVE THE FOLLOWING BIN FORMAT PAPER TAPES:

1) SI    ... TSS/8 SYSTEM INTERPRETER
2) FIP   ... TSS/8 FILE PHANTOM
3) XDDT  ... TSS/8 DEBUGGING UTILITY
4) INIT  ... TSS/8 INITIALIZER
5) TS8   ... FIELD 0 RESIDENT MONITOR
6) TS8II. FIELD 1 RESIDENT MONITOR
7A)PIP   ... PERIPHERAL INTERCHANGE PROGRAM *** IF NO DECTAPE ***
7B)COPY  ... DECTAPE COPY PROGRAM *** IF DECTAPE ***

THE BUILDING PROCESS IS DONE IN FIVE STEPS

1. LOADING MONITOR ONTO THE DISK
2. REFRESHING THE DISK
3. BUILDING UP THE SYSTEM LIBRARY
4. BUILDING UP THE FILE OF VALID PASSWORDS
5. DUMPING THE SYSTEM TO DECTAPE
        (IF DECTAPE IS ON THE SYSTEM.)

ONLY THE FIRST STEP IS DONE UNDER THE CONTROL OF TSS/8 BUILD.
STEP 2 IS DONE UNDER CONTROL OF INIT. STEPS 3 AND 4
ARE DONE WHILE THE TIME-SHARING SYSTEM IS ON-LINE. STEP 5
IS DONE BY STOPPING THE SYSTEM AND RECALLING INIT.

STEP ONE ---

AS EACH TAPE NAME IS TYPED OUT, MOUNT THE CORRESPONDING TAPE IN
THE HIGH SPEED READER AND TYPE CARRAIGE RETURN.

SI:
FIP:
XDDT:
INIT:
TS8:
TS8II:
COPY:

EXPLAIN STEP 2? (Y OR N): Y
REFRESHING THE TSS/8 SYSTEM DELETES ALL FILES,
ACCOUNT NUMBERS, AND DIRECTORIES FROM THE DISK.
```

```
THEN TWO PASSWORDS, THE SYSTEM PASSWORD(ALWAYS
ACCOUNT NUMBER ONE) AND THE LIBRARY PASSWORD (ALWAYS ACCOUNT
NUMBER TWO) ARE DEFINED. THESE PASSWORDS MAY BE ANY COMBINATION
OF 4 CHARACTERS.
THE PROCEDURE IS AS FOLLOWS:
NOTE: ALL USER RESPONSES ARE BRACKETED HERE BY <> FOR CLARITY
        DO NOT INCLUDE < OR > IN THE ACTUAL RESPONSES

LOAD, DUMP, REFRESH, START? <REFRESH>
REFRESH? <YES>
SYSTEM PASSWORD? <MAGI>
LIBRARY PASSWORD? <LIBR>

LOAD, DUMP, REFRESH, START? <START>
LOGIN MESSAGE? <NO>
LOAD EXEC DDT? <NO>
# OF USER FIELDS? <->
MONTH-DAY-YEAR? <12:24:84>
HOUR-DAY? <23:56>

AS PART OF THE INITIALIZATION PROCESS, TSS/8 ASKS HOW MANY
USER FIELDS ARE TO BE USED. FOR NORMAL RNING, THIS IS THE NUMBER
OF CORE FIELDS ON THE SYSTEM MINUS TWO (FOR RESIDENT MONITOR)
THUS, FOR A 16K TSS/8, RESPOND WITH THE NUMBER 2

AS SOON AS THE TIME OF DAY HAS BEEN ENTERED, THE SYSTEM IS UP
AND RUNNING. THIS COMPLETES STEP TWO.

EXPLAIN STEP 3? (Y OR N): Y
TSS/8 SYSTEM LIBRARY PROGRAMS ARE DISTRIBUTED ON A SYSTEM LIBRARY
DECTAPE. THEY ARE LOADED INTO THE LIBRARY BY THE PROGRAM 'COPY'

COPY HAS BEEN PRE-LOADED DURING STEP 1. TO USE IT, LOG IN WITH THE
PASSWORD YOU DEFINED FOR THE LIBRARY (THE ACCOUNT NUMBER IS 2.)
MOUNT THE LIBRARY DECTAPE ON UNIT ZERO.
TO START COPY, TYPE:

.<START 0>

COPY RESPONDS BY TYPING 'OPTION-'. TYPE <LIST>. COPY THEN
ASKS FOR 'DEVICE-'. TYPE <D0>. COPY WILL TYPE OUT THE NAMES
OF THE FILES ON THE TAPE. THE FIRST FILES ON THE TAPE, THE ONES WITH
A FILE EXTENSION OF '.SAV' ARE THE SYSTEM LIBRARY PROGRAMS.
(BASIC.SAV, FOCAL.SAV, ETC. ) THESE ARE THE FILES TO BE LOADED.
(YOU MAY ALSO WANT TO LOAD SOME OF THE OTHER DEMONSTRATION PROGRAMS
ON THE TAPE.)

WHEN COPY AGAIN TYPES 'OPTION-', TYPE <COPY>. WHEN IT
REQUESTS 'INPUT-', TYPE <D0:NAME> WHERE NAME IS THE
NAME OF ONE OF THE LIBRARY PROGRAMS.    WHEN COPY REQUESTS
'OUTPUT-', TYPE THIS NAME AGAIN. THE FILE WILL THEN
BE LOADED AND COPY WILL AGAIN TYPE 'OPTION-'.-EXAMPLE:

OPTION-<COPY>
INPUT-<D0:BASIC>
OUTPUT-<BASIC>

OPTION-

REPEAT UNTIL ALL PROGRAMS HAVE BEEN LOADED. REMEMBER TO LOAD
COPY ITSELF. WHEN DONE, TYPE CTRL/B AND S. THEN TYPE
<LOGOUT>
THIS COMPLETES STEP 3
```

```
EXPLAIN STEP 4? (Y OR N): Y
PASSWORDS MAY ONLY BE DEFINED BY A USER WHO IS LOGGED IN WITH THE
SYSTEM PASSWORD.  THIS IS THE PASSWORD DEFINED IN STEP 2. LOG IN USING
THIS PASSWORD. (THE ACCOUNT NUMBER IS 1).

THE LIBRARY PROGRAM 'LOGID' IS USED TO DEFINE NEW ACCOUNT
NUMBER/PASSWORDS. IT SHOULD HAVE BEEN LOADED INTO THE LIBRARY DURING
STEP 3. TO USE IT, TYPE

.<R LOGID>

WAIT FOR LOGID TO TYPE AN ASTERISK(*).  TO DEFINE NEW PASSWORDS,
TYPE THE ACCOUNT NUMBER (1 TO 4 OCTAL DIGITS), A SINGLE
SPACE, AND THE PASSWORD (1 TO 4 ALPHANUMERICS), THEN THE RETURN
KEY. WAIT FOR THE NEXT ASTERISK BEFORE ENTERING THE NEXT LINE.
WHEN ALL DESIRED ACCOUNT NUMBERS HAVE BEEN DEFINED, TYPE
CTRL/B AND S AND THEN LOG OUT. THIS COMPLETES STEP 4.

NOTE: IF YOU MAKE A TYPING MISTAKE, TYPE THE RUBOUT KEY. THIS DELETES
        THE LINE BEING ENTERED. TO DELETE AN ALREADY DEFINED
        PASSWORD, TYPE IT IN AGAIN, BUT TERMINATE WITH ALTMODE
        INSTEAD OF THE RETURN KEY. EXAMPLE:


.R LOGID

TSS/8 ACCOUNT MAINTENANCE--

* ACCT #<SPACE>PASSWORD <RETURN TO OPEN/CHANGE, ALT MODE TO CLOSE>

* 10 DEMO
* 277 XYZ
* ↑BS
.LOGOUT

EXPLAIN STEP 5? (Y OR N): Y
TO DUMP THE WHOLE SYSTEM TO DECTAPE(S), STOP THE SYSTEM (MAKE
SURE THAT ALL USERS ARE LOGGED OUT.) LOAD ADDRESS 4200
(FIELD ZERO) AND START. STARTING AT 4200 BOOTS A COPY OF THE INIT-
IALIZER INTO A HIGH CORE FIELD. IT RESPONDS BY TYPING
'LOAD, DUMP, REFRESH, START?'

MOUNT DECTAPES ON UNITS ONE AND TWO AND WRITE ENABLE. (IF DF32'S
ARE USED, ONLY ONE TAPE (ON UNIT 1) IS NEEDED. NOW RESPOND
'DUMP'. THE ENTIRE STATE OF THE SYSTEM WILL BE SAVED ON DECTAPE(S)
INIT WILL AGAIN TYPE 'LOAD, DUMP, REFRESH, START? '

END OF TSS/8 BUILD --
LOAD, DUMP, REFRESH, START? REFRESH
REFRESH? YES
SYSTEM PASSWORD? TSS8
LIBRARY PASSWORD? LBRY

LOAD, DUMP, REFRESH, START? START
LOGIN MESSAGE? NO
LOAD EXEC DDT? NO
# USER FIELDS - 2
MONTH-DAY-YEAR: 6:3:70
HR:MIN 12:03
```

```
.
TSS/8.21C       JOB 01    K00     12:03:14
YOUR MESSAGE IN THIS SPACE
.START 0


OPTION- LIST
DEVICE- D0


642. FREE BLOCKS

NAME       SIZE    DATE
BASIC .SAV  66   2-MAR-70
FOCAL .SAV  32  12-MAR-70
COPY  .SAV  20   3-MAR-70
EDIT  .SAV  14   2-MAR-70
PIP   .SAV  20   2-MAR-70
SYSTAT.SAV  10   2-MAR-70
LOGOUT.SAV  12   2-MAR-70
CAT   .SAV  10   2-MAR-70
LOADER.SAV   8   2-MAR-70
FORT  .SAV  12   2-MAR-70
FOSL  .SAV  14   2-MAR-70
FDCOMP.SAV  32   2-MAR-70
FOSSIL.SAV  20   2-MAR-70
PALD  .SAV  32   2-MAR-70
ODTHI .SAV   4   2-MAR-70
CATLOG.BAS  26   3-MAR-70
LOGID .SAV   6   2-MAR-70
PLOT  .FCL   2   3-MAR-70
BANDIT.BAS  14   3-MAR-70
BLKJAC.BAS  32   3-MAR-70
BUNNY .ASC  56   3-MAR-70
CRAPS .BAS  12   3-MAR-70
EVEN  .BAS  16   3-MAR-70
FILMS .BAS  16   3-MAR-70
FSPIEL.BAS   8   3-MAR-70
FTBALL.BAS  28   3-MAR-70
GOLF  .BAS  24   3-MAR-70
HOSSR .BAS  20   3-MAR-70
NELSON.BAS   4   3-MAR-70
NIM   .BAS  22   3-MAR-70
ROULET.BAS  26   3-MAR-70
SPORTS.BAS  22   3-MAR-70
TICTAC.BAS  10   3-MAR-70
CONVER.BAS   6   3-MAR-70
FACT  .BAS   6   3-MAR-70
FACTAL.BAS  30   3-MAR-70
FIBO  .BAS   6   3-MAR-70
INVERT.BAS  24   3-MAR-70
MAGIC .BAS   6   3-MAR-70
PRIME .BAS   8   3-MAR-70
INTER .BAS   4   3-MAR-70
PEACE .BAS  14   3-MAR-70
PEACE2.BAS  10   3-MAR-70
SNOOPY.BAS  12   3-MAR-70
WDGAME.ASC  10   3-MAR-70
DATA  .ASC   8   3-MAR-70
MATRIX.ASC   6   3-MAR-70
TYPE  .ASC  10   3-MAR-70
HAMURA.FCL  10   3-MAR-70
```

```
OPTION- COPY
INPUT- D0:BASIC
OUTPUT- BASIC

OPTION- COPY
INPUT- D0:FOCAL
OUTPUT- FOCAL

OPTION- COPY
INPUT- D0:COPY
OUTPUT- COPY

OPTION- COPY
INPUT- D0:EDIT
OUTPUT- EDIT

OPTION- COPY
INPUT- D0:PIP
OUTPUT- PIP

OPTION- COPY
INPUT- D0:SYSTAT
OUTPUT- SYSTAT

OPTION- COPY
INPUT- D0:LOGOUT
OUTPUT- LOGOUT

OPTION- COPY
INPUT- D0:CAT
OUTPUT- CAT

OPTION- COPY
INPUT- D0:LOADER
OUTPUT- LOADER

OPTION- COPY
INPUT- D0:FORT
OUTPUT- FORT
OPTION- COPY
INPUT- D0:FOSL
OUTPUT- FOSL
OPTION- COPY
INPUT- D0:FDCOMP
OUTPUT- FDCOMP

OPTION- COPY
INPUT- D0:FOSSIL
OUTPUT- FOSSIL

OPTION- COPY
INPUT- D0:PALD
OUTPUT- PALD

OPTION- COPY
INPUT- D0:ODTHI
OUTPUT- ODTHI

OPTION- COPY
INPUT- D0:LOGID
OUTPUT- LOGID

OPTION- LIST
DEVICE-
```

```
DISK FILES FOR USER  0,2  ON  3-JUN-70.

NAME       SIZE  PROT    DATE
BASIC .SAV   33    12   3-JUN-70
FOCAL .SAV   16    12   3-JUN-70
COPY  .SAV   10    12   3-JUN-70
EDIT  .SAV    7    12   3-JUN-70
PIP   .SAV   10    12   3-JUN-70
SYSTAT.SAV    5    12   3-JUN-70
LOGOUT.SAV    6    12   3-JUN-70
CAT   .SAV    5    12   3-JUN-70
LOADER.SAV    4    12   3-JUN-70
FORT  .SAV    6    12   3-JUN-70
FOSL  .SAV    7    12   3-JUN-70
FDCOMP.SAV   16    12   3-JUN-70
FOSSIL.SAV   10    12   3-JUN-70
PALD  .SAV   16    12   3-JUN-70
ODTHI .SAV    2    12   3-JUN-70
LOGID .SAV    3    12   3-JUN-70

TOTAL DISK SEGMENTS: 156
OPTION- EXIT
↑BS
.

.LOGOUT


JOB     1, USER [ 0, 2] LOGGED OFF K00 AT 12:17:31 ON   3 JUN 70
RUNTIME 00:00:20 (       3. CPU UNITS)
ELAPSED TIME 00:14:46

.
TSS/8.21C        JOB 01  K00      12:17:56

YOUR MESSAGE IN THIS SPACE

.R LOGID


TSS/8 ACCOUNT MAINTENANCE --

* ACC'T # <SPACE> PASSWORD <RETURN TO OPEN/CHANGE, ALT MODE TO CLOSE>

* 1215 JOHN
* 1066 HARO
* 1000 OTTO
* ↑RS
.LOGOUT


JOB     1, USER [ 0, 1] LOGGED OFF K00 AT 12:18:48 ON   3 JUN 70
RUNTIME 00:00:00 (       0. CPU UNITS)
ELAPSED TIME 00:01:33
.
LOAD, DUMP, REFRESH, START? DUMP

LOAD, DUMP, REFRESH, START?
```

Appendix D
TSS/8 Hardware Configurations

	Simultaneous Users						
	4	8		16			24
PDP-8/I-D and ASR33 (negative bus)	1	1	1	1	1	1	1
MC8/IA Memory Control +4K	1	1	1	1	1	1	1
MM8/IA 4K Memory	1						
MM8/IB 8K Memory		1	1	1	2	3	3
RF08 Disk Control	1	1	1	1	1	1	1
RS08 Disk	1	1	2	2	3	4	4
KT8/I Time-Share Modification	1	1	1	1	1	1	1
PR/8I Paper Tape Reader	1	1					
PC/8I Reader-Punch			1	1	1	1	1
KW8/IA Clock	1						
PT08B Teletype Interface	1						
PT08C Dual Interface	1						
PT08F Modem Adapter	4						
DL8/I Data Line Adapter		1	1	1	1	1	1
DC08A Line Multiplexer		1	1	1	1	1	1
M750 Dual Interface		4	4	8	8	8	12
DC08B Adapter Panel		1	1	1	1	1	1
BC01C-25 Modem Adapter		8	16	16			
DW08B Positive Bus		1	1	1	1	1	1
689-AG Modem Interface						1	1
689-LM Modem Adapter						16	24

	Simultaneous Users (Cont)						
	4	8		16			24
DM01 Multiplexer			1		1	1	1
TC01 DECtape Control			1		1	1	1
TU55 DECtape Transport			2		2	4	6
H961A Cabinet	1	1	1	2	2	3	3
TSS/8 Software Set	1	1	1	1	1	1	1

Appendix E
Required Modifications

The PDP-8/I or PDP-8 computer must have the following KT08/I timesharing modifications to be used in a TSS/8 system.

a. USER FLAG REGISTER (UF) - UF is a 1-bit register that specifies Monitor (UF=0) or user (UF=1) mode. In Monitor mode all instructions are legal; in user mode HALT, OSR, and IOT instructions are trapped via the program interrupt system. UF is cleared by the LOAD ADDress switch.

b. SAVE FIELD REGISTER EXTENSION (SF6) - When a program interrupt occurs, this bit is cleared and then loaded from the UF.

c. USER BUFFER REGISTER (UB) - The UB serves as a 1-bit input buffer for the UF. All transfers into the UF are made through the UB, except transfers from the computer console switches. The Change User Flag (CUF) instruction loads the UB with the value contained in the Memory Buffer (MB). The Restore Memory Field (RMF) instruction transfers the contents of SF6 into the UB to restore the UF to the condition that existed prior to a program interrupt. The UF is cleared by the LOAD ADDress switch.

The following machine instructions have been changed to operate as indicated.

a. Name and Mnemonic: CHANGE USER FLAG (CUF)

Octal Codes: 6264 and 6274
Execution Time: 1.5 μs
Operation: 6264 sets the UF to 0; 6274 sets the UF to 1. The next JMP or JMS instruction causes the appropriate mode to be entered.
Symbol: MB8→UB

b. Name and Mnemonic: SKIP ON USER IOT (SKIOT)

Octal Code: 6254
Execution Time: 1.5 μs
Operation: HALT, OSR, and IOTs when executed with UF =1, sets the user IOT (UIOT) flag and, if the program interrupt is enabled, causes an interrupt.

c. Name and Mnemonic: CLEAR USER IOT (CIOT)

Octal Code: 6204
Execution Time: 1.5 μs
Operation: Clears the user IOT (UIOT) flag.
Symbol: 0 → UIOT

d. Name and Mnemonic: READ INTERRUPT BUFFER (RIB)

Octal Code: 6234
Execution Time: 1.5 μs
Operation: The contents of the Instruction Field, Data Field, and User Flag held in the Save Field Register during a program interrupt are transferred into bits 6 through 8, 9 through 11, and 5, respectively.

Symbols: SF0-2 → AC6-8, SF3-5 → AC9-11, SF6 → AC5

e. Name and Mnemonic: RESTORE MEMORY FIELD (RMF)

Octal Code: 6244
Execution Time: 1.5 μs
Operation: This instruction is used upon exit from the program interrupt subroutine in another field. The Instruction Field, Data Field, and User Flag that were interrupted by the subroutine are restored by transferring the contents of the Save Field Register into the Instruction Buffer and Data Field Registers and the User Buffer.
Symbols: SF0-2 → IB, SF3-5 → DF, SF6 → UB

INDEX

U

W

X

READER'S COMMENTS

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 manual.

Please comment on this manual's completeness, accuracy, organization, usability, and readability.

Did you find errors in this manual?

How can this manual be improved?

Other comments?

Please describe your position.

Name _____ Organization _____

Street _____ Department _____

City _____ State _____ Zip or Country _____

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

HOW TO OBTAIN SOFTWARE INFORMATION

Announcements of new and revised software, as well as programming notes, software problems, and documentation corrections are published by Software Information Service in the following newsletters:

Digital Software News for the PDP-8 Family
Digital Software News for the PDP-9/15 Family

These newsletters contain information to update the cumulative

Software Performance Summary for the PDP-8 Family
Software Performance Summary for the PDP-9/15 Family

The appropriate edition of the Software Performance Summary is included in each basic software kit for new customers. Additional copies may be requested without charge.

Any questions or problems on the articles contained in these publications or concerning the use of Digital's software should be reported to the Software Specialist or Sales Engineer at the nearest Digital office.

New and revised software and manuals, and current issues of the Software Performance Summary are available from the Program Library. To place an order, please contact your local Digital office or write to:

Program Library
Digital Equipment Corporation
146 Main Street, Bldg. 1-2
Maynard, Massachusetts 01754

When ordering, include the code number and a brief description of the program or manual requested.

Digital Equipment Computer Users Society (DECUS) maintains a user library and publishes a catalog of available programs as well as the DECUSCOPE magazine for its members and non-members who request it. For further information, please write to:

DECUS
Digital Equipment Corporation
146 Main Street
Maynard, Massachusetts 01754

Please complete the return postcard below if you would like to receive Digital's newsletters.

Send ☐ Digital Software News for the PDP-8 Family, or
☐ Digital Software News for the PDP-9/15 Family

To Name ____________________

Company Name ____________________

Address ____________________

(zip code)

My computer is a ☐ PDP-8I ☐ PDP-12 ☐ PDP-9
☐ PDP-8L ☐ LINC-8 ☐ PDP-15
☐ PDP-8S ☐ Other ____________

HOW TO OBTAIN SOFTWARE INFORMATION

Announcements of new and revised software, as well as programming notes, software problems, and documentation corrections are published by Software Information Service in the following newsletters:

Digital Software News for the PDP-8 Family
Digital Software News for the PDP-9/15 Family

These newsletters contain information to update the cumulative

Software Performance Summary for the PDP-8 Family
Software Performance Summary for the PDP-9/15 Family

The appropriate edition of the Software Performance Summary is included in each basic software kit for new customers. Additional copies may be requested without charge.

Any questions or problems on the articles contained in these publications or concerning the use of Digital's software should be reported to the Software Specialist or Sales Engineer at the nearest Digital office.

New and revised software and manuals, and current issues of the Software Performance Summary are available from the Program Library. To place an order, please contact your local Digital office or write to:

Program Library
Digital Equipment Corporation
146 Main Street, Bldg. 1-2
Maynard, Massachusetts 01754

When ordering, include the code number and a brief description of the program or manual requested.

Digital Equipment Computer Users Society (DECUS) maintains a user library and publishes a catalog of available programs as well as the DECUSCOPE magazine for its members and non-members who request it. For further information, please write to:

DECUS
Digital Equipment Corporation
146 Main Street
Maynard, Massachusetts 01754

Please complete the return postcard below if you would like to receive Digital's newsletters.

Send ☐ Digital Software News for the PDP-8 Family, or
☐ Digital Software News for the PDP-9/15 Family

To Name ______________________________

Company Name ______________________________

Address ______________________________

(zip code)

My computer is a ☐ PDP-8I ☐ PDP-12 ☐ PDP-9
☐ PDP-8L ☐ LINC-8 ☐ PDP-15
☐ PDP-8S ☐ Other ____________