System Manager's Guide

for

PDP-8E

Dec System

TSS 8.24 Monitor

CHAPTER 1

INTRODUCTION TO TIMESHARING AND EDUSYSTEM 50

1.1 INTRODUCTION TO TIMESHARING CONCEPTS

In a typical programming environment, a user may desire to have a computer edit, test, debug and execute a program. However, there are frequently long periods of time when the computer waits for input from the programmer. At these times, it is desirable to find something else for the computer to do. One solution is called "timesharing". Typically, a number of users each has a terminal connected to a computer, and as each user requires computer time, the processor assigns computer time to each user. Thus, by proper allocation of processor time, each user, in effect, has all the computer time necessary for a particular program. Users do not have to feel rushed or be concerned about others who may desire to use the computer.

EduSystem 50 is such a timesharing system. From the user's viewpoint, each user on an EduSystem system has a terminal and 4K of core for his exclusive use. In addition, disk storage may be used on a first-come/first-serve basis. Any of the several peripherals connected to EduSystem 50 may be reserved by a user exclusively until completion of the program. The EduSystem 50 Monitor is the set of programs which allows all of this to happen.

The EduSystem 50 Monitor reserves fields 0 and 1 for its exclusive use; the rest of core is given to users as they need it. Since there may not be enough core for every user to simultaneously occupy 4K, each user has a 4K section of disk reserved for his program. When there is not enough room in core, the entire 4K is written to disk. Later, when enough core is available, the 4K is read from the disk back into core again. This process is called swapping, and the area on disk reserved for it is called the swap area.

[redacted]

[redacted] 0748

[illegible]

To prevent a user from interfering with the operation of others, several steps are taken:

a) An OSR is not allowed; this prevents disputes over the switch register setting.

b) The user cannot perform an HLT.

c) The program must be executed with the data field set equal to the instruction field; no change is allowed in either. In fact, the user may not do any kind of IOT.

In reality, any of the above instructions may be contained in a user's program. They are trapped by the hardware, which does not allow them to function. Instead, the time-share hardware option raises a flag and causes an interrupt. The Monitor must then determine the cause of the interrupt. If, for example, the user does a KRB, the AC will not be altered. Instead, the KRB causes an interrupt into Monitor code, which looks back to see what causes the interrupt. When is sees the KRB, it takes the necessary steps to simulate the KRB. It then returns to the user's program. Thus, there are 512 different IOTs the user can execute; the Monitor could be modified to perform almost any desired function for them. These trapped IOTs, and the trapped HLTs and OSRs are sometimes referred to an Unimplemented User Operations (UUOs).

The time-share hardware option makes this possible. The processor can be in one of two modes: executive mode and user mode. In executive mode, the processor operates like a standard PDP-8. However, when the processor is in user mode, the instructions discussed above cause an interrupt, and return the processor to executive mode.

NOTE

Instructions microcoded with HLT or OSR will complete their functions normally before the interrupt occurs.

To begin using the system the user must be identified at the terminal. Each user has an account number, under which disk files are stored. The Monitor also keeps track of system usage by account number. To protect against misuse of the system, the user must give an account number and a password to the system: this process is called a LOGIN.

Once logged in, the user may type various commands to the Monitor, execute a program (either original or from the library), or type characters to be input to the program. When finished, the user logs out enabling others to use the terminal.

1.2 EDUSYSTEM 50 HARDWARE CONFIGURATIONS

A minimum configuration for EduSystem 50 includes:

a) PDP-8, 8I, or 8E, with at least 12K memory (16K memory is better) and the time-share option. (All are referred to in the following text as PDP-8.)

b) RF08 with at least one RS08 (a DF32 with at least two platters is acceptable, however, this is not recommended because of the limited storage area and slow speed of the device).

c) Multi-terminal capability - one or more KL8Es or PT08s or a DC08A.

All configurations, except PDP-8Is with a DC08A, require a real-time clock.

Optional hardware supported:

a) Up to 32K memory.

b) DC08A - may be used only on a PDP-8I and may be used in addition to PT08s.

c) 689AG modem controller - for use with DC08A only.

d) EAE- all instructions of any standard EAE are supported with the exception of the traditional PDP-8 step counter; which is not saved or restored.

e) High-Speed Reader - a paper tape reader is required to build EduSystem. A low-speed reader may be used, however, the build procedure will be very time consuming.

f) High-Speed Punch.

g) Line Printer - LP08/LE8 or LS08/LS8E.

h) DECtape - TC01 or TC08 and up to eight drives (TD8E. DECtape is not allowed.)

i) Up to four disks (three additional RS08s).

j) Card Reader.

k) RK8E (up to four drives).

SOFTWARE

The following sections present an introduction to the comp-t programs of EduSystem 50.

1 INIT

INIT is the initializer program. It is the job of INIT to:

a) Build a new system on the system disk from paper tapes.

b) Initialize a file structure on the disk for program storage.

c) Allow the user to make patches to the system. This can be useful if a particular system feature must be altered.

d) Allow the user to transfer the entire contents of the disk to DECtape. This is called a "dump".

e) Allow the user to load the entire disk from DECtape. This allows all saved programs, account numbers, etc., to be restored to their condition when a dump was last taken.

f) Allow the user to start the system so that it is in its operating state. A start must set up fields 0 and 1 to the proper initial conditions, start the real time clock, and transfer control to the monitor.

1.3.2 SI

SI is the System Interpreter. Whenever a user types a command at the console, SI analyzes the command. SI is stored on the disk and is read into field 2 when it is needed.

1.3.3 FIP

FIP is the File Phantom. Whenever certain IOTs (see Appendix C) are executed, FIP is called in to handle them. FIP handles disk file storage and assigns devices to users when they request them. FIP is also stored on the disk and, when it is needed, it is read into field 2.

1.3.4 TS8 and TS8II

TS8 and TS8II are the two sections of the resident Monitor which are resident (always present) in fields 0 and 1, respectively. The resident code is responsible for:

a) Scheduling - The scheduler decides who uses the computer and when. If the user types a command, the scheduler brings in SI. If more than one user wants a program to run, the scheduler distributes time to the users. If there is insufficient core for everyone to use at once, the scheduler swaps users to and from the disk.

b) UUO Handling - When the processor is in user mode, remembe that if a user executes any type of IOT, an HLT, or an OSR. The hardware does not execute the instruction, but causes an interrupt. The IOT, HLT, or OSR is called a UUO in thi case. The Monitor has the job of simulating the function of the UUO, and possibly returning control to the user. By definition, resident UUOs are those which are handled b the code residing in fields 0 and 1. Non-resident UUOs must be handled by calling in FIP.

c) Interrupt Handling - All I/O Interrupts must be isolated and handled by the Monitor.

d) Keeping Time - Clock interrupts are counted and at certain intervals certain tasks must be performed.

5 System Programs

Th re are several programs run by a user to perform certain enient operations. For example:

a) PUTR - Used to transfer information from any device to another device. This is the new program which replaces the programs PIP and COPY which were operated under the previous version of the Monitor.

b) BASIC - A major interactive compiler language.

The list of programs is extensive and is covered in more detail r.

NUMBERS AND TERMINOLOGY

EduSystem 50 associates a number with each terminal. The inal with device codes 03/04 is called, variously K00, line 0, inal 0, or console 0. In order to accomplish anything, a user terminal must "log in". To do this requires a 1 to 4 digit o number", and a 0 to 4 character "password". When the user mpts to log in, the account number and password are given. If are valid and if the system permits logging in, the system gns a "job number". When the user is through, a LOGOUT or command is required. At this time his account is charged the amount of time used. In this way system usage can be itored.

Sometimes account numbers are split into two, 2-digit numbers separated by a comma. The first two digits are called the ject number, and the last two digits are called the programmer er. Account 1, 2 is the same as account 0102.

There are three permanently defined account numbers. Account 1 ongs to the system manager. Anyone logged in under account 1 has certain priveleges no other person has, such as defining other account numbers and their passwords. However, doing certain things under account 1 could be detrimental to the system. Account 2 belongs to the system librarian. The common programs are stored under account 2, and any user can access them there. Account 3 belongs to the system operator who has certain privileges which are not as detrimental as those granted to account 1.

1.5 SYSTEM CONVENTIONS

The disk is divided into tracks. One track is defined as 4K (4096 words) of disk storage. (Two revolutions of an RF08, four revolutions of a DF32D, eight revolutions of a DF32.) Thus, track 0 refers to the first 4096 words of storage on the RF08 or DF32. One segment is defined as two pages, 400 (octal) or 256 (decimal) words of disk storage. All disk files are measured in segments.

Unless otherwise noted, all commands and responses typed by the user should be terminated with a carriage return.

CHAPTER 2

BUILDING EDUSYSTEM 50 FROM PAPER TAPE

2.1 BUILDING EDUSYSTEM 50

Building an EduSystem 50 software system is accomplshed in four phases.

I) Loading and initializing the Monitor.

II) Building the system program library.

III) Defining account numbers, passwords and quotas.

IV) Dumping the newly built system to DECtape.

Phase I requires the four custom-made Monitor paper tapes (SI, FIP, INIT, and TS8) plus the binary paper tape for PUTR. These paper tapes will be loaded onto the system disk after which a number of questions will be asked.

Phase II is accomplished with the Edusystem 50 software running. The system library is built in one of three ways, depending upon which medium system library programs are distributed on. If the library is distributed on DECtape, this step requires the library DECtape. If the library is distributed on an RK05 cartridge, this step requires the library cartridge. If the library is distributed on paper tapes, this step requires the library paper tapes.

Phase III is also done while the EduSystem-50 Monitor is in operation and uses the program LOGID to define user accounts.

Phase IV applies only to systems which include DECtape. It is accomplished by running EduSystem 50 "INIT".

2.2 BUILD PROCEDURE

I. LOADING AND INITIALIZATION

1. Turn computer power key to POWER (ON)

2. Turn the conole terminal ON (LINE or REMOTE).

3. Turn high speed reader ON

4. Lower, then raise the HALT switch.
   NOTE: On PDP-8/I, to 'LOWER' a switch means press the top of the switch in, 'RAISE' means push the bottom of the switch in.

5. Raise the SING STEP switch. Lower the DATA FIELD and INSTRUCTION FIELD switches if included on your computer (PDP-8 and PDP-8/I).

6. Press the CLEAR switch, (not present on PDP-8 or PDP-8/I).

For each step in the table, place each of the console switches numbered 0 to 11 either in the up position if the corresponding table entry is 1, or in the down position if the corresponding table entry is 0. When all 12 switches have been set to correspond to a line in the table, follow the instructions in the right hand column and proceed to the next line. The tables also include octal values of the binary switch settings for the benefit of users familiar with octal numbers.

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

Step #	Octal Values	Switch [redacted] Setting				And Then
		012	345	678	91011	
1	0000	000	000	000	000	press EXTD ADDR LOAD
2	7756	111	111	101	110	press ADDR LOAD
3	6032	110	000	011	010	lift DEP key
4	6031	110	000	011	001	lift DEP key
5	5357	101	011	101	111	lift DEP key
6	6036	110	000	011	110	lift DEP key
7	7106	111	001	000	110	lift DEP key
8	7006	111	000	000	110	lift DEP key
9	7510	111	101	001	000	lift DEP key
10	5357	101	011	101	111	lift DEP key
11	7006	111	000	000	110	lift DEP key
12	6031	110	000	011	001	lift DEP key
13	5367	101	011	110	111	lift DEP key
14	6034	110	000	011	100	lift DEP key
15	7420	111	100	010	000	lift DEP key
16	3776	011	111	111	110	lift DEP key
17	3376	011	011	111	110	lift DEP key
18	5356	101	011	101	110	lift DEP key

TABLE [redacted]-2. RIM LOADER FOR HIGH-SPEED READER

Step #	Octal Values	Switch [redacted] Setting				And Then
		012	345	678	91011	
1	0000	000	000	000	000	press EXTD ADDR LOAD
2	7756	111	111	101	110	press ADDR LOAD
3	6014	110	000	001	100	lift DEP key
4	6011	110	000	001	001	lift DEP key
5	5357	101	011	101	111	lift DEP key
6	6016	110	000	001	110	lift DEP key
7	7106	111	001	000	110	lift DEP key
8	7006	111	000	000	110	lift DEP key
9	7510	111	101	001	000	lift DEP key
10	5374	101	011	111	100	lift DEP key
11	7006	111	000	000	110	lift DEP key
12	6011	110	000	001	001	lift DEP key
13	5367	101	011	110	111	lift DEP key
14	6016	110	000	001	110	lift DEP key
15	7420	111	100	010	000	lift DEP key
16	3776	011	111	111	110	lift DEP key
17	3376	011	011	111	110	lift DEP key
18	5357	101	011	101	111	lift DEP key

steps 7-11 missing

After RIM has been loaded, it is good programming practice to verify that all instructions were stored properly. This can be done by performing the steps illustrated in Figure 2-3

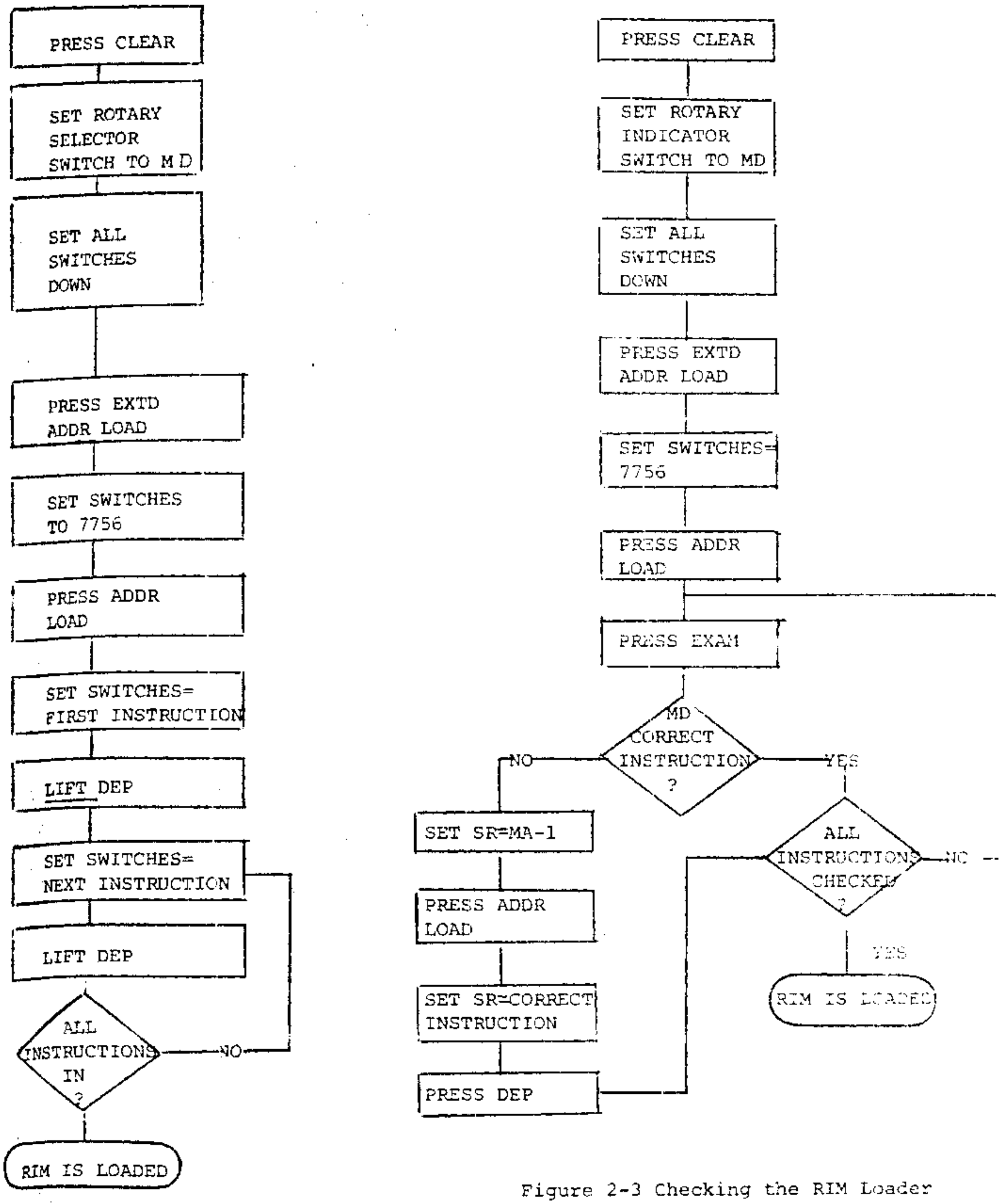
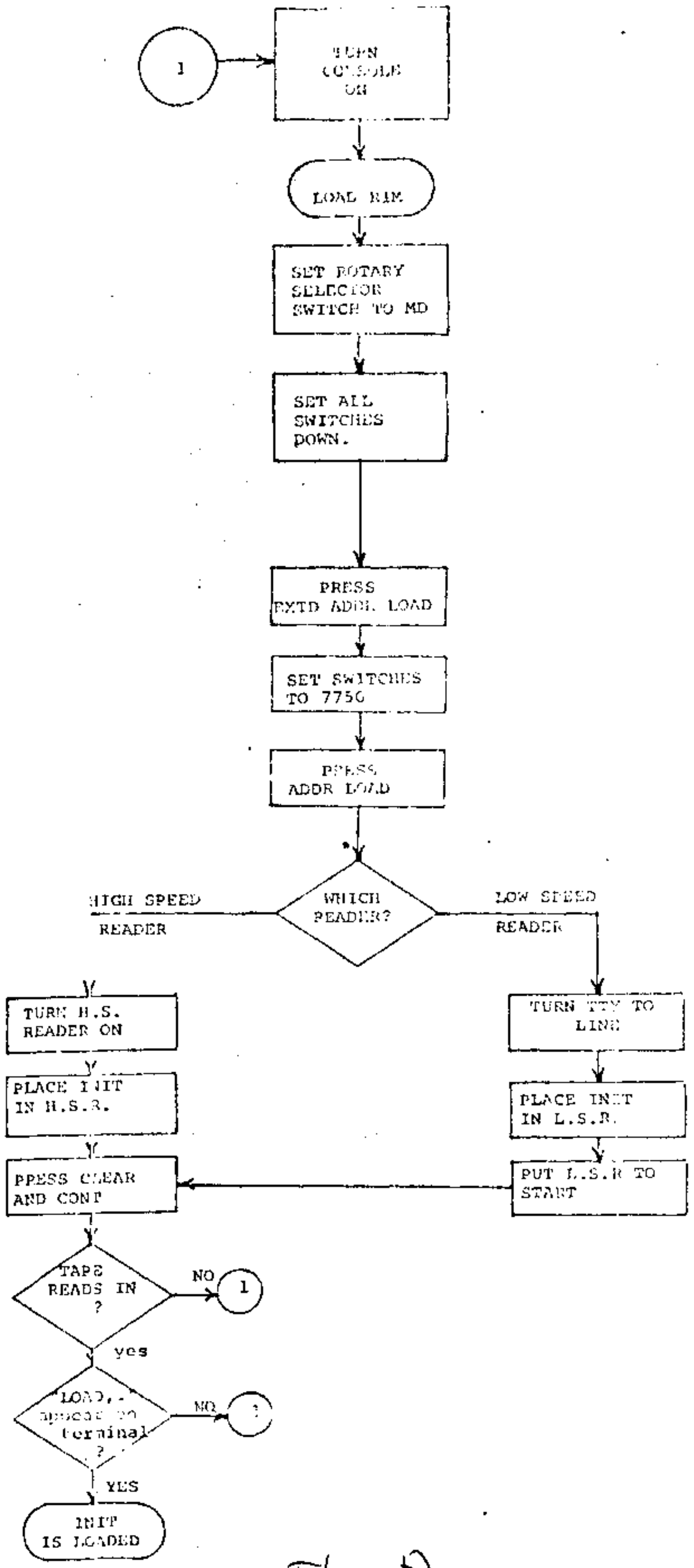


Figure 2-3 Checking the RIM Loader

12
12. After the RIM loader has been deposited and checked, raise switches 0,1,2,3,4,5,6,8,9, and 10; lower switches 7 and 11; then press ADDR LOAD. ↑↑↑ ↑↑↑ ↕↕↑ ↑↑↓
7 7 5 6

13
13. Press the CLEAR switch; then press the CONT switch. If your PDP-8 has no CLEAR switch press the START switch.

(figure E-5 on next page)


1
TURN CONSOLE ON
LOAD RIM
SET ROTARY SELECTOR SWITCH TO MD
SET ALL SWITCHES DOWN.
PRESS EXTD ADDR LOAD
SET SWITCHES TO 7756
PRESS ADDR LOAD
WHICH READER?
HIGH SPEED READER
LOW SPEED READER
TURN H.S. READER ON
PLACE INIT IN H.S.R.
PRESS CLEAR AND CONT
TURN TTY TO LINE
PLACE INIT IN L.S.R.
PUT L.S.R TO START
TAPE READS IN ?
NO
1
yes
LOAD,- appear on terminal ?
NO
?
YES
INIT IS LOADED

14. If the paper tape fails to read in at this point go back to step 1.

15. When the paper tape stops the message:

```
                LOAD, DUMP, START, ETC?
```

will be printed on the console. If this does not occur, go back to step 1.

16. Type BUILD followed by a RETURN. The system prints:

```
                BUILD?
```

Respond YES, followed by a RETURN, (labeled CR on some terminals).

17. The system prints:

```
                SI ↑
```

Place the paper tape 'SI' in the reader and type a RETURN. The tape will read in. If an error occurs while reading a tape, the wrong tape was loaded the system will print:

```
TAPE READ ERROR    OR    PLEASE TRY AGAIN
PLEASE TRY AGAIN         SI ↑
SI ↑
```

In either case, reload the requested tape ('SI' in this example) and type a RETURN .

18. After successfully reading 'SI' the system will request the paper tape 'FIP' be placed in the reader and the RETURN key typed, exactly as when loading the 'SI' tape.

19. Next the tape 'INIT' will be requested. This is the same tape which was previously loaded.

20. Next the tape 'TS8' will be requested.

21. Lastly the tape 'PUTR' will be requested.

If all Monitor subprograms have been loaded without difficulty, the printout on the console will appear as shown below.

```
LOAD, DUMP, START, ETC? BUILD
BUILD? YES

SI ↑
FIP ↑
INIT ↑
TS8 ↑
PUTR ↑
```

22. Next the system will print:

```
                NEW LOGIN MESSAGE?
```

The reply should be YES or NO, depending on wheth
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 anythin
else you desire as long as the message has no more
than 127 characters. Each RETURN counts as two
characters.

If the reply is NO the system will print its next
query, (go to step 23). However, if the reply is
YES, the system will print:

```
                END WITH ALTMODE
```

and position the console paper so that your messag
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 terminals.) The
printout and an example message would appear as
shown below.

```
NEW LOGIN MESSAGE? YES
END WITH ALTMODE
CONGRATULATIONS. YOU ARE NOW ON-LINE WITH EDUSYS
REPORT ANY PROBLEMS BY RUNNING "GRIPE".$
```

When typing the ALT MODE key at the end of the mes
a dollar sign ($) will be printed as shown above.

23. The system asks:

```
            LOAD EXEC DDT AT START-UP?
```

Respond NO, followed by RETURN.

24. The next query asks you to specify the number of
core fields available for user programs. Type a
number which is two less than the number of 4K fie
on the system, then type a RETURN. If the system
4 fields (16K of core) for example, the response s
be 2 as below.

```
                # USER FIELDS - 2
```

25. The system asks whether the CTRL/S feature is desi
This feature, which allows terminal output to be
stopped by typing CTRL/S and restarted by typing
CTRL/Q, is particularly useful on video terminals
such as the VT05 and VT50. Respond YES or NO, fol
by a RETURN.

```
              ENABLE ↑S FEATURE? YES
```

26. Next, the power frequency may be requested. If the power is 60 Hertz (normal in North America) respond YES; if not, respond NO, then type a RETURN.

```
60 HERTZ POWER? YES
```

27. The system asks whether the system disk should be zeroed. Respond YES, followed by a RETURN.

```
WRITE ZERO SYSTEM DIRECTORY? YES
```

28. Three passwords are requested, for accounts 1, 2, and 3. Enter these passwords, which may each be up to four characters long, followed by a RETURN.

```
1 SYSTEM PASSWORD? SYST
2 LIBRARY PASSWORD? LIBR
3 OPERATOR PASSWORD? OPER
```

29. The question "LOAD, DUMP, START, ETC??" is again printed. This time respond START followed by a RETURN.

```
LOAD, DUMP, START, ETC? START
```

30. When requested enter the current month, day, and year separated by hyphens followed by RETURN.

```
MONTH-DAY-YEAR: 1-23-75
```

31. Next type 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.

```
HR:MIN - 13:18
```

No additional questions will be asked after the time has been entered. This completes Phase I of the building process.

Your printout to this point might appear as follows:

```
LOAD, DUMP, START, ETC? BUILD
BUILD? YES

SI  ↑
FIP  ↑
INIT  ↑
TS8  ↑
PUTR  ↑
NEW LOGIN MESSAGE? YES
END WITH ALTMODE
CONGRATULATIONS. YOU ARE NOW ON-LINE WITH EDUSYSTEM-50.
REPORT ANY PROBLEMS BY RUNNING "GRIPE".$

LOAD EXEC DDT AT START-UP? NO
# USER FIELDS - 2
ENABLE ↑S FEATURE? YES
60 HERTZ POWER? YES

WRITE ZERO SYSTEM DIRECTORY? YES
SYSTEM PASSWORD? SYST
LIBRARY PASSWORD? LIBR
OPERATOR PASSWORD? OPER

LOAD, DUMP, START, ETC? START

MONTH-DAY-YEAR: 1-23-75
HR:MIN - 13:18
```

After entering the time of day and terminating the line with the RETURN key, control is transfered to the Edusystem-50 monitor. The system is now on-line and ready to operate. However, there are no programs in the system library.

II. BUILDING THE SYSTEM LIBRARY

Building up the system library is done while the system is on-line, i.e., operational and running.

1. LOGIN with the system library account number and password. Type

   LOGIN 2 LIBR

   replacing LIBR with the password for the library account. Terminate this LOGIN command with the RETURN key.

   The command LOGIN and account number and password will not echo (print) on the console paper.

When the LOGIN is accomplished, Monitor prints the version number of the Edusystem-50 Monitor being used, the job number assigned by the Monitor, the account number of the job, the number of the console being used, and the current time of day.

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/8.24    JOB 01  [00,02]  K00    13:18:10

CONGRATULATIONS. YOU ARE NOW ON-LINE WITH EDUSYSTEM-50.
REPORT ANY PROBLEMS BY RUNNING "GRIPE".
```

Type "START 0" followed by a RETURN.
This starts the program PUTR which was loaded during the build process. PUTR prints as asterisk, indicating that it is ready to accept a command.

```
.START 0

*
```

At this time, one of three procedures must be followed depending upon whether the library programs are supplied on paper tapes, Dectapes, or RK05 disk. If the library is on paper tapes, follow the steps in section A) below. If the library is on DECtape, follow the steps in section B), and if the library is on RK05 disk, follow the steps in section C).

A) The following steps are used to build the system library from paper tapes. First the system programs will be loaded, (those labelled "name.SAV", eg BASIC.SAV)

1) Load the paper tape labeled LOGOUT.SAV in the high speed reader. In response to the * which PUTR has printed, type:

```
COPY LOGOUT.SAV=PTR:/SAV
```

and terminate the line with a RETURN. PUTR will print "↑". Respond by typing a RETURN. PUTR will print "NONAME.", read the tape, and then print another *.

2) Step 1 should now be repeated for any other ".SAV" tapes to be loaded, substituting the name of the tape for "LOGOUT" in the COPY command. For example, if the tapes LOGID.SAV, BASIC.SAV, SYSTAT.SAV and PUTR2.SAV are loaded, the printout would appear as follows:

```
*COPY LOGID.SAV=PTR:/SAV
↑
NONAME.

*COPY BASIC.SAV=PTR:/SAV
↑
NONAME.

*COPY SYSTAT.SAV=PTR:/SAV
↑
NONAME.

*COPY PUTR2.SAV=PTR:/SAV
↑
NONAME.
```

As a minimum include at least LOGID.SAV, LOGOUT.SAV and PUTR2.SAV.

3) When all the desired ".SAV" tapes have been loaded type EXIT in response to the asterisk, followed by a RETURN.

4) Next type "R PUTR2" in response to the period the Monitor has printed at the left margin. PUTR2 will print an asterisk. The printout should appear as follows:

```
*EXIT
↑BS
.R PUTR2
*
```

5) If BASIC is loaded and it is desired to load some of the BASIC demonstration programs, ("name.BAS") load the desired tape into the reader and type, for example:

```
COPY FTBALL.BAS/BAS=PTR:
```

replacing FTBALL with the name on the tape. PUTR will respond as it did in Step 1.

6) If it is desired to load any ASCII paper tapes, ("name.ASC") load the desired tape into the reader and type, for example:

```
COPY WDGAME.ASC/TS8=PTR:
```

replacing WDGAME with the name on the tape. PUTR will respond as it did in Step 1.

Focal tapes ("name.FCL") must be loaded from the terminal using FOCAL.

7) When all tapes have been loaded type EXIT in response to the asterisk printed by PUTR followed by a RETURN.

8) Next type LOGOUT in response to the period the Monitor has printed at the left margin.

This completes phase two of the building process. Now go to Phase III.

B) The following steps are used to build the system library from the library DECtape.

1) Mount the library DECtape on unit 1, (see chapter 4 Introduction to Programming for complete instruction) and writelock the unit. In response to the * printed by PUTR, type the following command line:

```
COPY *=D1:
```

This command requests that all files on DECtape unit 1 be transferred to the system disk.

As each file is transferred, PUTR will type its name. When finished, PUTR will type another *.

2) In response to the *, type EXIT.

3) Next type "R PUTR2" in response to the period the Monitor has printed at the left margin. PUTR2 will print an asterisk. The printout for this step should appear as follows:

```
*EXIT
↑BS
.R PUTR2
*
```

4) In response to the asterisk, type EXIT, followed by a RETURN.

5) Next type LOGOUT in response to the period that the Monitor has printed at the left margin.

This completes phase two of the building process. Now go to Phase III.

C) The following steps are used to build the system library from the library RKØ5 cartridge.

1) Mount the library cartridge in unit Ø, and writelock the unit. In response to the * printed by PUTR, type the following command line:

```
    COPY *=RKAØ:
```

This instructs PUTR to copy all programs from the first half of the cartridge to the system library.

As each file is transferred, PUTR will type its name. When finished, PUTR will type another *. (The second half of the cartridge is a duplicate of the first half. To use this second half, substitute RKBØ for RKAØ.)

2) In response to the *, type E or EXIT.

3) Next type "R PUTR2" in response to the period the Monitor has printed at the left margin. PUTR2 will print an asterisk. The printout for this step should appear as follows:

```
    *EXIT
    ↑BS
    .R PUTR2
    *
```

4) In response to the asterisk type EXIT, followed by a RETURN

5) Next type LOGOUT in response to the period that the Monitor has printed at the left margin.

This completes phase two of the building process. Now go to Phase III.

III. Defining Account Numbers, Passwords and Quotas.

Users should never operate under accounts 1,2, or 3, therefore it is necessary to define additional accounts. Accounts can only be created by the system manager; someone logged in under account number one. Each account 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. 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. See the Protect command in Appendix B for details. In defining new account numbers it is useful to group users into projects, giving them account numbers which have a common project number.

As each account is defined the system manager also determines the maximum number of disk segments that the account may own. This is the quota for the account and is defined in multiples of twenty-five (25) segments; a minimum of Ø segments to a maximum of 1575.For normal use 50 - 100 segments will suffice.

The system manager also defines another parameter known as the "Grace Quota". This parameter applies equally to all accounts. The "Grace Quota" defines the number of segments each account may exceed its quota by for purposes of completing a program run. When an account's quota has been exceed the monitor will not allow any new files to be Created for that account, however any files already belonging to the account may be extended in lenth until the "Grace Quota" has been reached. At the time the system passwords for accounts 1,2 and 3 were defined the "Grace Quota" was automatically set to 1Ø segments and the quotas for accounts2 and 3 were set to maximum.

Passwords and quotas, including the "Grace Quota", may be changed at any time by the system manager and will take effect immediately. Account numbers cannot be changed, however accounts may be deleted, provided the account is not being used.

LOGID is the program used to create the user accounts and modify passwords and quotas. Since it can only be used by the system manager the next step requires that a console be logged in under account 1. The following responces should be terminated with the RETURN key.

1.) Type:

```
LOGIN 1 SYST
```

replacing SYST with the system password given during phase I step 28. The LOGIN command will not print on the terminal.

2.) Next LOGID must be called by typing:

```
.R LOGID
```

LOGID prints opening instructions, and then asks:

```
        PLEASE ENTER DISK QUOTA:
```

Enter a number which is a multiple of 25 (Ø-1575). This number will be used as the disk quota for the accounts defined or changed from this point on.

LOGID now prints an asterisk and waits for an account number, password combination separated by one space. Each account number can be from 1 to 4 octal digits (no 8's or 9's). Each password is made up of a maximum of 4 characters (all printable characters are legal). A maximum of 111 different accounts may be defined, (1Ø8 user accounts plus the 3 system accounts).

Typing CTRL/C causes LOGID to ask for the disk quota again. Therefore a number of accounts can be entered using one quota and then typing CTRL/C allows a new quota to be entered for the next group of accounts which are defined. An example dialogue might appear as follows:

```
        TSS8.24 JOB Ø1 (ØØ,Ø1) KØ4  13:3Ø:Ø8


        .R LOGID
        TSS8 ACCOUNT MAINTENANCE --
        *ACC'T # <SPACE> PASSWORD <RETURN TO CREATE/CHANGE,
        ALTMODE TO DELETE>
        PLEASE ENTER DISK QUOTA: 1ØØ
        * 1Ø DEMO
        * 732 TOUR
        * 1215 JOHN
        * 1Ø66 HARD
        * 1ØØØ OTTO
        *
        PLEASE ENTER DISK QUOTA: 75
        * 11Ø5 DECM
        *
```

To change the password or disk quota for an account, type the account number and old password as above, followed by a RETURN. LOGID will ask for a new password. Enter the new password and type RETURN. If only the quota is being changed simply type the RETURN key, no new password need be entered. In either case the quota last entered into LOGID will be applied to the account. For example:

```
        * 1Ø66 HARD
           CHANGE PASSWORD TO:  DZ8N
        *
        PLEASE ENTER DISK QUOTA: 15Ø
        * 732 TOUR
           CHANGE PASSWORD TO:
        * 1Ø DEMO
           CHANGE PASSWORD TO:  PLAY
        *
```

The disk quota does not apply to account 1. However, whenever the system manager's password is changed LOGID will request that the "Grace Quota" be entered. For example:

```
* 1 SYST
   CHANGE PASSWORD TO:  MNGR
GRACE: 29
*
```

To change both password and "Grace Quota" or simply:

```
* 1 MNGR
   CHANGE PASSWORD TO:
GRACE: 18
*
```

to retain the current password but change the "Grace Quota".

To delete an account type the account number and password as above but instead of typing the RETURN key, type the Altmode (ESC) key. If the account is not being used all files belonging to the account will be deleted, then the account w be deleted. When the account has been completely deleted the message:

```
$ DELETED
```

Will be printed to the right of the password as below:

```
* 1215 JOHN $ DELETED
*
```

When all desired accounts have been defined, type CTRL/B followed by S and RETURN.

```
* ↑BS
.
```

In order to create a listing of the accounts that have been entered the program CAT should be run as below:

```
.R CAT
SYSTEM ACCOUNT    26-FEB-75    10:18:00
PASSWORD    CPU         DEV       DISK  QUOTA

  1        00:00:00  00:11:05    12     18 (GRAC
  2        00:00:00  00:00:00   373   1575
  3        00:01:10  01:36:25    41   1575
 10 PLAY   00:00:00  00:00:00     0    150
732 TOUR   00:00:00  00:00:00     0    150
1066 D%8N  00:00:00  00:00:00     0     75
1000 OTTO  00:00:00  00:00:00     0    100
1105 DECM  00:00:00  00:00:00     0     75
RESET:  YES
↑BS
```

The RESET function causes the CPU and DEVICE time accumulato for all accounts to be set to zero. Running CAT by typing CAT:R will cause CAT to skip its listing phase and immediate ask RESET?

```
.LOGOUT
```

To complete phase III of the Build process type LOGOUT in response to the dot monitor has printed at the left margin.

```
JOB  1, USER [ 0, 1] LOGGED OFF K00 AT 10:18:57 ON 26 FEB 75
RUNTIME 00:00:01 (  0. CPU UNITS)
ELAPSED TIME 00:03:24
```

This completes phase III. If the system configuration includes DECtape continue with phase IV, otherwise this concludes the BUILDING process. EDUsystem-50 is ready to use.

Dumping the System To DECtape

To dump the newly completed system onto DECtape, restart INIT. as follows:

1.) Lower, then Raise the HALT (STOP) switch.

2.) Raise switches Ø and 4; lower switches 1,2,3,5,6,7,8 9,10, and 11. (The switches are now set equivalent to 4200 octal, the normal re-start address for Edusystem-50.)

3.) Press ADDR LOAD, then EXTD ADDR LOAD, then CLEAR, then CONT. (PDP-8 and PDP-8/I press ADDR LOAD, then START.)

4.) INIT will print the message:

```
LOAD, DUMP, START, ETC?
```

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 B.

A. Next mount DECtapes on units 1 and 2. Then set units 1 and 2 to WRITE ENABLE (see chapter 4 <u>Introduction to Programming</u> for complete instructions). Then type DUMP. INIT will copy an image of the entire system onto the DECtapes.

When INIT again prints:

```
LOAD, DUMP, START, ETC?
```

the entire system has been copied. Remove the DECtapes 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. (As in phase I steps 29 throught 31)

B. General Instructions for Dumping Disks to DECtape - The contents of an RS08 disk (256K words) will not quite fit on a single DECtape (190K words). Part of a second tape is required. In general:

Disks	DECtapes
1	2
2	3
3	5
.	.

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, ETC?.) 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. After the tape on unit 1 has re-wound, dismount it and 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. 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.

When INIT again prints:

```
LOAD, DUMP, START, ETC?
```

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. (As in phase I steps 29 through 31.)

CHAPTER 3

PATCHING EDUSYSTEM 50

The information in this chapter is not necessary to operate Edusystem 50. Most system managers will use the Edusystem 50 software exactly as it is supplied. Other users, however, will want to make minor modifications or, in some instances, major syster changes. This chapter describes the tools available for making sucl changes.

3.1 MODIFYING SYSTEM LIBRARY PROGRAMS

Modifying system library programs is an on-line process. Users who are familiar with Edusystem 50's advanced Monitor commands will finc it a simple 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 eve sixth character position to be a tab stop. The process is as follov for the 1970 version of Edusystem 50 EDIT:

```
.LOAD EDIT
.DEPOSIT 2 -6
.SAVE EDIT
```

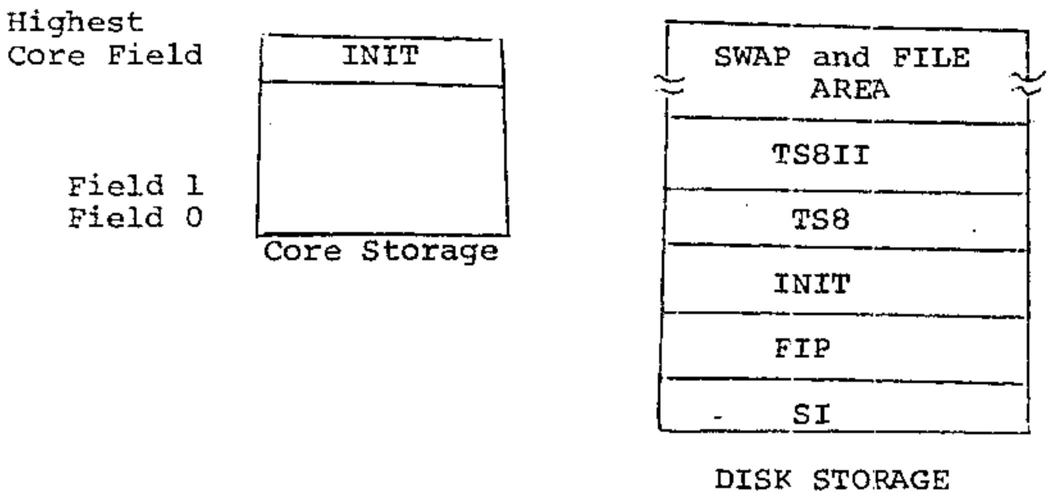
EDIT is now changed on the disk. If the system includes DECtape, dump the whole system so that the changed version is stored on the backup tape. If the system does not include DECtape, but has a higl speed punch, a new SAVE format paper tape should be punched with PUTR. Otherwise, the change must be made everytime the system is built. Other system library programs may be modified in a similar manner.

3.2 MODIFYING EDUSYSTEM 50

A formal procedure exists for making patches to the Monitor. In order to understand this procedure, it is necessary to understand how Edusystem 50 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 several programs, including 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 then start it at 4200. INIT is brought in and prints LOAD, DUMP, START, ETC??. At this point the layout of core and disk is as follows:



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

There are now two options for patching the system: either patch th disk using an overlay tape created with an assembler such as PALD, or manually inspect and change individual words using DISKLOOK.

3.2.1 Patching Edusystem 50 Monitor with an Overlay Tape. The overlay tape is created by writing and assembling a PALD program. The first item on the tape should be a field setting for the track number where the patch is to be made. The second item is an origin for the desired address within the field. Then include the data for the words to be altered. For example, to change words 6 and 7 of SI and FIP to 6213 and 5407, use the following program:

```
FIELD 0          /PATCH TO SI
*6               /START AT LOCATION 6
6213;5407        /DATA FOR LOCATIONS 6 AND 7
FIELD 1          /PATCH TO FIP
*6
6213;5407
$
```

Assemble using PALD, and punch out the binary tape. (XDDT users may find this patch helpful.)

Load the overlay tape into the paper tape reader. In response to INIT's "LOAD, DUMP, START, ETC??" message answer OVERLAY or simply 2. When the tape has been read, the patching is finishe. If there was a checksum error, a message "TAPE READ ERROR" will printed. The data previous to the most recent field setting wi have been written on the disk and thus may be incorrect.

3.2.2 Patching Edusystem 50 Using DISKLOOK - When INIT comes in, it prints LOAD, DUMP, START, ETC??. To start the patching procedure, type PATCH, or simply P.

DISKLOOK is now running, allowing the user to examine and modify single disk registers. To examine a register, type it's address (in octal) followed by a colon. DISKLOOK prints the present content of that register on the disk and waits 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 does not need changing, 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 7-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 are made, type CRTL/C to return to INIT.

An example of the usage of DISKLOOK:

```
                    LOAD, DUMP, START, ETC? P

                    42306:  5317  7604
                   76100:  6637  1220
                    40212:  6441  6051
                    40220:  6451  6061
                                                  (CTRL/C typed by
                    LOAD, DUMP, START, ETC?
```

Location 2306 in TS8II is changed from a JMP to a LAS. This change allows the system manager to examine selected Monitor regist by entering an address in the switches. If this patch is made, user programs may not use EAE Instructions. The pointer in locatic 6100 of SI is changed to point to an error return. This patch disables the TALK Command. Finally, locations 0212 and 0220 of TS8II are changed. This patch changes the device code of a termina from 44, 45 to 05, 06. (Note the exact locations may differ in future Monitors. These examples are for illustrations only.)

All changes to Monitor are made on the disk. Starting the system brings TS8 and TS8II into core from the disk, SI and FIP are swapped in by the system as needed, and INIT reads itself from the disk before it does any important operation. Therefore, any patche will become effective at the next startup and remain until the syst is rebuilt.

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.

3.3 CONTROLLING MONITOR EXECUTION

The XDDT program, which is always in core with INIT, is very useful for testing any modifications to Monitor.

There are two ways to keep XDDT in core while the Monitor is in operation. On systems with at least 16K, initialize the system specifying one fewer user field that normal. Then, insuring that

INIT (with XDDT) is in the highest field, start the system. The highest field will not be used by the Monitor, and XDDT will remain there.

Another alternative for getting XDDT into core is to initialize the system, and answer "YES" to the question about loading EXEC DDT. The result will be that when the system is started, XDDT will be placed into field 1 in an area normally used for free core. If EXEC DDT is loaded, the Monitor capacity will be restricted considerably, but otherwise will not be affected.

Once Edusystem 50 is up with XDDT in core, the system must be halted to start XDDT. Press the HALT key. If the EMA=0 and the MA=5200, fine. If not, press CONT and try again. Never attempt to halt the system if any I/O is in progress. Once the system has been halted at 5200 (this is the null job), restart the system at 7000 in the field of XDDT. XDDT may now be used to examine registers, set a breakpoint, etc. Information on the operation of XDDT is available from DECUS, order number 8-127. To restart the Monitor after being halted at 5200, start at 4201. (XDDT, type 0#4201').

Type CTRL/C to return to INIT from XDDT.

CHAPTER 4

LOAD, DUMP, START, ETC. USING INIT

4.1 HOW TO OBTAIN INIT

INIT is the program which allows the user to load, dump or start the system. Whenever "LOAD, DUMP, START, ETC?" appears, INIT is available. CTRL/C may be typed at any time to return to the entry point of INIT.

To start INIT if it is already in core, start at 4200 of the field where INIT is located. INIT can be found at various times in field 0, field 2, and the top field of the system.

If the Monitor is running, start at 4200 of field 0. TS8 includes a bootstrap starting at 4200 to read in INIT from track 2 of the disk to the highest field on the system, and transfer control to it. If the Monitor is on the disk, and the system has an EDU bootstrap, set the SR to 5350 and press and raise the SW switch.

If the Monitor is on the disk and there is no EDU bootstrap, load the following into field 0, then start the computer at 7750.

Address	Contents
7750	7600
7751	6603
7752	6622
7753	5352
7754	5752

If the above measures fail, use the rim or binary loader to load the paper tape of INIT.

INIT OPTIONS

CRTL/C	RESTART INIT
B	Build TSS/8 from paper tapes.
C	Transfer 4K sections between the system disk and core.
D	Dump the system disk to DECtape.
E	Bootstrap to a DECtape on unit Ø.
I	Initialize TSS/8 parameters.
K	Bootstrap to RK8E unit Ø.
L	Load the system disk from DECtape.
M	Compare the contents of two core fields.

If at any time, a drive is not ready, INIT waits for it. Just restart the drive when ready, and the tape will start. At the end of each tape, the tape will automatically rewind and unload. Remove the tape, label it with the unit numbers, and save.

4.4 LOADING THE SYSTEM FROM DECTAPE

At any time, the system may be restored to the state when the dump to DECtape occurred. To do this, mount the dump tapes on the same units they were on for the dump, get INIT, and specify L for Load.

4.5 INITIALIZING THE SYSTEM

To change any of the parameters (except for passwords), defined at system build time, get INIT, and specify I for INITIALIZE. One may now enter a new LOGIN message, change the number of user fields, etc. Upon the question "WRITE ZERO SYSTEM DIRECTORY?" answer no, or all of the files on the disk will be destroyed. If the LOGIN message is the only parameter to be changed, a CTRL/C may be typed after the ALTMODE. The following is an example.

```
LOAD, DUMP, START, ETC? I

NEW LOGIN MESSAGE? Y N
END WITH ALTMODE
THIS IS THE LOGIN MESSAGE!$
LOAD EXEC DDT AT START-UP? N N
# USER FIELDS - 2 6
ENABLE !S FEATURE? ~~N~~ Y Y
60 HERTZ POWER? Y   Y

WRITE ZERO SYSTEM DIRECTORY? N
```

[illegible]
32K = 8 fields - 2 = 6
see p. 2-7

4.6 USING TAPE READ OR WRITE

A selected number of disk tracks may be loaded from or dumped to DECtape. To do this, get INIT, and specify T for TAPE. Specify whether to read the tape (load the disk) or write the tape (dump), and how many tracks are desired (in octal). The following is an example.

```
LOAD, DUMP, START, ETC? T

TAPE READ OR WRITE - R
# - 5
```

4.7 USING THE 4K DISK-CORE TRANSFER

A selected core field may be read from or written to the disk, to or from any core field. To do this, get INIT and specify C. Then specify whether to read or to write the disk, the field number, and track number (in octal). Type CRTL/C to terminate this routine. The following is an example.

```
LOAD, DUMP, START, ETC? C

READ OR WRITE - R
FIELD NUMBER - 3
TRACK NUMBER - 5
```

4.8 READING A BINARY TAPE

INIT contains a modified binary loader beginning at 7777 of th field where INIT is. INIT's binary loader automatically choose low-speed or high-speed paper tape reader. If the device read responding, the loader times out in a few seconds, giving an error message. This is not a problem unless the tape is loadi into the same field as INIT. If there is an error condition, results will be unpredictable.

A binary tape may also be read by getting INIT, and specifying (Where n is the number of the field to which the tape is to be read). If a field setting for field n is encountered it will ignored. If any other field setting is encountered, the proc willhalt with that field in the AC. Press CONTinue to ignore For those using a PDP-8/E, set the data field as desired and p CONTinue.

4.9 ZEROING A FIELD

To write a zero in every location in a field, get INIT, and ty Zn, where n is the field number to be zeroed.

4.10 COMPARING FIELDS

To compare the contents of any two fields, get INIT and specit for Match. Then enter the two fields desired and the address t start at. Any differences between the two fields will be prin along with the address.

The printing may be interrupted by typing CTRL/C. The printin occur on the line printer if present and ready, or else on the terminal. If SRO is on, any mismatch with a zero on the first field specified will be ignored. Type CTRL/C to halt printing. The following is an example.

```
LOAD, DUMP, START, ETC? M
FIELD NUMBER - 0
FIELD NUMBER - 2
START AT - 0


0000  5401  0072
0001  4200  5402
0005  0477  0444
0442  0473  2477
```

4.11 DUMPING CORE

To dump the contents of core in octal, get INIT, specify W, and then give the desired field and starting address. Each line prin will contain a core address and eight data words. A dump may be interrupted by typing CTRL /C. The dump goes to a line printer if present and ready, or else to the terminal. The following is a example.

```
LOAD, DUMP, START, ETC? W
FIELD NUMBER - 0
START AT - 0


0000  5401  4200  0033  0033  0124  0477  0033  0124
0010  0007  0010  0011  0012  0013  3553  0215  0016
0020  0020  0021  0022  0023  0024  0025  0026  0027
```

4.12 BOOTSTRAPING TO OTHER DEVICES

When INIT is running, a bootstrap to DECtape or RK05 may be performed. To bootstrap to an operating system on an RK05 cartridge, load the cartridge on drive zero and type K.

To bootstrap to a DECtape, mount the tape with unit 0 or 8 selected, and type E.

To reboot INIT, or another operating system which may be on the system disk, type R.

4.13 USING XDDT UNDER TSS/8

When INIT is running, type X to jump to location 7000 which is the beginning of XDDT. XDDT is an octal - symbolic debugging program which preserves the status of the program interrupt system at breakpoints. (XDDT is DECUS order No. 8-527)

XDDT occupies locations 4434 through 7577 of any field. In addition, symbols may be defined by the user. These symbols will occupy locations 4433 down towards 000, destroying INIT or free core.

When XDDT sets a breakpoint, it uses locations 6 and 7 of every memory field. This may cause problems if an attempt is made to use breakpoint while user programs are running.

XDDT, as described here, has been changed slightly from the program available from DECUS. See the listing of INIT for these changes.

CHAPTER 5

USING EDUSYSTEM 50

LOGGING IN

protect the system from unauthorized use, each user must be
ntified by an account number and a password. For example, LOGIN
DEC is typed if the user's number is 456 and the password is DEC.
LOGIN command is not echoed on the terminal in order to protect
password.

STATES OF THE SYSTEM

yborad can be in one of three states: not logged in, SI (or
itor) mode, or user mode (not to be confused with the hardware
r Mode, UM light on PDP-8E). When the keyboard is not logged in,
y certain commands (such as LOGIN) are effective. When in SI
, the system is waiting for a command to be typed. Anything
d is considered to be a command. A user program may or may not
in the process of running at the time. If a program is not runnir
Monitor prints a dot to signify it is ready. When in user mode,
thing the user types is placed in a buffer, waiting for his
gram to ask for it (with a KRB, maybe). In summary, each characte
er than CTRL/B and CTRL/C) typed at the keyboard must go to one
two places. In user mode, it is saved for the user's program.
Monitor mode, it is saved and passed to SI as a command to the
itor.

SPECIAL CHARACTERS: RUBOUT, LINE FEED, CTRL/B, CTRL/C CTRL/S AND CTRL/Q

e characters have special immediate action and need not be
lowed with a carriage return to be effective.

keyboard is not logged in or is in Monitor (SI) mode, a RUBOUT
es the last character typed to be deleted. If the terminal is
d in the rubbed-out character is printed to help the user make
ections. In user mode, RUBOUT is just another character, which
ogram may interpret in any way. Most programs delete characters
RUBOUT is typed, usually typing a backslash or backarrow to
icate this function.

a terminal is in SI mode, a LINE FEED can be typed to cause
o print out the current command line. The procedure is
icularly useful if the user is in the midst of a complex
and and has used a number of RUBOUTS.

/B (printed ↑B on terminals) places the keyboard in Monitor
and clears the user's keyboard buffer to make room for a
and. The rest of the line is given to SI as a command. If
user's program is running, it continues to run. (However, if
program tries to do terminal I/O, execution is temporarily
ended until the user finishes the command.)

CHAPTER 6

RESTARTING EDUSYSTEM-50

Chapter 2 details the building procedure for the Monitor. Once this has been done, it is not normally necessary to repeat these procedures each time it is desired to start up EduSystem-50. Rather, a procedure called "bootstraping" or "booting" is followed.

There are several methods of booting EduSystem-50. The one which is most appropriate at any one time depends upon several factors, such as the configuration of the computer involved, and the state of the disk and core memories. The first method listed below is the easiest and is appropriate most of the time. The next two apply if the system disk is intact, while the following ones apply only if the system has been dumped to DECtape, and this tape is to be loaded back on to the system disk. If all else fails, the system must be rebuilt from paper tapes, as detailed in Chapter

6.1 PRELIMINARY PROCEDURES

Prior to using any of the methods listed below, it is necessary to initialize the computer system. If one of the methods is attempted and it fails, this initialization should be repeated before trying another method. To initialize the system: Ensure that the power to the computer is on, and that the switch is not in the PANEL LOCK position. Also make sure that the console terminal is turned on and is on-line and ready. Now press and raise the HALT switch, raise the SING STEP switch, and press the CLEAR switch. This completes the initialization.

METHOD 1

↑↓↓ ↓↑↓ ↓↓↓ ↓↓↓

Raise switches 0 and 4, while the others (1, 2, 3, 5, 6, 7, 8, 9, 10, and 11) should be lowered. Now press the ADDR LOAD, EXTD LOAD, HALT, and CONT Switches. The console terminal should print "LOAD, DUMP, START, ETC?" at which time the system may be STARTED. 4200

METHOD 2

This method applies if the system disk is intact and if the computer has a MI8-EG. Method 3 is a substitute for systems without the MI8-EG. Users who are not familiar with the MI8-EG may try this method anyway. If method 3 works where this one will not, the system does not have a MI8-EG.

5350

Raise switches 0, 2, 4, 5, 6, and 8, while switches 1, 3, 7, 9, 10 and 11 should be down. Press and raise the SW switch. The terminal should print "LOAD, DUMP, START, ETC?" at which time the system may be STARTED.

METHOD 3

This method applies whenever the contents of the system disk are intact.

...ons shown in the table below. For e step in the table, place each of the computer switch register swi numbered 0 to 11 either in the up position if the corresponding t entry is a 1 or in the down position if the corresponding table e is a 0. When all 12 switches have been set to correspond to a li in the table, follow the instructions in the right hand column an proceed to the next line.

At the completion of the last step, the console terminal should print "LOAD, DUMP, START, ETC?" at which time the system may be started.

Table 6-1 RF08/DF32 Disk Bootstrap

Step #	Octal Values	Switch Register Setting				And Then
		012	345	678	91011	
1	0000	000	000	000	000	press EXTD ADDR LOAD
2	7750	111	111	101	000	press ADDR LOAD
3	7600	111	110	000	000	lift DEP key
4	6603	110	110	000	011	lift DEP key
5	6622	110	110	010	010	lift DEP key
6	5352	101	011	101	010	lift DEP key
7	5752	101	111	101	010	lift DEP key
8	7750	111	111	101	000	press ADDR LOAD and press CLEAR and press CONT

METHOD 4

This method applies when there exists a set of dump tapes which w dumped previously, which it is desired to load onto the disk. In addition, this method makes use of the MI8-EG, and assumes that t dump tapes have been specially prepared using DTBOOT (see below). Method 5 is a substitute for systems without a MI8-EG, and Method is used when the dump tapes have not been specially prepared usin DTBOOT.

Place dump tape number one on a DECtape drive, write locked. How instead of selecting unit 1, place the unit select switch at 0 or 8.

Raise switches 3 and 4, while the others (0, 1, 2, 5, 6, 7, 8, 9, 10, and 11) should be down. Press and raise the sw switch. The should move for a few seconds, at which time the console should p "LOAD, DUMP, START, ETC?"

Turn the DECtape unit select switch back to its normal position o 1, and LOAD the system as described in section 4.4. After the LO has been accomplished, the system may be STARTED.

METHOD 5

This method applies whenever it is desired to load a set of dump tapes to the system disk which have been prepared with the progra DTBOOT. If the tapes have not been so prepared, use method 6.

dump tape number one on a DECtape drive, write locked. However, d of selecting unit 1, place the unit select switch at 0 or 8.

perform the switch manipulations shown in the table below. each step in the table, place each of the computer switch register ches numbered 0 to 11 either in the up position if the correspondin entry is a 1, or in the down position if the corresponding entry is a 0. When all 12 switches have been set to correspond line in the table, follow the instructions in the right hand and proceed to the next line.

Table 6-2 TC01/TC08 DECtape Bootstrap

[illegible] #	Octal Values	Switch Register Setting				And Then
		012	345	678	91011	
	0000	000	000	000	000	press EXTD ADDR LOAD
	7613	111	110	001	011	press ADDR LOAD
	6774	110	111	111	100	lift DEP key
	1222	001	010	010	010	lift DEP key
	6766	110	111	110	110	lift DEP key
	6771	110	111	111	001	lift DEP key
	5216	101	010	001	110	lift DEP key
	1223	001	010	010	011	lift DEP key
	5215	101	010	001	101	lift DEP key
	0600	000	110	000	000	lift DEP key
	0220	000	010	010	000	lift DEP key
	7754	111	111	101	100	press ADDR LOAD
	7577	111	101	111	111	lift DEP key
	7577	111	101	111	111	lift DEP key
	7613	111	110	001	011	press ADDRLOAD and press CLEAR and press CONT

the last step above has been performed, the tape should move a few seconds, and then the console terminal should print out DUMP START, ETC?"

the unit select switch on the DECtape back to its normal position and LOAD the system as described in section 4.4. After the has been accomplished, the system may be STARTED.

METHOD 6

method is used whenever it is necessary to load DUMPed tapes have not been specially prepared using DTBOOT.

to chapter two, following the steps for building a new system the message "LOAD, DUMP, START, ETC?" appears on the console . As soon as this message appears, return to the following .

response to the question "LOAD, DUMP, START, ETC?" the system may LOADed, as described in section 4.4. After the load has been accomplished, the system may be STARTed.

METHOD 7

When all else fails, the system must be rebuilt from scratch, as detailed in chapter 2.

PREPARING TAPES USING DTBOOT

To specially prepare a set of dump tapes using DTBOOT, take tape number 1 of the set of dump tapes, and mount it on a DECtape dri
However, instead of selecting the normal unit 1, place the unit s
switch at 0 or 8. Place the DECtape unit in the WRITE ENABLED position. Now, with the EDUsystem-50 monitor in operation, type "R DTBOOT" at any logged-in terminal (the program DTBOOT must, of course, be stored in the system library prior to this step).
all is well, the tape will move (possibly only a fraction of a r
and ↑BS will be printed. The completes the preparation of the ta
This preparation is permanent for this tape, and will normally ne
need to be repeated no matter how many times the system is LOAD
from or DUMPed to that set of tapes.

A NOTE FOR OS/8 USERS:

INIT may be used to advantage when using the system disk for running OS/8. Load the paper tape of INIT using standard procedu
for binary paper tapes, and start it at 24200. Even better, crea
a SAVE format file of INIT, and simply run it. When INIT asks
LOAD, DUMP, START, ETC?" dump OS/8 to DECtape exactly as if it were EDU-50. At the completion of the dump, INIT will boot the system which is on the system disk, which happens to be OS/8; hen
another OS/8 monitor dot will be printed. Now, EDU-50 may be loa
At any future time, when it is desired to run OS/8, get back INIT and LOAD the OS/8 tapes as if they were TSS/8 tapes. At the completion of the load, INIT will boot in the system disk, which contains OS/8 again, and an OS/8 monotor dot will be printed. Caution: do not attempt to use DTBOOT on these OS/8 dump tapes, a
do not try to boot in these tapes directly. Use INIT.

NOTE TO VERSION 3 USERS

The program BOOT may be used to boot EDU-50. When not running of
of the system disk, typing BO/RF will bring in INIT from the system disk. When not running from DECtape, typing BO/DT will br
in INIT from DECtape unit 0, provided that it has been specially prepared using DTBOOT.

CHAPTER 7

DETAILED MONITOR OPERATION: I

7.1 SHARING TIME

The most fundamental job of a timesharing Monitor is the sequent execution (generally for short bursts, or quanta of time) of a number of user programs. This implies that the Monitor has a place availabl where a user program can be brought to execute, and a place to put user programs when not being run. EduSystem 50 reserves one or more core fields within the PDP-8 as areas in which to execute user programs. A user program, and hence a user area, is 4K words long. EduSystem 50 may have from 1 to 6 user areas, depending on the amount of core available. Similarly, EduSystem 50 reserves a portion of its disk as a place in which to keep programs 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; the 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 EduSystem 50 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.

EduSystem 50's swapping algorithm may be best illustrated by assuming a very simplified situation. EduSystem has a number of user programs running within it; each is compute bound, 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 the 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. This information, the AC, PC, LK, and EAE registers, is stored in Monitor core. The Monitor 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, and EAE registers of each user are stored in the Monitor when they stop running) and the program is started. This procedure is continued as long as the user program needs to run.

Obviously, the Monitor has to maintain status information about each user program, whether or not it is in core. 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 whether or not a user program needs to be run. In actual 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 I/O, the only information that needs to be maintained is whether or not the user has finished. If a user program completes its run, the Monitor remembers this fact. The program is swapped out and remains 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 again, this user is swapped in to run some more.

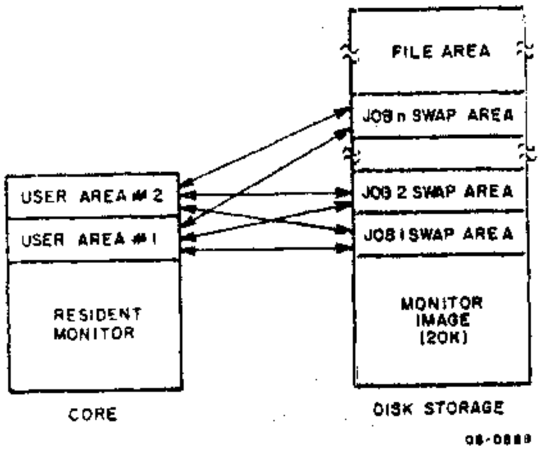


Figure 7-1. 16K EduSystem 50 Configured for n User Programs

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

In a system with a single user field, the scheduling algorithm is such that some previous user program job must always be swapped out to make room for the next. Once a user program is brought in and started, there can be no further scheduling activity until it has completed its quantum. Similarly, once the user program in core has started to be swapped out, the system must wait until the next user program 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 1-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 scheduler decides that a given user job 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 occurs. When the program has been run for a quantum, its run state is saved but it is not swapped out. The scheduler 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, the scheduler 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 scheduler maintains, in addition to its table of all user jobs, a table of all user jobs which are in user core. (A job may be in core, on the disk, or halfway between when it is being swapped.) It actually scans the former table to decide which to swap next and the latter table to determine what to do in the meantime (while it is waiting for the swap to be completed). 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 on which to work. When a user program is scheduled to be swapped out, it disappears from the list of in-core programs. Eventually, the next program scheduled to be swapped in will be read into core. 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 2-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 in which 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). This is explained in the following paragraphs.

A user program which has just been run is scheduled to be swapped out. In the table of in-core programs, it is marked as no longer in core. The scheduler then determines if there is anything 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. Scheduler then sets up and runs it. (Note that if this swap is not completed until after the second swap was started, the Monitor must wait for it to be made. 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 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 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 1-user core system, swaps and runs alternate; in a 2-user core system, they are simultaneous. It is a foreground-background operation.

The scheduler depends on various interrupts to continue this process. Specifically, the scheduling is driven by the clock and disk completion interrupts. After every successful swap and after every 1ØØ milliseconds the scheduler is run. If the scheduler is run because of a clock interrupt, it checks to see if this is the second such clock interrupt it has encountered since it started to run the presently-running user program. If not, then this user program has not had its full quantum of runtime. It is therefore restarted. When the second scheduler clock call occurs, indicating that the user program has run for a full 200 milliseconds, it is marked as having been run. The scheduler 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 a swap is being made, the scheduler knows that eventually a new user program will be in core. It returns and runs the same program. Eventually, the program being swapped will be in core and run. Even if there is another program in core, the scheduler checks to see if a swap is in progress. If it is, the scheduler simply starts and runs the next resident user program.

Whenever the scheduler finds there is no swapping, it checks to see if a swap is necessary. A swap is necessary if a user program is on disk which needs to be run. Thus, when the scheduler finds no swapping taking place, it checks its table of user programs to find a runnable, swapped-out user program. If it does, it schedules this program to be swapped in. (Generally, this means swapping out another user's program.) Once the scheduler has set up the swap (if one is required) it finds the next resident user program and starts it. (Note: the check for swapping activity actually occurs every 1ØØ milliseconds to assure that the swapping rate is maintained.)

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, the transfer 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 no longer to be in core at the time it is scheduled to be swapped, even though it may not actually be written out until sometime later.

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

Thus, scheduling 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 scheduler 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 scheduler. Thus, a user program which the scheduler selects to be swapped in will eventually be swapped into a core and hence appear in the table of in-core user programs. The scheduler, scanning this table for a resident job to run, will find and run it.

It is important to system efficiency that, at the time the scheduler goes to the table of in-core user programs to find one to run, it finds one. If it does not, it schedules a non-job, the "null job," to be run. This null job is run until a valid user program is in core. (The null job is a tight loop in Monitor core which increments the accumulator. It does not occupy a user core. It is not swapped.) Clearly, in a 1-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, the Monitor can do nothing but run null job. In a 2-user core system, the effeciency is much greater. The background swapping assures that a new user program will be in core at about the time the currently active program completes 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 later in this manual.

7.2 SOME DEFINITIONS

In the preceding discussion, we have referred to the programs running in EduSystem 50 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.

A job in EduSystem 50 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 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, the 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 terminal 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. The 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 the job number; if it is not, it is forgotten.

The Monitor maintains a table, JOBTBL, which indicates the status of each job. It has a 1-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. The Monitor also maintains a table of in-core jobs. This table is called CORTBL. It is made up of a 1-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.

7.3 TALKING TO THE USER PROGRAM

The preceding discussion is limited to compute-bound jobs, those that do neither input nor output. This situation is rare. Most user jobs do a great deal of console I/O. For the Monitor to process this console I/O, it must solve a number of immediate problems. First, it must be able to handle multiple consoles. All EduSystem 50 configurations have multi-terminal hardware, allowing the system to input characters from any given console and output them to any console. However, it must also determine which console is which, and which characters are received from which console. User programs on EduSystem 50 are regular PDP-8 programs; as such they input characters from the console and output them to the console. There is no ambiguity in a stand-alone system that has only one console. In EduSystem 50, where many jobs are outputting to the console, the potential for confusion is considerable. The EduSystem 50 Monitor must maintain a table listing which console is used by what jobs. Thus, when a job does console I/O, Monitor knows the individual console involved.

These are the immediate problems the Monitor must solve. However, to be useful, it must also be efficient. Normally, a PDP-8 program doing I/O spends virtually all its time waiting for the device; it is the monitor's responsibility to recover this time and use it to run another job. Finally, the Monitor should smooth the I/O. EduSystem 50 is a swapping system; user programs are in core only for short periods, then they are swapped out to the disk. If a user program could only output when it is in core, its typeout would be sporadic. Input would be worse. If a user program could only input when the job happened to be in core, no input would be done.

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

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 or not the user's program is in core. It is essentially an overhead function. A little processor time is taken from whatever program is currently running and used to keep up the I/O for all active users.

This interrupt driven terminal handler solves the problem of shuttling characters between console and buffers. There remains the problem of passing characters between these buffers and actual user programs. This character passing occurs via the EduSystem 50 hardware trapping capability. On input, the key instruction is KRB, the keyboard-read ICT. A user program, when it inputs a character, executes this KRB. The hardware modification causes a trap to Monitor, preventing hardware execution of the instruction. On identifying the trapped ICT as a KRB, the Monitor 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 never knows 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 is in the accumulator at the time of the trap and places it in the appropriate output buffer. Once again, the IOT has been precisely simulated. (The asynchronous terminal routine assures that the characters placed in the output buffer are typed eventually.)

However, the ability to simulate KRB and TLS is only half the job. There remain all the timing and synchronization problems which are normally solved by the skip IOTs: KSF and TSF. In a stand-alone PDP-8, KSF means "Is there a character in the input buffer?" In TSS/8, the 1-character hardware keyboard buffer is effectively replaced by a multicharacter software input buffer. Thus, in EduSystem 50, KSF means "Are there any characters in the input buffer?" KSF, being an IOT, traps from a user program. The Monitor, upon identifying the trapped IOT as a KSF, checks that user's input buffer. If there are any characters in it, the Monitor simulates a skip by returning to the instruction in the user program which is two locations beyond the KSF.

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

```
LOOP,   TAD I   AX      /AUTOINDEX
        SNA
        HLT
        TLS
        CLA
        JMP     LOOP
```

Each TLS puts a character in the output buffer (assuming it is not full). 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 terminal handlers type them all. This is, however, 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 I/O. Rather than wait 1/10 of a second between each character, output 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, the Monitor can use them to run other jobs. This timing is handled by the scheduler and device handlers. It is important that the user never hang in a KSF; JMP .-1 loop as on a stand-alone system. This is the code normally used to wait until more I/O can be done. On EduSystem 50, the user job cannot be left to waste processor time in this loop. Therefore, when the Monitor detects a KSF which is false followed by a JMP.-1 (i.e., the program must wait for the device), it stops the program just as if the time slice had been completely used. 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 why it is not runnable (waiting for input) is remembered. Since it is not runnable, it is dropped from the run queue and, when the scheduler finds it the next time, it will not be run. However, the Monitor continues to keep track of the state of the I/O. At the time when the device is again available, the Monitor changes the state of the job back to runnable. Now the next time the scheduler looks at this job it will run. The job will be started where it had stopped (at the skip instruction) but this time the skip is true, allowing the program to continue. Thus, by trapping the skip IOT, the Monitor has salvaged the wait time from a job and used it to run other jobs.

In order to make running user programs even more efficient, the 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 on the flag, thus starting the user program only when it is possible to process many characters. This is done on input by setting up a "break mask." This break mask tells the Monitor what characters are important delimiters. For example, BASIC considers carriage return and rubout to be delimiters. When BASIC is ready for keyboard input, it executes a KSF to see if there is any. Typically, there is none. (The user has not yet begun to type in the next line in his program.) Therefore, this user program is put in the I/O wait state and is marked as not runnable. It stays in this state until a delimiter appears. The keyboard flag is then set, the program is returned to the runnable state, scheduled, and run. As soon as the program starts, it executes KRBs to read input characters. BASIC 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 using very little system time. The same situation applies to output. As the program outputs characters, these characters are placed in the output buffer. As long as the buffer is not filled, the program is allowed to run. However, when the buffer fills, the program's teleprinter flag is cleared, thus suspending execution of the program (it moves into I/O wait). As characters from this buffer are subsequently typed, ending the buffer-full condition, the teleprinter flag is held down and the user program is kept in the non-runnable 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, keeping a continuous output. Programs like BASIC can fill a character output buffer in one time slice. Therefore, input, like output, is accomplished without substantial processor time.

It is the combination of the two parts of the I/O handlers: those driven by IOT traps and operated synchronously with respect to the user program, and those driven by device interrupts and operated asynchronously with respect to user programs, which accomplish I/O. The common communication 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. This is done on input by means of the delimiter, or break mask. On output, it is done by detecting buffer-full and buffer-almost-empty conditions.

The above discussion is somewhat simplified. Actually, programs need not use the skip-on-flag instructions at all and may use string I/O instructions to transfer many characters at once.

CHAPTER 8

MONITOR: A MORE DETAILED LOOK

Thus far, we have reviewed some of the operations of the EduSystem 50 Monitor and how it responds to various simplified situations. This chapter discusses these operations in greater detail: the various subsystems within the Monitor, the full scheduling algorithm, and the data base.

8.1 MONITOR AS INTERRUPT HANDLER

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

Interrupts to the Monitor are divided into three levels: level 0, level 1, and level 2. The DCØ8A clock is the only level 0 interrupt. The workings of the clock routines are dependent on the TSS/8 configuration. In a PTØ8 or KL8E system, there is a line frequency clock. In a DCØ8, there is just the DCØ8 baud clock. This clock then serves both as the system clock and the signal to enter the DCØ8 service routines. The Monitor does not take action on every clock interrupt. It waits for 100-millisecond intervals (12 ticks of a line frequency clock - 55 ticks of a DCØ8 clock). Thus, when the clock interrupt occurs, the clock interrupt handler simply increments a counter to see if 100 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 continue output.) Only at 100-millisecond intervals does it involve more Monitor processing. In this case, it is treated as a level 2 interrupt. DC08A 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 or KL8Es, the console terminal 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 system before dismissing. Therefore, they have no problems in protecting themselves against being re-interrupted. This is not the case with any of the other interrupt processors. These "reenable interrupts" which are considered to be level 2 interrupts, reenable interrupts 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 100-millisecond clock, service for keyboards, teleprinters, reader, punch, or line printer, operation complete for the card reader, disks, or DECtape, or a trapped user IOT interrupt. Level 0 and level 1 interrupt handlers take up a miniscule amount of code. Therefore, the 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 re-interrupted. 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, the Monitor checks for, and prevents, this situation. Whenever a level 2 interrupt is detected, the Monitor checks to see whether it is a user mode program which is 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 the Monitor was in the process of handling level Ø or a previous level 2 interrupt (the only conditions under which an interrupt out of exec mode could occur). In this case, processing 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, back to a location within the Monitor. At the completion of each level 2 interrupt, the Monitor checks this level 2 queue. If it is empty, it dismisses back to the user program. If it is not empty, the Monitor is reentered to process the next request on the queue. Only when the backlog on level 2 queue is exhausted does an exit occur from the Monitor.

In the case of 100-millisecond clock interrupts, the level 2 handlers save the whole state of the user job in his job status registers. When this interrupt is finally dismissed, the saved state of the job, with its job number in core register JOB, is restored. If, in the meantime, the scheduler has changed the contents of JOB, it is in fact a new job which is started. Thus, even the system scheduling is accomplished by means of the interrupt handlers.

It is important to keep this concept of the Monitor as interrupt handler in mind since the system is incomprehensible when viewed in any other light. All actions by the Monitor may eventually be traced to some interrupt. Swapping occurs in response to a disk completion flag. When the disk completion is detected, the Monitor, via a 2-level interrupt, looks to see what the next swap should be. When the swap is found, the Monitor initiates it. Scheduling occurs as a response to the 100-millisecond clock level 2 interrupt. If it is the second such interrupt since a user job has been started, the Monitor looks for a new job to run. Even in the interim level 2 clock interrupts, the Monitor tries to do some advance scheduling. If there is no swapping, the Monitor sees if it should begin swapping. Thus, while a job is running, the scheduler tries to get the next job ready, so that it may be started immediately after the current job completes its time slice. When a job does complete its time slice, the scheduler's task is to set up and start the next job.

Terminal, paper tape, and line printer I/O are handled by special level 2 routines. These routines are run every time an interrupt from one of the devices occurs. In the case of terminal input and the paper tape reader, characters are stored in a ring buffer at level 1, and removed and placed into the appropriate user buffer in free core at level 2. In the case of terminal output, the paper tape punch, and the line printer, there is a one-word buffer for each device. When a device interrupts, a character is removed from this one-word buffer (if there is one there) and immediately transmitted. Later, at level 2, a character is removed from the appropriate free-core buffer and placed into the one-word buffer. Special code allows the line printer to fill its hardware buffer at level 2 with the interrupt system enabled and without going through the one-word buffer.

All interaction between the jobs and the system takes place through the medium of the IOT traps. The scheduler 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 the Monitor has identified an IOT trap interrupt, it tries to identify the IOT that caused the interrupt. 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, the Monitor dispatches to the appropriate routine. If it is not found, 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 all this operates is duscussed in the next section.) Some IOTs always cause control to be returned to the user program immediately. Among these are all the 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 EduSystem 50 Monitor. Those IOTs which communicate just with the Monitor return to the user program immediately.

The IOTs which correspond to the actual devices, such as the terminal IOTs, may or may not return to the user program immediately. A true KSF (the keyboard buffer has one or more delimiters in it), allows control to be returned immediately to the user's 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 is 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.

8.2 I/O WAIT CONDITION

The user program is only allowed to run again after one of these IOTs if the program 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 the user program is looking for input which is not there (false KSF or unsuccessful KRB) or trying to output where there is no room. On a stand-alone PDP-8, the program goes into a wait loop until it can do more I/O. Under EdySystem-50, 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 terminal. In all cases, buffers of characters are passed between Monitor and user programs. The programs enter 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 for what I/O device the user program is waiting. The mask, which corresponds exactly to the user's status register (STR1), 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 program is to be stopped, the bit corresponding to the device for which the program is waiting is set. Thus, if the user program executes a KRB when its input buffer is empty, the bit in the wait mask which corresponds to the keyboard flag is set. The user program is not restarted and control returns to scheduler so that another user program can be run. Thus, whenever a user program is in an I/O wait, a single bit in the wait mask indicates the device for which it is waiting. (Some transfers, such as file reads and writes, always place the user program into a wait state. Others, like the terminal do so only when a buffer fills.)

The scheduler uses the wait mask to decide which jobs to run. First, the scheduler keeps a run bit in the job status register for each user. A user's run bit is on if there is 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 having run bits set, only those not in an I/O wait state are actually scheduled to run.

In deciding what user to schedule next, the scheduler scans the list of active jobs looking for one with its run bit 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 is zero. Only one bit in the wait mask is set - the bit corresponding to the flag for which the job is waiting. 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 types a delimiter on the keyboard. This causes the delimiter bit to be set. The next time the scheduler 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 set as a result of level 1 interrupts; i.e., those that do the data transfers. They are detected by the level 2 scheduler when it looks for the next runnable job.

This mode of operation characterizes the whole Monitor. The Monitor is made up of a number of asynchronous elements which communicate via status registers and request queues. The Scheduler, which is the heart of the Monitor, is guaranteed to be run every 100 milliseconds. Therefore, it is not necessary for another routine, such as the disk handlers, to jump directly to the scheduler in order to indicate that a swap is complete. To indicate the new system status, the disk routines need only set the appropriate status bits. The next time the scheduler is run, it finds this updated status and acts accordingly. Similarly, if the scheduler 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 is checked and the queued transfer initiated. However, if there is no disk transfer in progress when the scheduler decides to do a swap, it cannot just queue the request. In this case, the scheduler itself must initiate the transfer.

8.3 OTHER PARTS OF MONITOR

The Monitor code which performs the functions discussed so far is permanently resident in field zero while EduSystem 50 is operating. Field zero contains almost all the resident code. The 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.

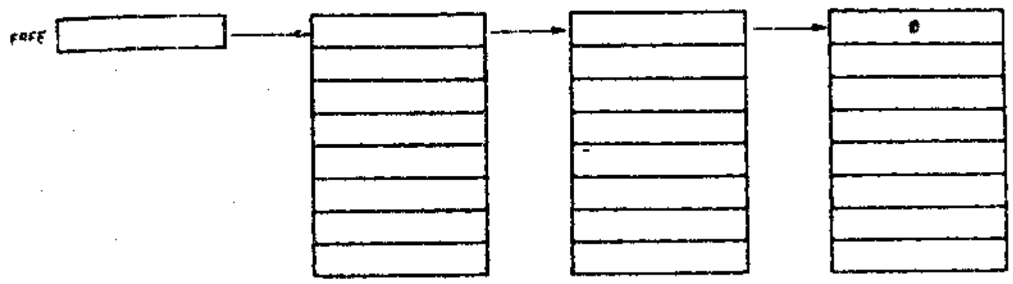
In addition, there are two nonresident sections of Monitor code. They are the System Interpreter (SI) and the file handler (FIP). These routines are not frequently used and do not need to be core resident. FIP is a 4K block of code which resides in the second 4K block of the disk (disk locations 10000-17777). SI makes 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 the resident Monitor; they go into the first user field. In fact, the scheduler sets them up 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). 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.

8.4 THE MONITOR DATA BASE

Some mention has been made of the tables and buffers used by the Monitor. Diagrams of the tables and buffers used by the Monitor may be found in Appendix D. These should be referred to as specific tables are mentioned. A brief discussion of the tables follows.

The Monitor does a great deal of dynamic storage assignment. It uses 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.

A location in field 0, called FREE, contains a "pointer" to the first unused block of free core. A pointer is simply the address of the first word in the block. The first word of this block contains a pointer to the next block, etc., to the end. When a routine needs some place to store data, it can remove a block from free core, adjust the list accordingly, use the block, and later return it.



ORGANIZATION OF FREE CORE

As blocks are removed from and replaced back into free core, a count is retained of free core size. This is in location FRECNT. FRECNT always contains the number of unused 8-word free core blocks.

Note that all free core blocks begin at an address divisible by 8; that is the last octal digit is a Ø. If a free-core block ever seems to start at an address which does not have the last digit 0, the system is in trouble.

This free core is used by the Monitor for a variety of purposes. Terminal buffers are made up of linked blocks of free core; device and job status information are also stored in the free core. Free core is also used in a number of instances for temporary scratch storage.

The device handlers for terminals 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 1-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 the device 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. For most devices, the buffers are dynamically allocated from free core. As characters are entered from the keyboard and put in the buffers they are put into 8-word blocks of free core. As one block fills, 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 KRBs), 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 is to be put into the buffer) and 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 in the same way.

Thus, console input and output operate independently from the rest of the system. As characters are entered, they are put into input buffers (up to about 9Ø 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 terminal handler. The terminal handler merely passes characters and adjusts the appropriate flags.

Just as each active console is marked in DEVTBL, each active job is marked with a job status table, JOBTBL, which is a fixed table with a 1-word entry for each possible system job. Non-existent 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 linked together; these status blocks contain all information about this job's running state. If there are open files, blocks exist which contain their status.

Finally, there are tables the 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 9

SYSTEM STORAGE AND COMMUNICATION

9.1 TALKING TO THE USER

Until now, we have assumed that jobs running in the system ither 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 in the rst place, much less how it was loaded and otherwise controlled. ese are functions which, on a stand-alone machine, are not per- rmed through the terminal at all -- they are done through the itches on the console. When talking to EduSystem 50, however, ere is only one physical device, the user's terminal, through hich to perform these two kinds of communication: communicating ith EduSystem 50 and communicating with a user program running as a job within that system.

EduSystem 50 makes a careful distinction between these two odes. A user is always uniquely in one mode or the other, dependi on the state of his job. Whenever a user starts executing a progra his console is put in program communication mode. It stays in that ode until the program is interrupted or terminated. If the progra is terminated, the console automatically returns to system communi- ation mode. It is also possible to make one-shot inputs to the system without halting the user program.

In order to minimize confusion, EduSystem 50 has some conven- tions to distinguish between system and user mode. The system alwa types a period at the margin to indicate that a terminal is in syst ode and that the system is ready to accept a new command. The CTRL/B character tells the system that regardless of the mode of t terminal, the characters following the CTRL/B are to be treated as though the terminal were in system mode. Thus, even if the termin is in user program mode, all characters following a CTRL/B up to t next carriage return, are input to the system.

When the user walks up to an EduSystem 50 console, he finds it in system mode. If the user types a carriage return, thereby entering a null command, EduSystem 50 responds with a period at the margin. The user can then type a command to the system. At this point, the terminal is actually in a special system mode -- it is logged out. This means

a) Input is not echoed to the terminal, and

b) Only selected commands, such as LOGIN, TIME, and VERSION, are considered valid.

To the system all other commands are 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 a password are valid, the user is logged in. The terminal remains in the system mode, but input is now duplexed and all system commands are now valid. (If the login is invalid, the user must try again.)

The user remains in the system mode then types a command which causes a program to be started. This is done by means of the START command which takes an octal address as an argument. (A program can also be started with an R or RUN command.) The START command starts the program and puts the terminal 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 way is for the program to execute a HLT. The other way is to type an S (for STOP) command to the system. However, since the terminal is in user program mode, it is necessary to preface this S by a CTRL/B in order to get the attention of the system. Notice that by typing CTRL/B while a program is running, many commands may be entered to the system. Only S, however, will send the terminal back to system mode. With the other commands, the program continues to run and hence the terminal returns to user program mode.

So far only three Monitor commands have been discussed: START, STOP, and LOGIN. There are, however, many more. (They are described in Appendix B.)

The set of commands enumerated in Appendix B is designed to give the user convenient and comprehensive control over programs. The user can do debugging tasks with commands such as EXAMINE and DEPOSIT;

store and retrieve programs with commands such as SAVE, LOAD, and RUN; and control additional peripherals with commands such as ASSIGN and RELEASE, 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 the EduSystem 50 Monitor. SI is called by the terminal 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 terminal input routines exactly the same way as characters being input to a user program. In either case they are placed in the Multicharacter terminal input buffer until a delimiter is detected. (Delimiters for SI strings are CR, LF, vertical tab, form feed, and rubout if the buffer is empty.) 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 terminal is in SI mode. It is also set whenever a CTRL/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, VT, FF line feed, and rubout). If the input character is a delimiter, a second DDB bit, the "SI command delimited" bit, is set. Also, a scheduler 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. The scheduler checks COMCNT every time it runs. As long as COMCNT is zero, everything is up to date. However, if COMCNT >Ø, 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 the Monitor core in order to inspect DDBs. When it finds an "SI command delimited" bit set, SI knows what called it.

Once it has found what called it, SI reads the command string to find the basic command. SI has a dispatch table for all valid commands. For commands which take arguments, the string is scanned to pick them up. If an error is detected anywhere along the line, SI exits back to the 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 the Monitor.

SI itself is essentially reentrant. It gets its input, the command string, from the Monitor core. SI 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 English language 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 that 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.

9.2 DISK STORAGE AND FILES

Up to now, the EduSystem 50 disk has been mentioned only as a swapping device. For each job, there is a dedicated 4K area on the disk in which the job 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. Ø-7777 contains the System Interpreter Phantom. 1ØØØØ-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 the 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 4K for each possible system job. (A 16 user system thus uses 64K of the disk for swap tracks. This, plus the 2ØK 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.

EduSystem 50 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. 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 the disk for the file. Extending a file puts the new segments on 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. The user 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 the 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 is available 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 the 4K core and the addressed point in the 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 [illegible] 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. How the data of a given file is interpreted by a program is, of course, what matters.

That segments of a file appear to the user to be contiguous is an illusion. Disk segments are, in fact, allocated at random. EduSystem 50 maintains directories in order to remember the segments allocated to certain 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, EduSystem 50 maintains a User File Directory (UFD) that holds the names of all files 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 EduSystem 50 file structure.

The UFD is divided into 8-word entries. For each file there is a single filename entry. The first three words contain the filename (6 characters packed in EduSystem 50 internal format).

Words 4-6 contain information about this file. Word 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 has 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 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 than filenames. The first word contains the account number as a 12-bit binary number, the next two words contain the 4-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 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 initialized. 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 the segments 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.

9.3 TALKING TO THE DISK: THE FILE PHANTOM

Most of the tasks described in Section 9.2 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 the 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, performs all the needed searches of these tables, and executes the basic file commands of CREATE, EXTEND, REDUCE, and RENAME as discussed above. These all happen independently of the 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 the resident Monitor and a disk file. (It is important that OPEN and CLOSE do not affect the disk. 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, Figure D-8.) When a file OPEN command is given, FIP sets up a new file control block in free core. This block is used to hold pertinent information about the open file. A pointer to this file control 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 control 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 that does not have to be preceded by an OPEN. All other file commands operate on internal file numbers rather than filenames. 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 a file but a write-protected bit is set in his file control 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 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. The 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 executes the transfer. (Note, that if it is a write, the write-protect bit in the file control block block is checked first.) If the window is not properly set, the resident Monitor calls FIP to move the window so that it is looking at the specified part of the file. FIP then returns to the Monitor so that it can do the transfer.

FIP is called whenever the 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 EduSystem 50 IOTs. 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, the IOT and all its parameters are placed in 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 FIP 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 is the only parameter. In this case, no IOT Parameter Block is needed. The IOT goes into JOBLNK. (The AC is, of course, stored in another job status block.) If JOBLNK is less than 400, it contains an IOT with the first 4 bits stripped off. If JOBLNK is 400 or greater, it contains the address of a free-core block which contains the IOT.

FIP maintains the Storage Address Table (SAT) which is located in the high end of FIPs 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. (Initializing, 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 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 does not 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 UFDs. 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 Appendix D for a more detailed discussion of the FIP tables.

9.4 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. Most disk transfers go between user fields; resident Monitor never does transfers into field Ø, and only the DECtape handler requests transfers in field 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 4-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 4-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. The resident Monitor uses the retrieval window for that file (calling FIP to move it, if necessary) to figure out the physical disk address. It then 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 on the system disk and the RK05, and card reader transfers, which are not buffered in Monitor core, 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 for queueing requests.

9.5 ASSIGNABLE DEVICES

All EduSystem 50 systems include a high-speed paper tape reader. Some may include optional devices, such as a high-speed punch and 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 the Monitor buffers and the user program. Assignable devices have their slots in DEVTBL just as the terminals do. 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 the 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 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.

Different assignable devices use different methods of buffering their I/O. For example, the paper tape reader, which uses a free-core buffer, is activated by a RRB IOT. Finding the buffer empty, the Monitor puts the user job into an I/O wait state. This clears its reader flag and sets the corresponding bit in the wait mask. It then sets up the reader service routine to read characters into the reader buffer. When the buffer has been nearly filled, the user's reader flag is reset, making the program runnable again. The program then executes successive RRB IOTs to pick up individual characters. When the buffer empties again, the process repeats. The user may clear the buffer by executing an RCB instruction.

Operation of the high-speed punch is very similar. The running program passes characters to the 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. The Monitor does not wait for the buffer to fill. Thus, a user program may overlap execution with punching as long as [illegible] does not fill. The line printer is handled in a manner similar to the high speed punch.

DECtape handling is different. DDBs are set up when they are assigned and returned when released. Since there are eight possible DECtapes, the Monitor reserves eight words in DEVTBL. The same DDBs are used for reads and 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 is only one, regardless of the number of units. At the time a user program requests a transfer, the Monitor starts the desired transport toward the requested block, and the job is put into a wait state. When the block is almost at hand, the Monitor assigns the DECtape buffer to that job, stopping the tape to wait for the buffer if necessary. On a read, the selected block is now read into the buffer, and transferred to the user, either by transferring to his core field or by writing it to his swap track on the disk. Conversely, on a write, the block is moved from the user's core or from his swap track to the DECtape buffer, and then written to the tape.

The RK05 and card reader are similar in that they both lock the user into core and transfer directly to/from his buffer.

Although these are the only peripheral devices supported by the EduSystem 50 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 the Monitor, and one to do the actual device handling. Space in the Monitor is available but not in large quantities.

9.6 ERROR HANDLING

The EduSystem 50 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. A user may have a job do anything that 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 the 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, the Monitor returns an error code to the user program. (For more information, see Appendix B.) 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 non-logic error is the parity error or directory error on a file read or write. This is the result of physical malfunction of the disk, a transfer error occurred either within the file itself or within one of the Monitor's directories.

The second level of error handling also comes when a user program requests something which the Monitor cannot do. For example, the user program 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, the Monitor terminates them, and types 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 IOT, which gives the Monitor an address to JMS 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. The user must clear the error status via a CLS IOT.

The 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. All I/O devices except the system disk have a timer. Each time an I/O operation is started, the timer is set for a number of seconds, depending upon the device. If an interrupt does not occur before the timer times out, the Monitor will signal a hung device. In the case of a terminal printer, the output buffer is simply cleared. All other devices report a system error when hung.

When the punch or line printer hangs, the Monitor reports the error and attempts to re-report it every five seconds until either the device is put on line, or the device is released. In the first case, output continues, and in the second, the buffer is cleared. If SI is called to report a hung device, it will report it only once. If SI continues to be called every 5 seconds, it will simply ring the terminal bell, trying to get the user to do something.

APPENDIX A

UTILITY PROGRAMS

The following programs are used commonly in EduSystem 50. The information given here is meant to be only a quick summary of their use. For more information, refer to Users Guide.

A.1 BASIC

Type R BASIC to execute BASIC. BASIC asks "NEW OR OLD?." Answer OLD to execute a program stored on disk. BASIC asks for the name of the program. Respond with the name if the program is stored under your account number. If the program is stored under account 2, respond with the name immediately followed by an asterisk. Optionally, follow the name with a space and an account number.

BASIC now responds with "READY." You may now add or change any lines simply by typing them, list the program by typing LIST, or run it by typing RUN. To interrupt a running program, type a CTRL/C. To return to the Monitor from BASIC, type BYE.

A.2 CAT

A user may type R CAT to run CAT, and obtain a listing of disk files.

The system manager, logged in under account 1 may type R CAT to run CAT and obtain a listing of all users, their passwords, amount of time used, etc. The accumulated time may be reset by answering "YES" to the question "RESET?".

The system manager may type a R CAT:L to get a disk directory of any user. CAT asks the account number of the directory it is requested to list.

Any user may type R CAT:S to obtain a short SYSTAT.

The system manager may type R CAT:R to reset all users' CPU time.

A.3 LOADER

The LOADER loads BIN format files into core from disk. For input, type the name(s) of the input file(s), separated by commas. For option, specify D if debugging using ODT is desired. Normally, just give a carriage return. The LOADER will not correctly load locations 7767 - 7777. If ODT is used, locations 4 and 7000 to 7777 must be reserved for it.

A.4 LOGID

The system manager (account 1) may define, change, and delete accounts and passwords at will. See Section 6.1, defining accounts and passwords.

A.5 LOGOUT

LOGOUT is run in response to the LOGOUT or KJOB Monitor commands. See LOGOUT under Monitor commands for additional details.

A.6 PIP: PERIPHERAL INTERCHANGE PROGRAM

PIP moves files between paper tape and disk, deletes disk files, or prints them on the line printer. PIP has been replaced by PUTR. However, in case some systems desire to use PIP, here are instructions.

When PIP requests INPUT or OUTPUT, respond with a carriage return only, to specify a paper tape reader, paper tape punch, or terminal. Respond with a filename for a file under your account. Respond with a filename, space, then account number for a file stored under someone else's account.

When PIP requests OPTION, choose from the following list:

B - Transfer a BASIC program file between the disk and the high-speed reader or punch. The response to INPUT: and OUTPUT: indicates the direction of the transfer.

D - Delete the file specified for input.

F - List a BASIC program on the line printer.

K - Load a save format paper tape from the terminal. The Monitor must be patched to enable this option to operate properly, as it normally forces the parity bit on for terminal input.

L - Transfer an ASCII file from the disk to the line printer.

P - Punch the contents of a disk file on the high-speed punch.

R - Read a tape from the high-speed reader and store it as a disk file.

S - Transfer a SAVE format file between the disk and the high-speed reader or punch. The response to INPUT: and OUTPUT: indicates the direction of the transfer.

T - Transfer a file between the disk and the terminal reader or punch. The response to INPUT and OUTPUT indicates the direction of the transfer.

A.7 SYSTAT

SYSTAT may be run to obtain the status of the system by typing SY or SYSTAT. The SYSTAT may be output to the line printer by typing SYSTAT-L.

A.8 PUTR

PUTR is a program designed to transfer information from any EduSystem 50 device to any other EduSystem 50 device, with numerous options for different formats. For further details see Users Guide

A.9 PTLOAD

PTLOAD is TSS/8's version of the Binary Loader. To use PTLOAD, load a binary tape in the appropriate tape reader, and type "R PTLOAD". To "OPTION-", respond with "T" for the terminal (low-speed) reader, or "H" or any other letter for the high-speed reader. When using the low-speed reader, turn it off when the tape reaches trailer code. Binary tapes may not be read from a terminal without patching the monitor.

A.10 GRIPE

GRIPE is a program which allows any user to leave a message for the system operator. To initialize GRIPE, log in under account 3 and type "R GRIPE". GRIPE should print "THAT'S ALL", and return to the Monitor. The initialization is complete.

To use GRIPE, any user types "R GRIPE". GRIPE prints "END WITH ALTMODE", and then allows the user to type his message, after which he should type an ALT MODE (sometimes labelled ESCAPE).

When the operator desires to read the collected messages, he should log in under account 3 and type "R GRIPE". After the messages have all been printed, they will be deleted and GRIPE will be ready to collect more messages.

GRIPE stores gripes in an unprotected file under account 3 named " GRIPE" Any user who discovers this can read or destroy this file if he wishes.

A.11 OTHER PROGRAMS INCLUDED IN THE LIBRARY:

CATALOG may be LISTed under BASIC for a list of some BASIC games and demonstration programs.

PLOT is a FOCAL program to plot a damped sine wave.

HAMURS is a FOCAL game, as is ROCKES

HAMURA and ROCKET are saved images of HAMURS and ROCKES which are run by simply typing "R HAMURA" or "R ROCKET".

WDGAME is a FORTRAN demonstration. It may be used as follows:

```
.R FORT
INPUT - WDGAME
OUTPUT -
INPUT - DATA
OUTPUT -
```

MATRIX is a FORTRAN demonstration which multiplies 2 square matrices.

TYPE is a PALD demonstration program. When assembled, loaded, and started at 4ØØ it prints "Ø123456789".

A.12 TEST PROGRAMS

The following programs are written specifically to run under EduSystem 50 and to test various capabilities of the Monitor. These programs can be used as system confidence tests, or they may be used by service personnel to exercise peripherals without bringing down EduSystem 50.

A.12.1 TSTMEM

This is an EduSystem 50 memory diagnostic. If an error is detected, the diagnostic prints a message or halts.

Execution:

To execute TSTMEM, type "R TSTMEM". The program prints "11" occasionally to indicate that it is running. Otherwise, the program runs until an error occurs, or until it is stopped.

A.12.2 TSTDT

This is an EduSystem 50 DECtape diagnostic which writes and reads random data on random blocks in a random direction with a random current address. Data and status errors are reported, and a status report is available.

Execution:

If a line printer is available and on line, type "R TSTDT:Ln", where n is the number of a DECtape drive which has a scratch tape mounted on it. The drive should be placed in remote, write enabled. If a line printer is not available, type "R TSTDT", and the diagnostic will ask for the desired unit number.

The diagnostic initially confines itself to the first 200 blocks on the tape. After a short while, it prints a status report, and then begins exercising the entire tape.

STATUS REPORTS:
Typing any character other than CTRL/C causes a status report to be printed. If the character is an E, the diagnostic will stop after the status report (the operator may type "START" to continue). The status printed includes the total number of blocks read and written, the number of words of data error the number of status errors, and the Inclusive OR of all status bits returned by the Monitor. These status bits correspond to TC01/TC08 status register B.

ERROR REPORTS:
If a status on data error occurs, the printed information will include the DECtape unit number, status A and B for the transfer, the block number of the transfer, and the buffer address. In addition, if there were data errors, a tally will be printed, followed by the locations in error. The first address in the block should be the block number. This will be at the high end of the core buffer, if the block was read in reverse. The data is printed as it would appear in

if the block had been read in the forward direction. never a read error occurs there is no way of knowing long ago, or in which direction the block was written. the user desires to stop a long error report, he types CTRL/C. This halts printing and resumes testing.

12.3 TSTRK

is is an EduSystem 50 RK05 diagnostic which writes and reads a random number of pages of random data beginning at a random core address and a random sector. Data and status errors are reported.

Execution:

If a line printer is available and is on line, type "R TSTRK:Ln", where n is the number of a disk drive which has a scratch pack mounted on it. The drive should be in the ready position, not write-protected. If a line printer is not available, type "R TSTRK" and the diagnostic will ask for the desired drive number.

The diagnostic exercises the disk until it is stopped. After 4096 transfers, it prints a "PASS COMPLETE" message. To exit from the test, type E, followed by a carriage return. If there is a long data error, printing can be stopped and testing can be resumed by typing CTRL/C.

ERROR REPORTS:

If an error occurs, the transfer parameters are printed. This includes whether the transfer was a read or a write, the unit number, the number of pages in the transfer, the RK05 status, the contents of the AC after the DLAG, the initial sector, and the beginning current address.

The status returned corresponds to the RK05 status register, but will be 0 if no error has occurred. The value returned in the AC, after the DLAG, should be the number of blocks successfully transferred. If the transfer is completed normally, this will be (P+1)/2, where P is number of pages transferred.

In addition, if data errors were detected, the information printed will include the disk sector number where the error occurred, the address within this sector (0-377) and the good and bad data.

A.12.4 TSTPT

This is a test of the high-speed paper tape reader and punc
The test punches the special binary count pattern and reads either the special binary count pattern, OR A ONES AND ZERO TEST TAPE.

Execution:

Type "R TSTPR". The test prints a quick option summary and waits for a command. If a P is typed, the punch begins punching. After a sufficient amount of tape has been punched, type P again to stop the punch.

If an R is typed, the reader starts when the test reads the first non-zero frame, it decides which type of tape is in the reader, and then it continues. If the user wants to stop the reader, he has to type another R. Typing CTRL/C causes the test to halt after releasing the devices.

ERRORS:

If the first non-zero frame, on a tape being read, is not 001 or 377, the test prints a message requesting the paper tape. If an error is encountered within the tape, the expected and read values are printed. Because of the buffering by the Monitor, the physical position of the tape will not be close to the frame in error. If a "HUNG DEVICE" message is printed, the paper tape punch is probably not turned on or is not responding for some reason. This could also occur if the reader was turned off. If the reader hangs (reads to the end of the tape), the message "READER ASSUMED", may be printed. When a device is hung it is not always possible for the test to know whether it was the reader or punch which hung. But if it was wrong, the punch will hang again in a few seconds.

An "ILLEGAL IOT" message probably means that another job owns the punch or reader, and it cannot be assigned. The PC does not necessarily point to the invalid IOT in question.

A.12.5 TSTLPT

This is a test of the ability of EduSystem 50 to output to the terminal and the line printer. The test will handle 72, 80, or 132 column printers or terminals, 64 or 96 characters, and four different patterns.

Execution:

Type "R TSTLPT". The test prints a quick option summary and then waits for a command. If a "T" is typed, the terminal begins printing. If a "T" is typed again, the printing stops. The same is true of "L", and other commands may take some time to take effect.

Typing a 0, 1, 2, or 3 causes the pattern on the line printer or terminal to change. If "T" was typed more recently than "L", the terminal pattern will be affected, and conversely, if "L" was typed more recently than "T", the line printer pattern will be affected.

Typing a 9 causes the line printer or the terminal to use 96 characters. Typing a 6 restores the normal case of 64 character This affects "L" or "T" as above.

Typing a 7 causes the line printer or terminal to use 72 columns; 8 causes the line printer or terminal to use 80 columns; W (for Wide) causes the line printer or terminal to use 132 columns. Again, this affects either "L" or "T" as explained above.

Typing CTRL/C causes the diagnostic to halt and the line printer to be released.

Errors:

The diagnostic itself detects no errors. Printed output should be visually inspected. If the line printer is not on line or does not respond, the monitor prints a "HUNG DEVICE" message.

A.12.6 TSTBAS

This is a test of the ability of EduSystem 50 BASIC to interact with a user.

Execution:

Type the underlined parts of the following dialogue:

```
          .R BASIC

          NEW OR OLD--OLD
          OLD PROGRAM NAME--TSTBAS*

          READY

          RUN
```

At this time, there is a pause for compilation. Then instructions for use of the program are printed. To terminate the test, do the following:

```
           ↑C

           READY

           BYE
           ↑BS
```

A.12.7 TSTFOC

This is an EduSystem 50 FOCAL program which plots a damped sine wave on the terminal, testing terminal output.

Execution:

To execute TSTFOC, type "R TSTFOC". The program prints an asterisk (*). Type G, followed by a carriage return, and the plotting should start. TSTFOC continues until it is stopped.

APPENDIX B

MONITOR COMMANDS

An alphabetical list of all Monitor commands is included here to make it easy to find any particular one. Some are restricted, and may be used only by someone logged in under account 1 or 3.

When typing a command, it is not always necessary to type the entire word. In fact, each command may be shortened as long as that command does not become ambiguous. For example, EXAMINE may be typed instead as EXAMI, EXAM, or even EXA. However, EX or E cannot be used, because there is a command named EXTEND, and System Interpreter (SI) would not know which one was wanted.

All numbers in SI commands are octal, with two exceptions. The word count in the EXAMINE command and disk segment counts in all cases are in decimal.

Commands may be concatenated by putting a semicolon between them; for example, DEPOSIT 0 5000; EXA 0 1; START 0 causes three commands to be executed in sequence. Some commands may be entered while a program is running. To do this, preface them with a CTRL/B. CTRL/B followed by WHERE will allow a user to find out what his program is doing without stopping it.

All commands, terminated by a carriage return, cause SI to be read from track 0 into field 2 to interpret the command. Many commands require File Phantom (FIP) for processing. SI then causes FIP to be read from track 1 into field 2 over SI and executed. When FIP is finished, it must cause SI to be read in over FIP to finish up.

Note that the functions of many of these commands may also be accomplished by having a program execute a UUO, which often results in calling FIP.

B.1 ASSIGN

Purpose

To allow a user to reserve a device. Devices are:

- R - High-speed paper tape reader
- P - High-speed paper tape punch
- L - Line printer
- D - DECtape
- C - Card reader
- K - RK8E

Example:

```
.A R
R ASSIGNED
.A D
D Ø ASSIGNED
.A D 4
D 4 ASSIGNED
```

Note that one may either request a specific DECtape unit or one can request any DECtape unit. The same is true for the RK8E. If a specific unit is not requested, an available unit is assig

How? SI calls FIP to complete this command. FIP checks whether the user will get the device and, if so, sets up a DDB and puts its address in DEVTBL.

B.2 BREAK

Purpose

To find or to change the value of the user's break mask. When in user mode and typing at the keyboard, the user's break mask determines which characters are significant enough to cause the user program to restart execution (if it is waiting for input). See IOT 6400-KSB Appendix C for details.

Example:

```
          .BREAK 4000

          .BREAK
          4000
```

First, the break mask is set to 4000. Then its value is determined.

How? The break mask is kept in the third word of the keyboard DDB. It is retrieved from there or stored there by SI.

B.3 BROADCAST

Purpose

To allow a message to be sent simultaneously to all users. May be used only by a user under account 1 or 3.

Example:

```
        .BROAD THE SYSTEM IS GOING DOWN FOR PM IN 5 MINUTES
        *** THE SYSTEM IS GOING DOWN FOR PM IN 5 MINUTES

        OK
```

The message is sent to everyone, including the sender.

How? The message is simply jammed into all output buffers. SI checks to make sure the account number of the sender is 1 or 3. If free core runs out before the message has been given to all terminals, SI returns the error message "BUSY". Otherwise, the message OK is printed.

B.4 CLOSE

Purpose

To inform the Monitor that the user is finished with a file. See OPEN.

Example:

```
               .CLOSE 0
```

More than one unit can be specified at the same time.

Example:

```
                 .CLOSE 0 1 2
```

How? FIP is called to process the command. FIP simply undoes everything done by an OPEN command.

B.5 CREATE

Purpose

To allow the user to create a new file with the length of one segment. The file must have a name consisting of 1 to 6 characters, the first one being a letter. If a file already exists by that name, it is first deleted. Under account 1, the create command will not delete files (UFD's) nor will it allo duplicated account numbers to be created.

Example:

```
        .CREATE FILE23
```

How? FIP checks for validity of the command, deletes any file named FILE23, makes a directory entry for FILE23, and reserves one disk segment for the file. A protection code of 12 is assigned to the new file (see PROTECT).

B.6 DEPOSIT

Purpose

To allow the user to change any words in the 4K of core at will. The user gives any address, and up to ten (decimal) values in octal to deposit.

Example:

```
        .DEP 1Ø 7ØØ1 6Ø46 5Ø1Ø
```

This deposits a simple program starting at location 0010.

How? As needed, SI either stores the values directly in the user's core area or writes the information to the swap area on the disk.

B.7 DUPLEX

Purpose

To put the user's terminal into duplex mode. Normally, when a user program is being executed, and the terminal is in user mode, characters typed at the keyboard are not printed unless the program causes it. Putting the terminal in duplex mode causes the Monitor to perform this function automatically.

Example:

```
.DUPLEX
```

How? SI sets the duplex bit in the terminal keyboard DDB. This is bit 4 of the first word of the DDB.

5.8 EXAMINE

Purpose

To allow the user to examine his 4K of core at will. Type the initial address first, and then the number of words wanted (in decimal, up to 10), if greater than one.

Example:

```
.EXAMINE 10 2
7021 6046
.EXA 12
5010
```

How? SI either takes the information directly from the user's core or, if necessary, reads the information from the user's swap area.

B.9 EXTEND

Purpose

To increase the size of a file. The file must be opened first (see OPEN). Give the internal file number, and then the numbe of segments (in decimal).

Example

```
.EXTEN 1 10
```

This extends the file, which is presently open under internal file number 1, by ten segments. The segments are added to the end of the file; any previous contents are unaltered. Any files belonging to account 1 (MFD and UFD's) may not be extend

How? FIP processes the EXTEND. FIP reserves the require disk segments by setting bits in SAT, and makes the necessary changes in the user's directory.

B.10 F

Purpose

To get information about a file. The file must first be opened (see OPEN).

Example:

```
                  Protection
                  code
                   ↙
   .F 1
   0003 FILE23 00 12 11
     ↑     ↑    ^       ↖ size
 account   ↑  Extension
              code
        File name
```

This indicates the file currently open under internal file number 1 belongs to user (or account number) 3, is named FILE23, has an extension of 0, has a protection code of 12 (see PROTECT), and has a size of 11 segments (decimal).

How? FIP is called to obtain the needed information, which is then printed by SI.

B.11 FORCE

Purpose

The FORCE command helps the system manager control the other users. If desired, the manager may interrupt or even log out a user. The FORCE command allows the system manager or operator (anyone logged in under account 1 or 3) to connect to any other terminal long enough to issue a command. For example, if the user 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 a console uses the CTRL/B (echoed ↑B) to put the console in Monit mode. Within a FORCE command, ↑ is used to indicate that the next letter typed is a control character. For example, the above command should really be typed:

```
        .FORCE 10 ↑BS; RELEASE R
```

The uparrow followed by BS (not CTRL/B followed by S) acts just like 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 an uparrow followed by BS and semicolon, as shown above. Terminating the force command with a form-feed (CTRL/L) will prevent a carriage return from being sent to the forced terminal.

For example, if the user at K10 complains that his terminal is completely dead, the operator may discover with SYSTAT that he typed a CTRL/S accidentally. To restart this user, the operator may type:

```
.FORCE 10 ↑Q
```

and terminate the command with CTRL/L (Form Feed) instead of a carriage return. Terminating with CTRL/L is also useful when forcing a CTRL/C to a user.

When bringing down the system, the following command will stop most obstinate users:

```
.FORCE 10 ↑B↑BS;K↑Q
```

Care should be exercised. If there is an error in typing the FORCE command, the error message may show up on the forced console, and the user will not know what happened.

How? The forced command is placed into the proper keyboard buffer. If the command includes an uparrow, the next character is changed to a control character.

B.12 KJOB

Purpose

A KJOB is identical to LOGOUT in function. See LOGOUT.

B.13 LOAD

Purpose

The LOAD command allows the user to load the core area with data from a disk file. Often, this file is created by a SAVE command, and has an extension of .SAV. To use load, type (separated by spaces):

a) LOAD

b) The account number the file is under. May be omitted if it is the user's own account.

c) The name of the file.

d) The address within the file at which to start. If omitted, 0 is assumed.

e) The address in core at which to start. If omitted, 0 is assumed.

f) The address in core at which to stop. May be omitted. The transfer will continue until one of three things happens:

   1) The end of core is reached

   2) The end of the disk file is reached, or

   3) The core stopping address (if given) is reac

Examples:

```
                    .LOAD PIP
                    .LOAD 2 SYSTAT
                    .LOAD PIP 200 300 400
                    .LOAD PIP 200
```

The first example reads the file named PIP into the user's core, starting at location zero. The second example does the same task for the program SYSTAT which is stored under account 2. The third example reads 201 words from the file named PIP into core, starting at file address 200 and continuing through core address 400. The fourth example reads the file PIP starting at file address 200 and core address 0

How? SI calls FIP to open the file under internal numbe 3, and then passes a RFILE parameter block to the file handler in the resident monitor.

B.14 LOGIN

Purpose

To notify the Monitor that a person wishes to use a terminal, and to give an account number and password. Type LOGIN, then a space, the account number, a space, the password, and then a carriage return. If the LOGIN command is terminated with a line feed, the login message will not be printed.

Note that the command itself is not printed, to protect the password.

How?	First, SI checks the command for validity. Then FIP is called to set up a number of tables to indicate terminal assignments, what time the user obtained it, and the user's job number, etc.

.15 LOGOUT

rpose

o indicate to the Monitor that the user is finished and ready to eave the terminal. Also, LOGOUT gives the user a number of onvenient options. Type LOGOUT:? for an explanation of options, r substitute one of the following for the ?:

K - to cause the LOGOUT to delete all non-protected files.

L - to list the user's disk directories.

S - to save all non-temporary files, or

I - to individually determine whether to save or delete each of the user's files. Each filename will be printed. Type a P if it is desired to protect this file, an S to save it as is, or a carriage return only to delete the file.

Q - to logout quickly and quietly.

yping no option causes a logout with the default option, which is S.

xample:

```
LOGOUT:I

UNK  .ASC    <12>      1. BLOCKS  :
ORT  .SAV    <12>      6. BLOCKS  : S
OSL  .SAV    <12>      6. BLOCKS  : S
EMP00        <12>      1. BLOCKS  : DELETED
ILE23        <12>     11. BLOCKS  : P

OB  1, USER [ 0, 3] LOGGED OFF K00 AT 21:16:39 ON 20 JUL 74
ELETED   2 FILES (   2. DISK BLOCKS)
AVED  3 FILES (  23. DISK BLOCKS)
UNTIME 00:00:10 (  1. CPU UNITS)
LAPSED TIME 00:06:49
```

B.16 OFF

Purpose

When the manager desires to bring down the system for vario
reasons, the OFF command is given. (The manager must be logged i
under account 1 or 3). Then, anyone who does a LOGOUT cannot LO
unless the account number is 1, 2, or 3. See ON. The manager ca
then broadcast warnings and/or force a LOGOUT.

Example:

```
    .OFF
```

.w? SI sets F1OFFJ in field 1 to 7774, allowing only acco
1, 2, or 3 to log in.

B.17 ON

Purpose

The opposite of the OFF command. See OFF. Used by account
only. The system is restored to its normal state so that any use
may LOGIN.

Example:

```
    .ON
```

How? SI sets F1OFFJ in field 1 to ZERO, its normal value.

3.18 OPEN

Purpose

Whenever files are manipulated by the user, they are identified by "internal file numbers." At any one time, a user may have access to up to four files, with internal numbers 0 through 3. The open command associates the internal file numbers with the actual file on the disk.

Example:

```
.OPEN 1 FILE23
```

Assuming that file FILE23 already exists (see CREATE), this statement now allows the file named FILE23 to be referenced with the internal file number 1.

Account 1 may open any user file, and not be protected regardless of the setting of the file's protection word, provided the file is not open to another user. Account 1 files are always write protected even against account 1.

Account 1 has the privilege of deleting any file which is not in use merely by opening and reducing the file. For example:

```
.OPEN Ø GAMES 14,3;REDUCE Ø 3ØØØ
```

will delete the file GAMES which belongs to account 14Ø3. See also REDUCE command .

B.19 PROTECT

Purpose

Each file has a protection code assigned. This code determines who may read or write the file. The protection word is stored in the disk directory, and includes the protection code and the filename extension, as follows:

```
0   1   2   3   4   5   6   7   8   9   10   11

Filename Extension  Unused  Protection Code
```

Bit 11, if 1, means that the file cannot be read by users whose project number differs from the owner's.

Bit 10, if 1, means that the file cannot be altered by users whose project number differs from the owner's.

Bit 9, if 1, means that the file cannot be read by users whose project number is the same as the owner's.

Bit 8, if 1, means that the file cannot be altered by users whose project number is the same as the owner's.

Bit 7, if 1, means that the user cannot alter his own file, without first changing the protection.

NOTE

A user's project number is the first 2 digits of the 4-digit account number. The system normally sets bits 8 and 10 for the user protection code 12.

The filename extension gives additional information about the file, which is printed in some directory listings. The filename extension codes are:

0	blank	
1	.ASC	ASCII files, such as FORTRAN source.
2	.SAV	Save format files.
3	.BIN	Binary files; must be loaded with program LOADER.
4	.BAS	BASIC source file.
5	.BAC	BASIC compiled program file.
6	.FCL	FOCAL file.
7	.TMP	Temporary file
10	blank	
11	.DAT	BASIC data file.
12	.LST	Listing file
13	.PAL	PAL source
14-17	blank	

The protection word may be set by using PROTECT:

```
.PROT Ø Ø217
```

This changes the protection of the file open under internal file number 0 so that the file has an extension of .ASC, and that it cannot be read or written by any person other than its owner.

This command changes the protection word of the file, currently opened under internal file number zero, to Ø217. Since bits Ø-4 are equal to Ø1, the files have an extension code of .ASC. Since bits 8,9,10 and 11 are set, the file will not be accessible to anyone other than the user. The protect command cannot be used for MFD's and UFD's. See the REN IOT for details.

How? FIP is called to do the actual protect word changing. The protect word is in word four of the name block for the file in the user's UFD.

B.20 R

Purpose

The R command searches the directory of account 2 (Library) for a program and, if found, loads it and starts execution at location 0. If the program name is followed by a number, execution will start at that address instead of at 0.

Example:

```
        .R CAT
```

CAT will be executed, and will list the directory. R CAT is equivalent to RUN 2 CAT.

How? SI calls FIP to open the file for the program, then passes a RFILE parameter block to the file handler in the Monitor to read it in. It then causes execution to start at location 0, or the address specified. The R and RUN commands operate slightly differently than the LOAD command. If the program to be run is shorter than 4K, the unused portion of core may end up containing part of SI rather than what was there before the R or RUN.

B.21 REDUCE

Purpose

To make a file smaller. REDUCE removes segments from the end of the file, leaving the others (if any) intact. If a file is reduced until there is nothing left, the file is deleted completely, including the entry in the user's directory.

Example:

```
        .RED 1 5
        .RED 0 1000
```

The first command causes the file open under internal file number 1 to be shortened by five segments. The second command deletes the file open under internal file number 0, provided its length is 1000 segments or less.

Account 1 files (UFD's) cannot be reduced if there are any users logged in on or using the UFD, or if the UFD owns any files.

How? SI calls FIP to do the REDUCE. FIP finds the first segment of the file to be deleted, removes it from the directory, calculates its bit in SAT, clears that bit, and repeats the whole operation the desired number of times.

B.22 RELEASE

Purpose

To release devices so that other people may use them. The opposite of ASSIGN.

Example:

```
.REL R
.REL D 1
```

The first command releases the high-speed reader. The second command releases DECtape unit 1.

How? Provided the user owns the device, SI calls FIP which zeroes the proper word in DEVTBL, releases the DDB to free core, and charges the user's account for the elapsed time.

B.23 RENAME

Purpose

To allow the user to rename a disk file.

Example:

```
.REN 2 HELPME
```

The file currently open under internal file number 2 is given the name HELPME. The keyboard RENAME command will not rename a MFD or UFD (which would change passwords).

How? SI calls FIP, which changes the name in the user's UFD.

B.24 RESTART

Purpose

To set or determine the restart address.

Example:

```
.RESTART 200
.RES
0200
```

When the program is running and CTRL/C (↑C) is typed, the Monitor causes the program to restart execution at the restart address. Thus, after the first command above has been given, if the user types CTRL/C (↑C) during program execution, the effect will be a CLA CLL and a JMP to location 200. Monitor also clears the terminal buffers when it recognizes the ↑C. The second command determines the current restart address.

How? SI sets the restart address in the user's job status block 0, the 7th word.

B.25 RUN

Purpose

The RUN command searches under an account number for a file. If it is found, it is loaded into core and executed. If the program name is followed by a number, execution will begin at that address; otherwise it will begin at 0.

Example:

```
.RUN MYPROG
.RUN 1234 PROGB
```

The first command requests the program named MYROG to be loaded and executed. Since no account number is given, the user's account number is used to search for the program.

The second command requests execution of PROGB, stored under user number 1234. It is equivalent to LOAD 1234 PROGB; START 0.

How?	SI calls FIP to open the desired file, then passes a RFILE parameter block to the file handler in the Monitor to read it into core. It causes the user to start executing at location 0, or the address specified.

B.26 S

Purpose

Stop the user's program. The terminal is in Monitor mode at this time.

Example:

```
&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRS↑BS
.
```

It is necessary to type CTRL/B, to put the keyboard in Monitor mode, before the S can be typed. If unwanted output is occuring, the user can type CTRL/B twice to stop the output, followed by S and carriage return to stop the program.

How?	SI clears the run bit in the user's STR0.

B.27 SAVE

Purpose

To write out portions of the user's core area to a disk file. The SAVE command has the same format as the LOAD command (see LOAD).

Example:

```
.SAVE NEWPRO
.SAVE PROG2 200 200
```

Be sure that the named disk file exists and is the proper size.

How?	SI calls FIP to open the desired file, and then passes a WFILE parameter block to the file handler in the Monitor to do the requested disk transfer.

B.28 START

Purpose

To begin or continue the program which is already in core.

Example:

```
.START
.START 10
```

The first command is similar to the PDP-8 continue key. A program is continued from the place where a HLT instruction was executed or a CTRL/B followed by S was typed. The second command clears the AC and LINK and begins execution at address 10.

How? The START command, if an address is given, causes SI to alter the user's saved registers to the proper value. The run bit is then set, allowing the job to execute.

B.29 SWITCH

Purpose

To find the value of or set the user's switch register. The switch register is implemented by software to prevent users from having to set the computer's switches. There is no way that the user can determine the hardware SR setting, short of looking.

Example:

```
.SWITCH 3210
.SW
3210
```

The first command sets the switch register to 3210. The second command determines the value from this point. Whenever the user does an OSR, the constant 3210 will be ORed into the AC.

How? Each user's switch register is stored in the job status block 1. SI simply stores the SR there.

B.30 SYSTAT

Purpose

To obtain information about the status of the SYSTEM.

Example:

```
.SYSTAT

STATUS OF TSS/8.24 DEC PDP-8 #1 AT 19:42:29 ON 20 JUL 74

UPTIME 00:27:55

JOB     WHO     WHERE    WHAT      STATUS      RUNTIME

 1      0, 3     K00     SYSTAT    RUN    ↑Q   00:00:09
 2      0, 3     K04     PIP       ↑BS    ↑Q   00:00:08

AVAILABLE CORE  0K   FREE CORE=311

BUSY DEVICES

DEVICE  JOB

K1      2


 219 FREE DISK SEGMENTS
```

For installations with a line printer, type

```
SYSTAT:L
```

How? SYSTAT effectively causes a R SYSTAT.

B.31 TALK

Purpose

To enable messages to be sent between terminals.

Example:

```
.TALK 1 DID YOU HEAR ABOUT JOHN?
```

If terminal 1 is not busy, the above message is printed on the console. If it is busy, SI will return the error message "BUSY." Accounts 1 and 3 are usually allowed to interrupt even when a terminal is busy.

B.32 TIME

Purpose

To get information.

Examples:

```
.TIME
00:00:05
.TIME 0
22:12:38
.TIME 2
00:00:25
```

The first example gives the elapsed processor time used by the user issuing the TIME command since he logged into the system. The second example gives the time of day. The third example gives the amount of processor time used by job 2 since the user logged in.

How? SI takes the information from the proper location, converts and prints it.

B.33 UNDUPLEX

Purpose

To take the user's terminal out of DUPLEX mode. (See DUPLEX).

Example:

```
.UNDUP
```

How? SI clears the DUPLEX bit in the user's keyboard DDB.

B.34 USER

Purpose

To find information about a selected job.

Example:

```
.USER
JOB Ø1  [ØØ,Ø3]  KØØ
.USER 2
JOB Ø2  [Ø1,23]  KØ4
```

The first command informs the user that he is job 1, account 3, and is at console 0. The second command asks about job 2, which is being used by account 123 at terminal 4.

How?   SI takes the information from various tables in core.

B.35 VERSION

Purpose

To determine the version number of the Monitor currently running.

Example:

```
.VERSION

TSS/8.24
```

How?   SI prints the answer.

B.36 WHERE

Purpose

To determine the status of the user's program.

Example:

```
.WHERE
SW=3210 PC=ØØØØ L=Ø AC=ØØØØ MQ=56Ø2
```

All registers relevant to the user are printed. SW is the switch register (see SWITCH). This command is useful when used in conjunction with a CTRL/B, it enables the status of an executing program to be determined without stopping.

How? SI takes the information from the user's job status block 1.

APPENDIX C

UNIMPLEMENTED USER OPERATIONS (UUOs or user IOTs)

This Appendix includes a numerical list of all the valid UUOs.

Whenever an IOT is executed by a user, it causes a trap into the tor. The Monitor looks for the IOT in a table, and one of several gs happens.

If the IOT is not found in the table, the Monitor simply returns he user's program; the IOT functions as a NOP.

If the IOT is resident - i.e., if code to handle the IOT is in - the Monitor executes the proper routine.

If the IOT is non-resident, FIP must be called. If the IOT is t- i.e., if all information transfer is through the user's sters - the IOT is ANDed with 0377 and stored in JOBLNK in the 's job status block 1. If the IOT is a long non-resident IOT, AC will contain the address of some parameters. These parameters, g with the IOT, are transferred to a free block, the address of the block is placed in JOBLNK, and FIP is called. Note that free always starts above address 377 so that FIP can tell whether NK contains a free core pointer or an IOT. If FIP cannot figure why it was called, it executes a HLT.

6004 - GTF - Get Flag (PDP-8/E only)
The link is placed into AC0, and the EAE GT flag (if present) is placed into AC1. The rest of the AC is cleared.

6005 - RTF - Restore Flags (PDP-8/E only)
AC0 is placed into the link, and AC1 is used to set or clear the EAE GT flag. (if present) The AC is not changed.

6006 - SGT
This UUO causes a skip if the GT flag is set. Applicable only to 8E systems with EAE.

6010 - RRS - Read Reader String

Before executing the RRS, load the Load AC with the address of a 2-word block, where:

Word 1: is minus the number of characters to be transferred.

Word 2: is the address of user core, minus one.

This starts the transfer. These characters will be read from the high-speed reader and placed in the user's buffer. The AC is cleared by RRS. The two words are incremented according to the number of frames read. A system error is generated if the tape runs off the end. See 6431-SEA.

6011 - RSF - Skip on Reader Flag

The next instruction is always skipped.

6012 - RRB - Read Reader Buffer

The next character from the high-speed reader buffer is ORed into the AC. If the buffer was empty, the reader is started and this command is not executed until the buffer is nearly full.

NOTE

Successive RRBs will not retrieve the same character.

6014 - RFC - Reader Fetch Character

This instruction performs no operation.

6016 - RRB RFC

Functions as a RRB.

6017 - RCB - Reader Clear Buffer

This IOT causes the high-speed reader buffer to be cleared of any frames read from the tape but not yet transferred to the user's core. This is useful between tapes when reading more than one tape.

6020 - PST - Punch String

Before executing PST, load the AC with the address of a 2-word block where word 1 contains the negative of the number of characters to punch (word count), and word 2 contains beginning address minus 1 of the string to be punched (current address). The block of data will be punched and control returned to the IOT+2 with clear AC. If the PST does not punch all the characters, control is returned to the IOT+1.

NOTE

On most string IOTs, the following programming sequence may be used:

```
                TAD       ADDR
                PST
                JMP       .-2
                    .
                    .
                    .
      ADDR,     TWOWD
      TWOWD,    -10
                BUF-1
```

As characters are placed into the punch buffer, the word count and current address are updated. If word count reaches zero, the string IOT skips, going to the instruction following the JMP. However, if the buffer fills up, the Monitor returns control to the JMP.-2, causing the IOT to be retried. If desired, the JMP.-2 may be replaced with a jump to other processing, which can later return to retry the string IOT. This allows a program to overlap I/O to a greater extent.

6021 - PSF - Skip on Punch Flag
The next instruction is always skipped.

6024 - or 6026 - PLS - Punch Load Sequence
The AC is placed into the punch buffer, but not cleared. Note that bit 10 of a PLS is ignored (6022 is a NOP).

6030 - KSR - Read Keyboard String
Execution of this instruction initiates a transfer of one or more characters from the user's keyboard to a designated core area. Before executing KSR, load the AC with the address of a 2-word block, where:

Word 1:	negative of the number of characters to be transferred.
Word 2:	address of the core area into which characters are to be placed, minus one.

The transfer is terminated when either:

a) the indicated number of characters have been input, or,

b) a delimiter is seen. At the end of the transfer, the word count and core address are updated and the AC is cleared.

6031 - KSF - Skip on Keyboard Flag

Operation - the next instruction is skipped if the keyboard flag is set. The flag is set whenever a delimiter character is typed. If the KSF does not skip, and is followed by a JMP.-1, the user will be put into a wait state until a delimiter is typed.

6032 - KCC - Clear Accumulator.

6034 - KRS

A character from the keyboard buffer is ORed into the user's AC. If none are available, the job is put to sleep until a delimiter is typed. User will be put into a wait state until a delimiter is typed. See 6400 - KSB.

6036 - KRB - Read Keyboard Buffer

A combination of the 6032 and 6034 instructions.

6040 - SAS - Send a String

Causes a block of data to be output to the user's terminal.

Before executing an SAS, load the AC with the address of a 2-word block, where:

Word 1:	contains the negative of the number of characters to be sent.
Word 2:	contains the address -1 of the first word of the string.

The characters are stored one per word right justified starting at the address specified by word 2. Upon execution of SAS, the system takes only as many characters as will fit into the output buffer. It then makes the appropriate adjustment to word 2 to indicate a new starting address and to word 1 to indicate the reduced character count. It then returns to the instruction following the SAS. If the character count is reduced to zero, the instruction following SAS is skipped. The instruction following the SAS should contain a JMP.-2 to continue the block transfer of terminal characters. The AC is cleared by SAS.

See 6020 - (PST) for sample string programming.

6041 - TSF - Skip on Teleprinter Flag
The next instruction is always skipped.

6042 - TCF - no operation is performed.

6044 - 6046 - TLS - Load Teleprinter Sequence
The contents of the AC are placed into the user's teleprinter buffer.

6200 - CKS - Check Status
Load the AC with the address of a 3-word block and execute the CKS. Upon return, the AC will be 0. STR0, STR1, and the device status register will be placed into the 3-word block. The formats of these registers are:

STR0 Bits

0	Run Bit	User program is in the run state
1	Error Enable	Program handles its own errors
2	JCOMBD	Program was compute bound
3	JSPEEK	User has R privilege
4	JSACC	User is privileged account
5	JSIOT	System use only
6	JSIOTC	System use only
7		Not used
8	JSINER	System use only
9-11	Error Code	System detected error condition: 1 Illegal IOT 2 Swap read error 3 Swap write error 5 Disk file error 6 Hung device

STR1 Bits

0	Timer	Time is up
1	File 0	Internal file 0 is not busy
2	File 1	Internal file 1 is not busy
3	File 2	Internal file 2 is not busy
4	File 3	Internal file 3 is not busy
5	Keyboard	There is a delimiter in the input buffer
6	Line Printer	Output buffer is not full
7	Teleprinter	Output buffer is not full
8	Reader	Character in reader buffer
9	Punch	Punch buffer is not full

10	Error	System error detected, code in bits 9 through 11 of STR0
11	Wait	Job is not waiting.

Device status register: See IOT 6772-RDS- for details.

6400 - KSB - Set Keyboard Break

This performs the same function as the Monitor BREAK command.

Operation: Rather than activate a user's program to receive each character as it is typed, EduSystem 50 accumulates input characters until a certain character(s), is seen. To tell the Monitor which characters to look for (these characters are referred to as delimiters), load the AC with a 12-bit mask before executing a KSB. For each bit set in the mask, the Monitor considers the corresponding character or characters to be delimiters.

Bit	Specifies
0	0 = check rest of mask 1 = any character is break
1	301-332 (all letters)
2	260-271 (all numbers)
3	211 (horizontal tab)
4	212-215 (line feed, vertical tab, form feed; RETURN)
5	241-273 (!"#$%&'()*+,-./:;)
6	240 (space)
7	274-300 (<=>?@)
8	333-337 ([\]↑←)
9	377 (RUBOUT)
10	375 (ALT MODE)
11	any characters not mentioned above.

Alternatively, clear the AC and execute the KSB. The Monitor will return the current value of the user's break mask in the AC.

6401 - SBC - Set Buffer Control

SBC permits the user program to clear its terminal input and/or output buffer. Before executing SBC, set bits 0 and 1 of the AC as indicated below:

Bit 0 = 1 Clear output buffer.

Bit 1 = 1 Clear input buffer.

6402 - DUP - Duplex
Performs the function of the DUPLEX Monitor command. No user registers are affected.

6403 - UND - Unduplex
Performs the function of the UNDUPLEX Monitor command. No user registers are affected.

6405 - CLS - Clear Status
Load the AC with the address of a 3-word block. Any bits set in this 3-word block will be cleared in the user's STR0, STR1, and device status register. Use this ICT with caution.

6406 - SEGS - Segment Count
The number of available disk segments is returned in the AC.

6411 - URT - User Run Time
Load the AC with the address of a 3-word block; where word 1 contains the number of the job for which the run time is sought. The run time is returned in the last two locations of the block. If job 0 is specified, the run time of the current job is returned. The AC is cleared.

6412 - TOD - Time of Day
Load the AC with the address of a 2-word block. The value of the system clock will be placed in the two locations, and the AC will be cleared.

6413 - RCR - Return Clock Rate
The number of clock-ticks per second (ten decimal) is returned in the AC.

6414 - DATE - Date
Returns the date in the AC. The format of the date is ((YEAR-1974)*12+(MONTH-1))*31+DAY-1. This number will overflow in 11 years, 4 days; Jan. 4, 1985 is the last day which will work without changing the base year.

6415 - SYN - Quantum Synchronization
This instruction causes the scheduler to allow any other program to run. When this program is restarted after the SYN, it will have a full quantum (200 ms) of execute time without being swapped out.

6416 - STM - Set Time

Load the AC with a number. The job will be suspended for the number of seconds in the AC. The user's job is put into a wait state, and the two's complement of the AC is placed into CLKTBL, where it is incremented once a second. When it reaches zero, the job is allowed to run again.

6417 - SRA - Set Restart Address

This command allows the user to specify an address to which control is transferred when CTRL/C is typed on the user's console. Load the AC with the restart address and execute SRA. If CTRL/C is detected, the program's input and output buffers are cleared, the AC and Link are cleared and control goes to the restart address.

This function is also performed by the RESTART Monitor command.

6420 - TSS - Skip on EduSystem 50

The next instruction is skipped and the current version number of EduSystem 50 is placed into the AC. This instruction is useful in programs which may run under EduSystem 50 and also under other operating systems, where it will function as a NOP.

6421 - USE - User

Returns in the AC the number of the current job.

6422 - CON - Console

Returns in the AC the console number assigned to the job whose number was in the AC. If that job does not exist, -1 is returned.

6423 - PEEK - Peek

Allows the user to inspect the Monitor core; fields 0 and 1. Load the AC with the address of a 4-word block where

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

The specified Monitor information will be transferred to the user's core, and the AC cleared.

PEEK is a privileged IOT. Privileged IOTs may be executed only in 2 cases: If the account number of the user is 1 through 3, or if the IOT is executed by a library program. The privilege for the second case is enabled by the R, SYSTAT, LOGOUT, and KJOB commands, and is cleared every time SI is entered. The privileged IOTS are allowed if either privilege bit in STR0 is set; if not, the IOT generates an error condition.

6430 - SSW - Set Switch Register

The content of the AC is stored in the user's switch register, and the AC is cleared. The Monitor command SWITCH performs the same function.

6431 - SEA - Set Error Address

This instruction allows the user to specify an address to which control is transferred in the event of a system error. Load the AC with an address before executing SEA. If a system error is detected, the Monitor simulates a JMS to the error address. The program counter is stored in the error address and control transferred to the error address +1. AC, Link, and input/output buffers are not affected. The error code of the system error is in STR0 bits 9-11.

The error routine must read these bits (by a CKS) to determine the cause of the error, then clear them by means of a CLS.

The only error code that occurs in normal system usage is due to a hung device. The error occurs, for example, when 1) the user attempts to use a punch not already turned on or, 2) allows the paper-tape reader to run off the end of a tape. The illegal IOT error probably means that an assignable device IOT was executed without the device first being assigned. Swap and file errors occur if a hardware error is detected while the Monitor is swapping user programs, or reading or writing file directories. These are system malfunctions from which there is no recovery.

6440 - ASD - Assign Device

If the device specified by the content of the AC is available, it is assigned to the user program and the AC is cleared. Otherwise, the job number of the device is placed in the AC. If the device does not exist, 7777 is returned in the AC.

4000	Paper-tape reader
4001	Paper-tape punch
4003	Line Printer
4004	Card Reader
4005+N	DECtape unit N, N=Ø-7
4015+N	RK8E drive N, N=Ø-3

The assignment is in effect until a corresponding REL instruction or LOGOUT.

The same function is performed by the ASSIGN Monitor command.

NOTE

The device number for the line printer has been changed to 4003 from 4002.

6442-REL-Release Device
The device specified by the contents of the AC is released (providing it was owned by the user executing the REL). The AC is cleared. Releasing a device makes it available to other users.

The same function is performed by the RELEASE Monitor command.

6600-REN-Rename a File
REN is used to change the name of a file. Load the AC with the address of a 4-word block where:

Word 1:	contains the internal file number associated with the file whose name is to be changed.
Words 2-4:	contains the new name. This name is in 6-bit characters and packed two in a word.

The same function is performed by the RENAME Monitor command.

When executed under account 1, the REN IOT has a special function. Load AC with the address of a four work block, where:

Word 1:	Contains the account number of an existing user
Words 2-3:	Contains the desired password for that account, and
Word 4:	Contains the desired disk quota for the user.

The UFD to be renamed need not be open. The quota is placed in the word erved for the protection code in files. For UFD's, bits Ø through 5 of the ota are reserved for future use. Bits 6-11 contain the disk quota for that ount divided by 25. The quota word for the MFD contains the grace quota. This a 12-bit number (not divided by 25) which determines how many segments over quota a user may extend his files.

Upon return from all file IOTs, the AC is either cleared or contains of the following error codes:

4000	There was no file open on the specified internal file number. On an open command, this error indicates the open failed because of a l ck of table space or free core.
4400	Attempting to alter a file which is open to another user (or to one user?)

for a user whose directory is full, or who has reached or exceeded his disk quota.

5400	Bad directory
6000	File protection violation
6400	Invalid file name
7000	Attempting to open a nonexistent file
7400	The disk is full.

01 - OPEN - Open a file

Open is used to associate a file with an internal file number. This is necessary because all file operations are in terms of internal file numbers. Before executing the OPEN IOT, load the AC with the beginning address of a 5-word block, where:

Word 1:	contains the internal file number.
Word 2:	contains the account number of the file owner. If 0, the account number of the current user is specified.
Word 3-5:	contain the name of the file to be opened. This name is in 6-bit characters packed two to a word.

If there was another file associated with the internal file number beford the execution of the OPEN IOT, it is closed automatically. This is done before the new file is associated with the internal number. Account 1 may open any used file and not be protected, regardless of the setting of the file's protection word However, account 1 files, (MFD,UFD'S), are always write-protected even ag account 1.

02 - CLOS - Close a file

CLOS terminates the association between files and their internal file numbers. Before executing CLOS, load the AC with a selectio pattern for the internal file numbers whose associated files are to be closed. The file is closed if bit I is 1, where I = bit 0,1,2

The same function is performed by the CLOSE Monitor command.

3 - RFILE - Read file and - 6605 - WFILE - Write file
Once the association of a file with an internal file number has been made, these IOTs allow the actual file reference to be made. re illegal on a file that has not been opened(associated with a nternal file number).

ad or write a file, load the AC with the address of a 6-word
, then execute the IOT. The format for the 6-word block is:

d 1: contains the high-order file word address

d 2: contains the internal file number.

d 3: contains the negative of the number of words for the operation. This number is either the number of words to be read or the number of words to be written.

d 4: contains a pointer to the beginning address -1 of a buffer located in the user program. On a read operation this buffer receives the information from the file; on a write operation this buffer holds the information that is to be sent to the file.

rd 5: contains the least significant 12 bits of the initial file word address to begin the operation.

rd 6: contains an error code:

0	If no error
1	If parity error
2	If file shorter than word count
3	If file not open
4	If protection violated

e read or write begins at the word specified by words 1 and 5.
For example:

```
                TAD     X
                WFILE
                    .
                    .
                    .
        X,      .+1
                Ø
                1
                -2ØØ
                6477
                2ØØ
```

ans: write 200 (octal) words starting at word 200 of the file associat
th internal file number one from a core area starting at location 6500.

After completion of the transfer, the word count (word 3) and core address (word 4) are updated. If an error was detected the appropriate error code is placed in word 6.

6604 - PROT - Protect a file
To use, load the AC with the desired protection word in bits 0-4 and 7-11, and the internal file number of the file to be protected in bits 5-6. The meaning of the bits is explained in the PROTECT Monitor command. (See 6600-REN for error conditions.) For directories, the REN IOT must be used.

6605 - WFILE - Write a File
WFILE uses the same calling form as RFILE; see 6603 - RFILE.

This command performs the same function as the SAVE Monitor command.

6610 - CRF - Create a File
The user can request the system to create a new file of one segment. The user provides the new name for the file. Load the AC with the beginning address of a 3-word block, where:
Word 1 through 3: contains the 6-character name.
If there is some reason why the request cannot be granted, the system will return a non-zero error code in the AC. (see 6600-REN for error conditions.) The protection code of a newly created file is 12. When account 1 creates a file, the two accounting words are zeroed by file; as is the file itself.
This command performs the same function as the CREATE Monitor command.

6611 - EXT - Extend a file

To extend the length of an existing file, that file must be currently open. Load the AC with the beginning address of a 2-word block, where:

Word 1:	contains the internal file number of the file to be extended.
Word 2:	contains the number of segments the system should append to the file.

When a fi is to be extended, a check is made to see if the entire extend wil succeed. If not, the extend will not be started. If an extend auses a users total file size to exceed his quota, the Monitor wi l allow that file to be extended only until the quota has been exce ed by the grace quota. The first time that any file is extended s that its owning account is over quota, the Monitor will print an i formational message such as:

MY FILE EXCEEDING DISK QUOTA

Where MYFI E is the name of the file being extended.

When a use has reached his quota, he may no longer create any file. An attempt do extend an account one file (MFD or UFD's) will always fail.

If, for so e reason, the request to extend a file cannot be granted, the AC wil contain 4000, 4400, 6000, or the number of segments it failed to ppend. See 6600-REN for error conditions.

This comma d performs the same function as the EXTEND Monitor command.

6612 - RED - Reduce a file

To re uce the length of an existing file, that file must be currentl open. Load the AC with the beginning address of a 2-word block where:

Word :	contains the internal file number of the file to be reduced.
Word :	contains the number of segments to be removed. If negative, the file will be deleted.

This reque t is granted unless the file to be reduced is currently opened to another user or if the file is write protected against the user. See 6600-REN f r error conditions. A UFD can only be reduced if the account is inacti and owns no files. Reducing a UFD always causes it to be deleted if word 2 is non-zero.

6613 - FIN - File Information

FINF nables a user program to determine what file, if any, is asso ated with an internal file number Load the AC with the beginning dress of a 7-word block, where:

Word l:	contains the internal file number for which the user program wishes information.
Word 2-7:	contain the information that the system returns after executing FINF.
Word :	contains the account number of the owner, or zero, if no file number, that is, the file is not open.
Word 3-5:	contain the name of the file in 6-bit code.

ord 6: contains the protection code. See the Monitor command PROT
(Disk quota if directory)

or 7: contains the number of segments which compose the file.

is command performs the same function as the F Monitor command.

514 - SIZE - Return segment size

This UUO sets the user's AC to 0400, the size of a disk segment.

514 - LIN - Log In
When SI calls FIP with a 6614, FIP will perform a LOGIN. A prog
cannot execute this IOT (see SIZE).

515 - LOGOUT - Log out

This is a triple purpose IOT. If the AC is zero, it will return the number of users logged in with the same account number as the user executing the IOT. If the AC contains the job number of the user's job, the user will be logged out. All assigned devices ar released, and user's terminal becomes inactive. LOGOUT is a privileged IOT. See 6423 - PEEK for details. If account 1 executes the IOT with AC equal to negative job number, FIP will reset all CPU and device time accumulators in the MFD; this is used by the Reset command in CAT.

516 - WHO - Who

The account number and password of the current job are returned to the 3-word block whose address is in the AC and the AC is cleared.

517 - ACT - Account

Load the AC with a job number. The account number of that job is returned. If the AC is 0, the account number of the current job is returned. If the requested job does not exist, 0 is returned.

532 - RCRA - Read card Alphanumeric

534 - RCRB - Read Card Binary

536 - RCRC - Read Card Compressed

Load the AC with the address munus 1 of an 80-word buffer. A ca is read and the data is put into the buffer in the same form as corresponding hardward IOT. The UUO returns in the AC the numbe characters successfully transferred to the user's buffer. (See 6772 - RDS.)

- LST - Line printer Send-a String

Performs the same function as a 6020 - PST, except that the output goes to the line printer.

- LSF - Line printer Skip on Flag

The next instruction is always skipped.

- 6664 - LPC - Line Printer print

The contents of the AC are loaded into the line printer buffer. The AC is not cleared. Note that bit 10 of the IOT is ignored; 6662 is

3 - DLAG - Disk Load Address and go

Allows the user to read or write on the RK8E. To use, load the AC with the address of a three word block, where:

```
Word 1:     Bit 0 = 0 for a read,
                  = 1 for a write
            Bits 3-8 contain the number of pages to
                     read/write, 1 to 40.
            Bits 9-10 contain the drive number 0 to3.
            Bit 11 contains the high order sector address.
Word 2:     contains the core buffer address minus one,
Word 3:     contains the low order sector address.
```

n return, the AC contains the number of blocks transferred. To determi
or conditions, see 6772-RDS. The disk transfer is made in 400 (octal)
ks. Each RK05 drive contains 14540 (octal) blocks. To specify the
ial block number, the high order bit goes into word 1 of the paramete
k, and the remaining 12 bits into word 3. If the transfer requests ar
number of pages on a write, the last page of the last block on the dis
l contain zeros. Upon return, the AC contains (P+1)/2, where P is the
er of pages successfully transferred.

4 - DTXA - DECtape go
Load the AC with the address of a 3-word block, where:

```
Word 1:      Contains the unit, direction, and function
Word 2:      Contains the block number
Word 3:      Contains the core buffer address minus one.
```

s 0-2 of word 1 contain the unit number. Bit 3 of word 1 should be set
R/W in reverse. Bits 6-8 of word 1 should be 2 to read, 4 to write.

n return, none of the parameters are altered, and the AC is cleared. T
should execute a 6772-DTRB instruction to find out whether the transf
successful.

Note that this allows the user to read or write a block in the reverse direction. The user must be aware that if a block is written in one direction and read back in the opposite direction, the order and contents of the data words will be changed.

771 - DTSF - Skip on DECtape flag. The next instruction is always skipped.

772 - DTRB - Read DECtape Status B, or

772 - RDS - Read Device Status
The information obtained pertains to the RK8E, DECtape, or Card Reader, depending on which was most recently used. The contents of the device status register are:

RK8E: RK8E Status Register

Bit 0:	Control done
1:	Heads in motion
3:	Seek fail
4:	File not ready
5:	Busy error
6:	Time out error
7:	Write lock out error
8:	CRC error
9:	Data request late
10:	Drive status error
11:	Cylinder address error

DECtape: TC08/TC01 Status register B

Bit 0:	Error flag
1:	Mark track error
2:	End of tape
3:	Select error
4:	Parity error
5:	Timing error
11:	DECtape flag

he status register for DECtape may also include 4ØØØ or 4ØØ1. These
oftware generated errors such as block number out of rang.

Card reader: The device status register contains the address of
last word of data transferred to the user's buffer. In addition,
device status register may contain 7777. This indicates that CTRL
followed by S was typed while a DECtape or RKØ5 transfer was in
progress, and the transfer was not finished.

73 - DTSF DTRB -

The status is placed into the AC and the next instruction is skipp

APPENDIX D
DETAILS OF MONITOR'S DATA BASE

INPUT/OUTPUT DATA BASE

All I/O, except for the disk, is controlled from a single, fixed-
gth table, DEVTBL. Actual data about the status of each device is
d in a Device Data Block (DDB). DDBs are dynamically assigned
cks of free core. The actual data to be transferred is contained in
fers. In the case of terminal I/O, the high-speed reader and punch
the line printer, these buffers are dynamically assigned blocks of
e core. One or more (linked) blocks of free core make up a buffer.
minals are considered to be two devices: a keyboard and a tele-
nter. Each as a DDb, and each has its own buffer. Some of the assign
ices, which have higher data rates, do not use dynamic core buffers.
e DECtape uses a fixed 201 word buffer, the disks and card reader
nsfer directly to and from the user's core area.

The tables, DDBs, and buffers are linked together by pointers.
TBL is, in fact, a table of pointers. If a device is inactive (a
rminal not logged in or other devices not assigned) the corresponding
ble entry is zero. If the device is in use, the table entry is a
inter to its DDB. The DDB for each device also contains pointers,
e fill, and the empty pointer. The fill pointer points to where the
xt character to be put into the buffer should go; it points to the
ead" of the buffer. The empty pointer points to the next character
be taken from the buffer. Each buffer block contains in its first
rd a pointer to the next block. The last block in the buffer contains
fill count indicating to which position in that block the next char-
ter should go. Figure D-1 shows the relationship of tables, DDBs,
d buffers.

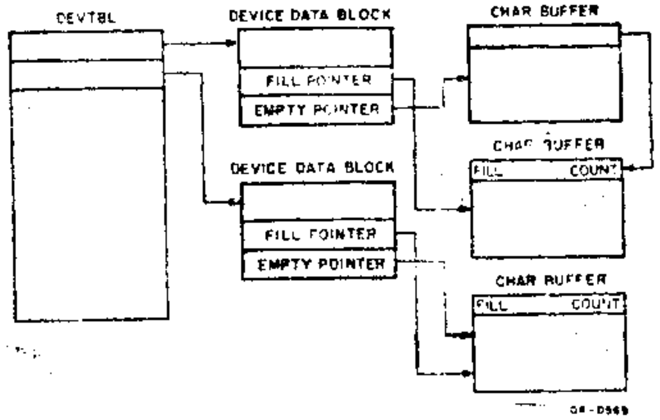


Figure D-1. Relationship of DEVTBL, DDBs, and Buffers

DEVBL is set up with the terminal entries first, the entries for the reader, punch, an unused entry, the line printer, the card reader second, eight entries for DECtape, four entries for RK05 third, and finally a 7777 terminator. The number of entries for terminals, and hence the size of the table, is dependent on the configuration parameters specifying the number of terminals. DEVBE marks the beginning of the assignable device section of DEVBL, which always contains 17 entries even though all these devices may not be included in the system. All slots in DEVTBL which correspond to non-existent devices are filled with dummy pointers to prevent assignment. See Figure D-2.

Device Data Blocks are always 8-word blocks assigned from free core. The DDBs for the assignable devices exist for as long as the device is assigned. For all DDBs, bits 7 through 11 of word zero contain the unit number. Bits 7 through 11 of word 1 contain the job number which owns the device. Word 3 contains the time at which the device becomes active. This 12-bit time is taken from bits 3 through 11 of CLK2, and bits 0 through 2 of CLK1. The use of the remainder of the DDB depends on the particular device.

There are a number of status bits in word zero of the keyboard DDB. The XON bit is set when a buffer is almost full and XOFF is sent to the terminal. When the buffer is emptied, XON must be sent. SI is set to indicate that the terminal is in Monitor mode. DUP is set to indicate that the terminal is in duplex mode. SICOM, when set, indicates that the user has just finished typing a command to SI.

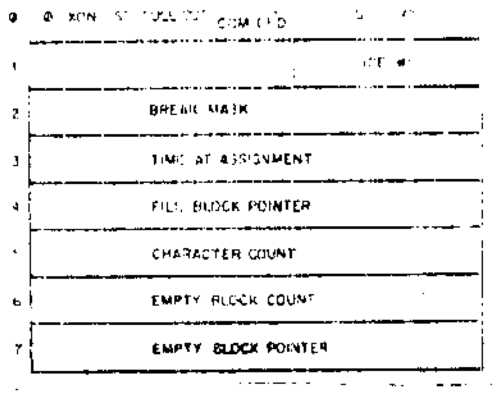


Figure D-3. Keyboard Device Data Block

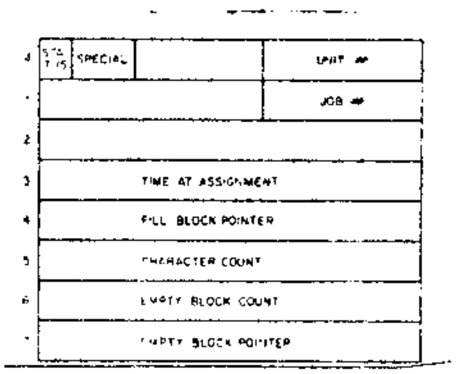


Figure D-4. Device Data Block-Teleprinter, Reader, Punch and Line Printer.

Free core buffers are packed 10 charachters to a block. Characters 1 throught 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. See Figure D-5.

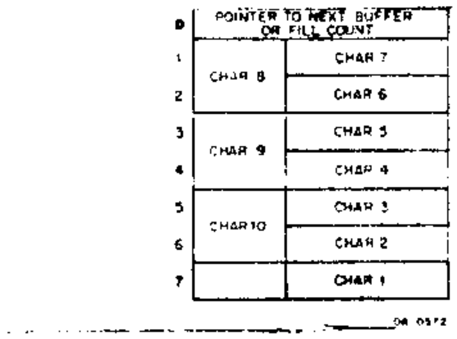
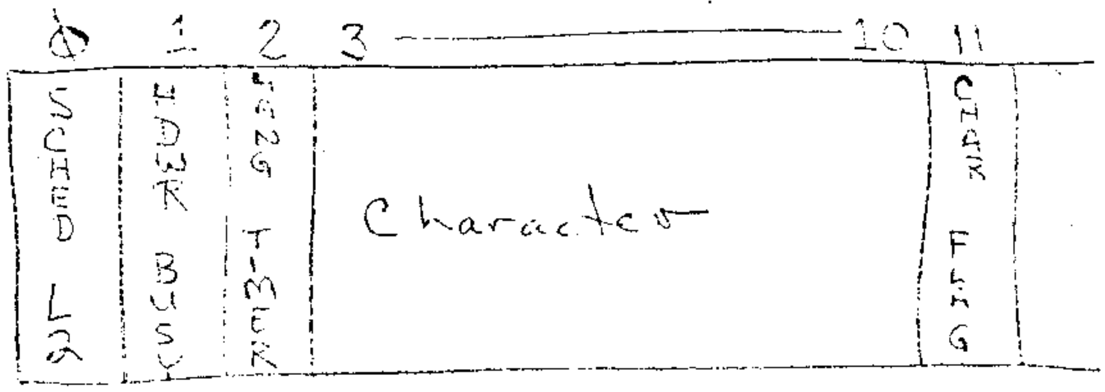
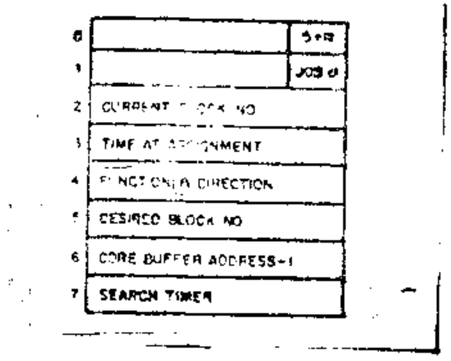


Figure D-5. Character Buffer

WORD FORMAT OF OUTPUT BUFFERS for TTY, Punch & Printer



The DDBs for DECtape and RK05 are shown in Figures D-6 and D-7



igure D-6. Device Data Block - DECtape

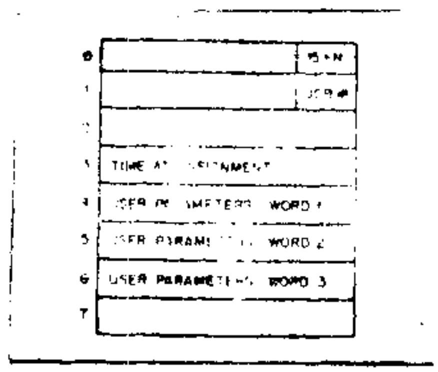


Figure D-7. Device Data Block RK05

2 USER PROGRAM STATUS

All job status information is based on a single, fixed-length
ble in Monitor core, JOBTBL. JOBTBL has a 1-word entry for each
ssible job. If the job does not exist, that is, no one is logged in
that job, the corresponding entry in JOBTBL is zero. If that job
es exist, the entry contains a pointer to the first of three (linked)
b Status Blocks. These contain complete information about the running
ate of that job. For each file open to that job, there are two
ditional blocks; one additional block contains information about the
le, and the other additional block indicates where it is on the disk.
ile a file transfer is in progress, still another block exists which
ntains parameters for the transfers. Finally, when executing an IOT
ich requires a FIP call, a block may be set up to pass the parameters
e Figure D-8.

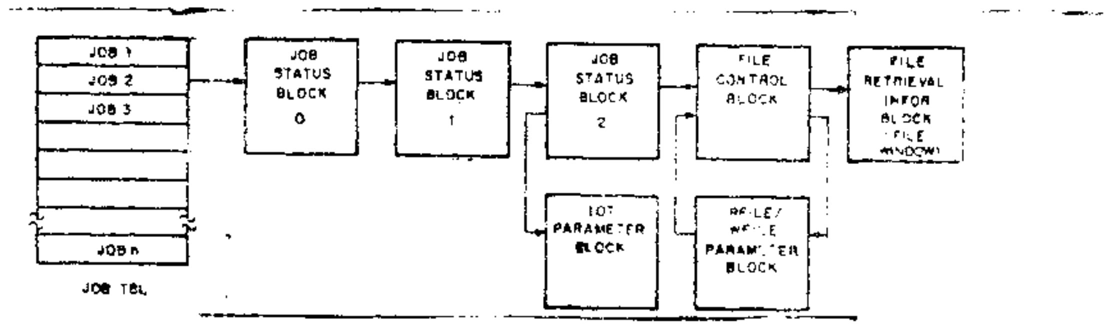


Figure D-8. Job Status Information

The three job status blocks exist for all jobs They contain th saved state for the job, AC, PC, LK, and the EAE registers They als contain the status of the job's I/O. See Figure D-9.

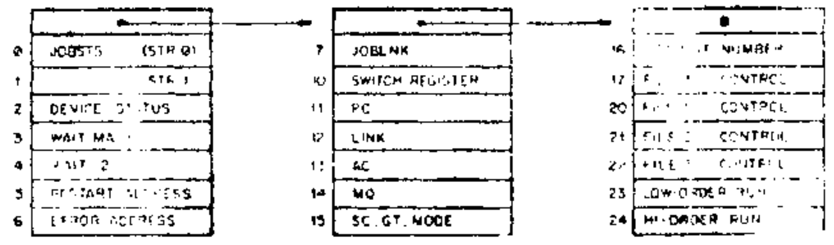


Figure D-9. Job Status Blocks

STRO contains status bits which are not directly associated with I/o. STR1 contains bits which may be considered flags. They are set and cleared according to whether the associated device is ready or not ready. The "wait mask" masks STR1. When a job is waiting for a device, a single bit in the wait mask, corresponding to the device bit in STR1, is set. When that device finishes, its bit in STR1 is set. STR1 and the wait mask are "ANDed" together, and if the result is non-zero, the scheduler knows that the job should be run. When a job is not waiting, bit 11 (the dummy wait bit) is set in both STR1 and the wait mask, allowing the job to run. See Figure D-10.

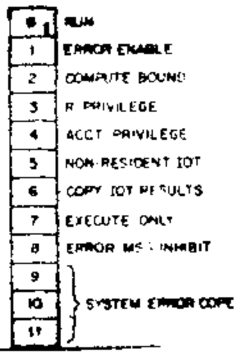


Figure D-10a STRO

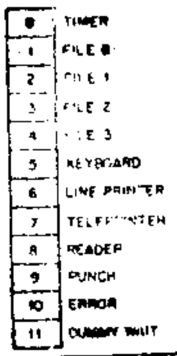


Figure D-10b STR1

The DECtape, card reader, and RK8E are exceptions. When a job is waiting for one of these device, the wait mask is set to zero. Bits 0-8 of wait 2 are set to the address of the DDB for the device, and bits 9-11 are set to 1 or 2 for DECtape, 3 for RK05 or 4 for the card reader.

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, and no IOT parameter block is set up.

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 for file reads and writes. If part of the file being accessed 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 pointer indicating where to go in the Monitor, when the transfer is complete, is also stored. This block is also linked into the disk queue (DSUTBL). See Figures D-11 and D-12.

CLKTBL is used to execute the STM instruction. It has a 1-word entry for each job. If that job is not waiting out an STM, its entry is zero. If it is waiting, the entry contains the number of seconds left to wait (in 2's complement). When the counter goes to zero, the timer flag for that job is set. See Figure D-13.

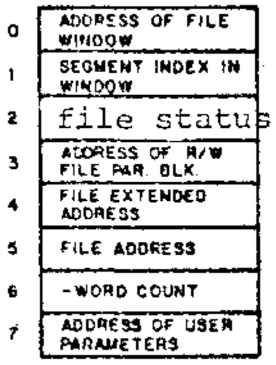


Figure D-11a.
File Control
Block

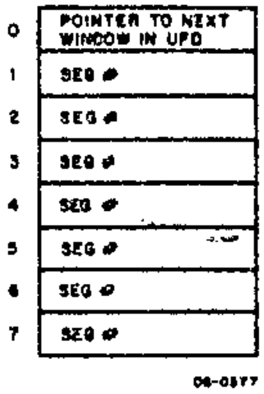


Figure D-11b.
File Window, or
File Retrieval
Information Block

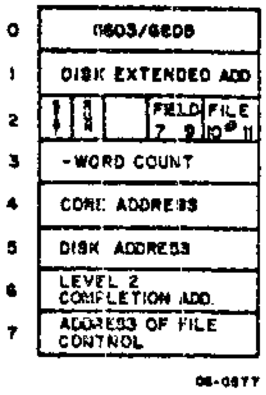


Figure D-12.
Read/Write
File Paramster
Block

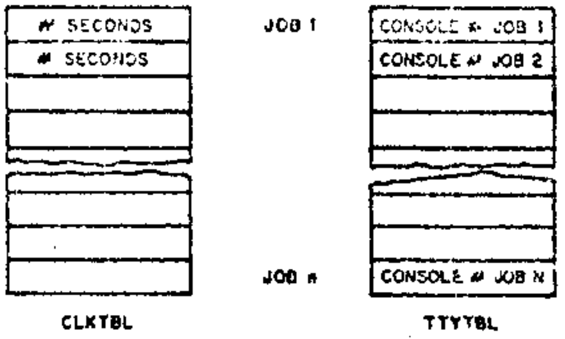


Figure D-13. CLKTBL and TTYTBL

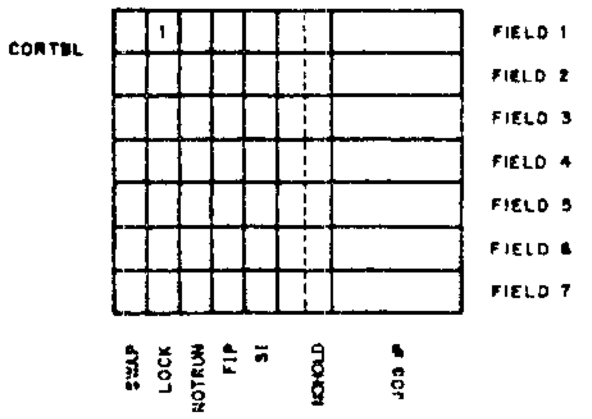
The TTYTBL table has a 1-word entry for each possible system job. Each entry contains the number of the terminal associated with that job. See figure D-13.

D.3 MONITOR SCHEDULING DATA BASE

DEVTBL, JOBTBL, and related status blocks maintain some EduSystem 50 status information relating to individual jobs. The monitor also maintains some of its own tables. These are used primarily to schedule.

CORTBL contains the status of the user fields. It is a 7-word table in monitor core, each word corresponding to a core field. Within each entry, bits 7 through 11 contain the job number of 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 is 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 swapp
Bit 2 is set if the job in that field has not been run. It can
swapped out until it has been run. FIP and SI are called phant
the sense that they run in place of a user job. Therefore, whe
is running, the calling job number is stored in CORTBL. Bit 3
set to remember that it is actually a phantom. Phantoms can ru
in field 2. CORTBL has an entry for every core field but fielc
whether it is available or not. At startup time, the Monitor's
and nonexistent fields have their lock bits set to prevent use.
Figure D-14.



04-0578

Figure D-14. CORTBL

PRGTBL maintains information on what program each user is
It has a 3-word entry for each possible job. When a user type
or RUN command, the filename (one to six characters packed in
format) is stored in PRGTBL. This information is used solely
EduSystem 50 SYSSTAT.

DSUTBL is the disk request queue. It contains a 4-word e
each core field in the system. A 7777 word terminates DSUTBL.
each 4-word entry there is a register for each of the four pos
files open to the user currently in that core field. If the e
zero, there is no file transfer pending for the internal file
user in that core field. If the entry is nonzero, it is a poi
a parameter block (the RFILE/WFILE parameter block) which desc
the transfer to take place. A pointer, DSKPTR, cycles through
to do transfers. See Figure D-15.

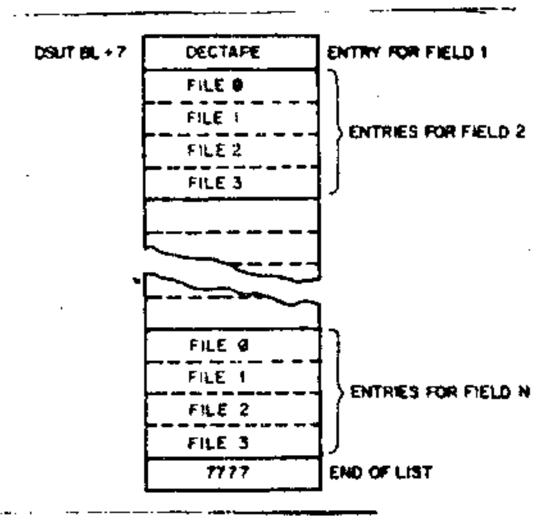


Figure D-15. DSUTI

DISK FILE DATA BASE

For each EduSystem 50 account number ther is a separate disk
e library that contains named files. The User File Directory,
ch ontains the filename (and some associated information) and
e location information for each file, controls this library.
name is in an 8-word name block; the retrieval information is
one or more 8-word file retrieval information blocks. The UFD
elf is stored in disk segments, up to a maximum of seven.

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

The MFD is identical in form to a UFD. The only difference
in the contents of the name block. Where the UFD has six file-
e characters packed into three words, the MFD has the account
ber in the first word, then two words of password. Altogether,
se three words are the name of the associated UFD. See Figure
6.

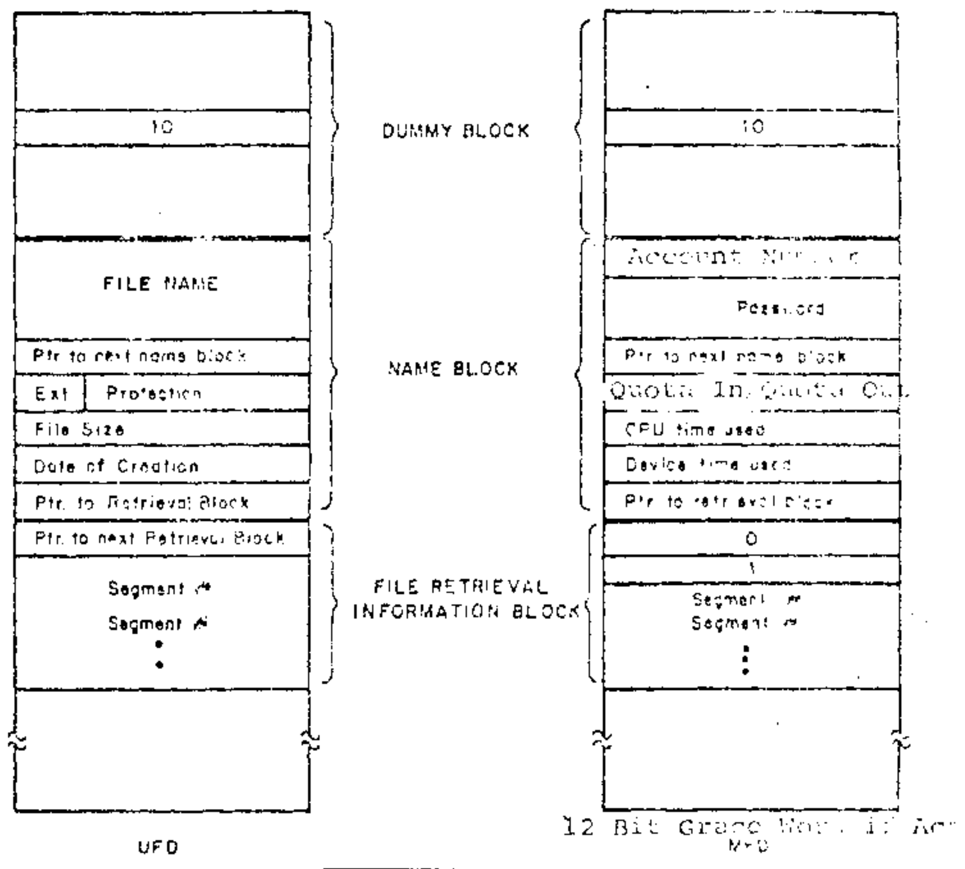


Figure D-16. File Directories

D.5 FILE PHANTOM DATA BASE

The primary data base of the File Phantom is the directories, MFD and UFDs. Although they may be accessed as files by a user log in with the system password, these directories are normally used on 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 f 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 IN sets all bits which correspond to nonexistent disk to 1s. The SAT located at the top of FIP's 4K. It is therefore swapped into core FIP. If the SAT is updated, it is written back to the disk. Below SAT is a register, SATCNT, which records the number of free disk se ments. See Figure D-17.

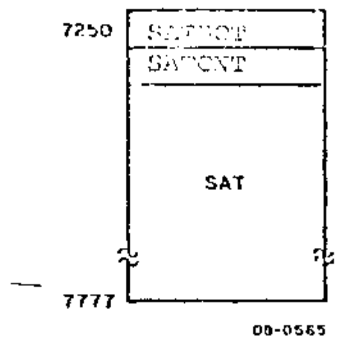


Figure D-17. SAT

FIP also maintains some convenient 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 1-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, the account number is stored. (Do not confuse FIP's JOBTAB with the Monitor's JOBTBL.) See Figure D-18.

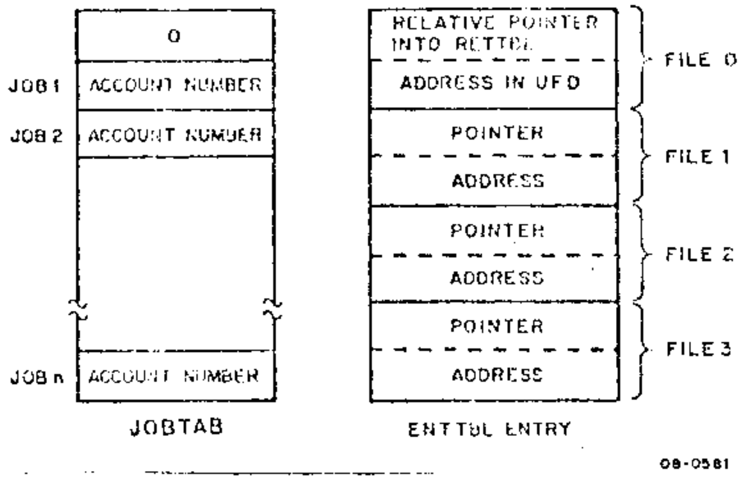


Figure D-18. FIP Tables

ENTTBL contains an 8-word block for each possible system job. Within these eight words are four 2-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. See Figure D-18.

UFDTBL and RETTBL work together to maintain retrieval information for all UFDs in use within the system. A UFD is in use if one or more users are logged in with that account or if the 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 4-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 2's 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. 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. For instance, the library UFD is invariably open to several users. See Figure D-19.

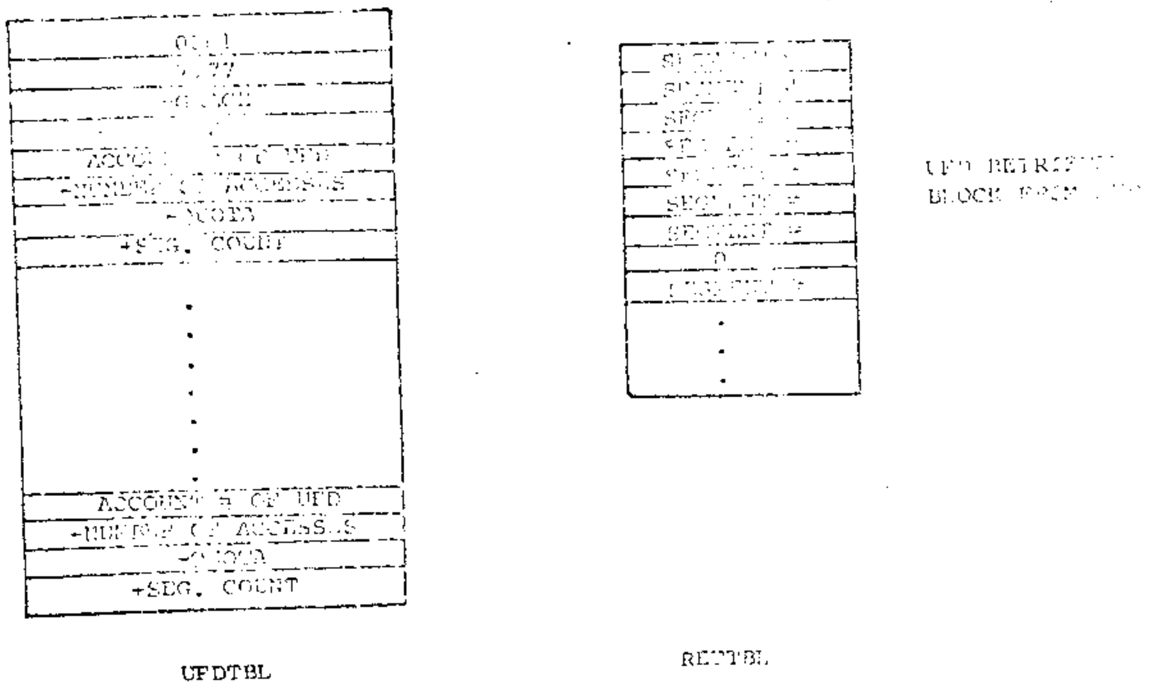


Figure D-19. UFD Retrieval Data

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 initialize time.

MONITOR TABLES

NAME	WHERE	TABLE SIZE	ENTRY SIZE
CORTBL	TS8	6	1
L2QTB	TS8	16	1
SKPTBL	TS8II	2xT+4	1
UUODTB	TS8	56	1
UUOTBL	TS8	56	1
BRKTBL	TS8	33	3
DEVTBL	TS8II	2xT+18	1
JOBTBL	TS8II	J+1	1
CLKTBL	TS8II	J+1	1
TTYTBL	TS8II	J+1	1
PRGTBL	TS8LL	3xJ	3
DSUTBL	TS8II	4xUF+1	4
JTABLE	FIP	J+1	1
ENTABL	FIP	8xJ	2
UTABLE	FIP	8xJ	4
RTABLE	FIP	16xJ	8
SATTBL	FIP	344	1/12
T=NULINE+1	J=JOBMAX	UF=# USER FIELDS	

APPENDIX E

ASSEMBLING AND LOADING EDUSYSTEM 50 FROM SCURCES

E.1 ASSEMBLING EDUSYSTEM 50 MONITOR

Use the following command lines under OS/8 version 3 to assemble Monitor, where CONFIG.PA has been modified to reflect the desired configuration:

```
.PAL SI<CONFIG,PARA,SI/K
.PAL FIP<CONFIG,PARA,FIP
.PAL INIT<CONFIG,PARA,INIT
.PAL TS8<CONFIG,PARA,TS8,TS8II/K
```

When using CREF, the M option will be necessary. The binaries may be punched on paper tape, or on 20K or larger systems, a shortcut may be taken. Type, under OS/8,

```
.LOAD INIT,SI,FIP,TS8/G=24200
```

Use the C option in INIT to write fields Ø through 4 to tracks Ø through 4 on the disk, initialize the system, load PUTR using the 0 option, start the system, log in under account 2, type "START Ø", and the system is up.

E.2 ASSEMBLING AND LOADING BASIC

To assemble BASIC under OS/8 version 3, type:

```
.PAL BASED
.PAL BASCOM
.PAL BASLDR
.PAL BASEXC
.PAL BASICN
```

create a SAVE format file, the binaries must be transferred to
System 50 library. Then, perform the following under EduSystem 50:

```
.CRE BASIC;OPE 3 BASIC;EXT 3 37
.R LOADER
INPUT-BASED
OPTION-
↑BS
.SAVE BASIC;R LOADER
INPUT-BASCOM
OPTION-
↑BS
.SAVE BASIC 6144 400;R LOADER
INPUT-BASLDR
OPTION-
↑BS
.SAVE BASIC 13150 400;R LOADER
INPUT-BASEXC
OPTION-
↑BS
.SAVE BASIC 13714 400;R LOADER
INPUT-BASICN
OPTION-
↑BS
.SAVE BASIC 14157 400
```

SSEMBLING AND LOADING THE FORTRAN SYSTEM

o assemble the FORTRAN system under OS/8 version 3, type:

```
.PAL FORT,DECODE
.PAL FOSL,DECODE
.PAL FDCOMP
.PAL FOSSIL
```

o create the new SAVE files under EduSystem 50, first type:

```
.CRE FORT;OPE 3 FORT;EXT 3 5
.CRE FOSL;OPE 3 FOSL;EXT 3 5
.CRE FDCOMP;OPE 3 FDCOMP;EXT 3 14
.CRE FOSSIL;OPE 3 FOSSIL;EXT 3 9
```