TENEX

MONITOR MANUAL

Technical Summary of
the TENEX Monitor

FEBRUARY 1972

Bolt Beranek and Newman Inc.
Cambridge, Mass.

TENEX

MONITOR MANUAL

Technical Summary of
the TENEX Monitor

Raymond S. Tomlinson
Daniel L. Murphy

Bolt Beranek and Newman Inc
50 Moulton Street
Cambridge, Massachusetts

TABLE OF CONTENTS

	SECTION	Page
SOURCE FILES	I	1
MONITOR	I	2
FILESYSTEM	I	6
UTILITY	I	9
BUILDING A SYSTEM	II	1
PARAMETER FILES	II	1
PARAMS.	II	1
FPARAM.	II	3
ASSEMBLIES	II	5
LOADING	II	5
STARTING SYSTEM	II	8
REFRESHING THE DISK	II	9
STOPPING THE SYSTEM	II	11
SYSTEM DEBUGGING.	III.	1
EDDT	III.	1
BUGHLT,BUGCHK	III.	1
DBUGSW	III.	3
CONSOLE SWITCH CONTROL	III.	4
MDDT	III.	5
PATCHING THE MONITOR.	III.	5
MINI-EXEC	III.	6
REAL CORE AND MONITOR VIRTUAL ADDRESS SPACES	IV	1
REAL CORE BOUNDARIES.	IV	1
MONITOR SPACE BOUNDARIES	IV	1
MONITOR SPACE STORAGE ASSIGNMENTS	IV	2
MONITOR TABLES AND DATA STRUCTURES	V	1
TABLES	V	1
STORAGE ADDRESSES.	V	1
CORE STATUS TABLE.	V	2
SPECIAL PAGES TABLE	V	5
DRUM STATUS TABLE.	V	6
JOB TABLES	V	6
FORK TABLES.	V	8
BALANCE SET TABLES	V	11

TABLE OF CONTENTS (cont'd)

SECTION Page

ADDING A NEW DEVICE. . . . . VI . . 1

FILESYSTEM TABLES AND FUNCTION LOGIC . . . . VII. . 1

DIRECTORY FORMAT . . . . VII. . 2
THE FIXED AREA . . . . VII. . 3
THE FREE AREA. . . . . VII. . 5
THE SYMBOL TABLE AREA . . . . VII. . 5

PRIVILEGED SUBSYSTEMS . . . . VIII . 1

ACCT1Ø . . . . VIII . 1
CHECKPOINT. VIII .43
DELD. . . . . VIII .45
NOTIFY . . . . VIII .46
SETMRP . . . . VIII .47
ULIST . . . . VIII .48
DISCUSE. . . . . VIII .49

SECTION I - TENEX SOURCE FILES

There are three groups of source files for the TENEX monitor; the groupings are principally for programming convenience.

GROUP	SOURCE FILES		LOADER FILES
MONITOR	PARAMS.		parameters for other assemblies
	PROLOG.		
	LDINIT		LDINIT
	SCHED.		MON
	PAGEM.		
	PISRV.		
	SWPMON		SWPMON
	TTYSRV		
	IMPDV		
	DSK		
	DSKPAK or DSKBRY		
	DRMDMY or DRMBRY		
	POSTLD(1)		POSTLD
FILESYSTEM	FPARAM		FILE
	FILE		
	DIRECT		
	FREE		
	GTJFN		
	IO		
	JSYS		
	DECTAP		
	DISC		
	DISPLA		
	LINEPR		
	MAGTAP		
	NETWRK		
	PLOTTE		
	PTP		
	PTR		
	ZLAST		
UTILITY	MFLIN	-	MFLIN
	MFLOUT	-	MFLOUT
	MR	-	MR
	DATIME	-	DATIME
	EDDT(2)	-	EDDT
	TENDMP(3)	-	TENDMP

(1) - Used after loading only, overwritten by variable storage at routine.

(2) - Not part of running monitor, for usual debug functions only.

(3) - Not part of running monitor; for bootstrapping swappable monitor only.

The above files constitute the entire TENEX monitor. The TENEX Executive language, while an integral part of the TENEX system, and the TOPS-10 compatibility package which provides monitor-type functions, are independent of the monitor and do not exist in the monitor address space. These are described in separate documentation.

Monitor Source Files

The sources in the monitor group are all written in the MACRO assembly language. They all have the extension .MAC, except for the installation-dependent parameter file. Only the SWPCOM assembly contains swappable code.

PARAMS

This file has an extension which serves to identify the installation which uses it. It contains parameters which control the overall size and capacity of the system, and condition the presence of various I/O devices and define their PI channels. Certain storage boundaries which must be defined at load time are also specified. This file is used in the LDINIT, MON, and SWPMON assemblies.

PROLOG

A file of parameters, storage assignments and macro definitions which are not, in general, installation dependent. This file defines the five major regions of the monitor address space, and numerous reserved pages in these spaces. All of the storage assignment macros (LS, GS, etc.) are defined (generally using the ASSIGN pseudo-op), as are macros affecting PI, bug strings, pseudo-interrupts, and scheduling. All PSB and JSB storage defined by the monitor group of assemblies is specified in a single group. All JSYS's in the system are defined here using a macro that also builds the JSYS transfer vector.

LDINIT

A short file that serves to define the six storage PCs for the loading. Operation of the ASSIGN pseudo-op in the loader requires that an assignment PC be defined in a file previous to the file in which it is first used.

POSTLD

The code in this file is run immediately following the loading to perform a variety of functions outside the capabilities of the loader. It:

1. Creates (via a call to the disk driver module) the short monitor bootstrap file RLRMON.SAV.

2. Moves the symbol table to its runtime location, removes redundant symbols, writes the symbol table file MONSYMS.TBL;version, and removes additional symbols from the table if necessary to bring the top of the syymbol table below a parameter maximum value (MAXSYM).

3. Sets several cells which depend on the loaded size of the system e.g. SWPCOR, MONCOR, SWCEND.

4. Sorts the bug location table numerically, and creates the BUGSTRINGS.TXT and BUGTABLE.IMG files.

5. Sets the entry vector and cleans up the core image, including itself.

SCHED

The TENEX scheduler and related routines. Defines storage for all system tables and variables for forks, jobs, and the balance set. Contains: the primitives for entering the scheduler (SCHEDP, DISMSE, channel 7 PI); routines for instruction trap, several common dismiss conditions; the scheduler proper including balance set and process priority routines; the system clock; fork and job creation code; and the break and de-break functions of the pseudo interrupt system.

PAGEM

The TENEX page management module. Includes initialization for the CST and SPT; drum-to-disk storage flow; OFN control; the primitives for setting, changing and reading all page tables and index blocks; core page locking primitives; pager setup; the core manager routines (system and per-process); the swapper (in and out) routines; and the pager trap logic.

PISRV

PI service and miscellaneous routines. Contains the system start vector; system initialization code which calls all device or module initialization code; system crash and restart procedures; all APR PI handelers; dispatch procedures for device PI channels; and the UUO and JSYS enter and return routines.

SWPMON

Swappable monitor routines. Includes: job and system initialization; swappable monitor bootstrap procedures; the MINI-EXEC, a limited command interpreter for certain system loading and maintenance functions; the periodic system check and error reporting logic; and a large set of JSYS's, including fork creation and control.

TTYSRV

Teletype (terminal) service. Line-indexed storage; terminal JSYS primitives; terminal input and output drivers.

IMPDV

Driver for the IMP (Interface Message Processor, ARPA Network). Interfaces mainly to NETWRK, and also to TTYSRV. Handles formatting of input and output streams, constructs and analyzes messages; compiles and dispatches control messages.

DSK

Device independent disk routines. Includes: disk bootstrap related routines; and the disk storage assignment logic.

DSKPAK,DSKBRY

Disk driver modules (device dependent). Interfaces for swapping I/O and utility I/O to disk; system bootstrap from disk routine; parameters for storage allocation; tables and routines for converting internal (linear) addresses to/from hardware addresses.

DRMBRY,DRMDMY

Drum driver modules (device dependent, DRMDMY is a dummy driver that uses disk). Interface for swapping I/O to drum; parameters and routine for drum storage allocation.

FILESYSTEM SOURCE FILES

The filesystem consists of 8 files of device dependent code. All files are assembled together by FAIL to produce a "REL" file which is loaded with other system files.

The assembly procedure is usually controlled by the file ASSFIL.installation via RUNFIL. This file should be tailored for each installation to include any device dependent files for that installation and to exclude files not needed. All optional files include a conditional assembly around the body so if a file is not wanted, changing the appropriate parameter (see below) will exclude it. However, assembly will be speedier if an ASSFIL is tailored for the situation.

The first file asembled must be FPARAM.installation which contains parameter definitions.

The second file in the assembly must be FILE.FAI which contains various macro definitions, storage allocation, AC definitions, and various other things which belong first.

The last file in the assembly must be ZLAST.FAI which finishes up the bug-string information, assembles literals and contains the end statement.

The other files may appear in any order. Normally, the preferred order is device independent files alphabetically followed by device dependent files alphabetically.

File Contents

Each of the 17-odd files which constituTe the TENEX filesystem are described below.

FPARAM.installation

This file contains various parameters which control the assembly of the TENEX filesystem. Note that the extension should reflect the installation to which the parameters pertain.

A parameter is defined for each optional device which is to be included in the assembly. For multiple unit devices this symbol specifies the number of units on the device which are to be supported by the software.

DIRECT.FAI

The file DIRECT.FAI consists of 3 blocks named DEVICE, LOOKUP, and DIRECT. These blocks constitute the routines for doing device name lookup, device independent file name lookup, and disc file directory management.

FREE.FAI

This file contains routines for managing storage associated with the file system.

GTJFN.FAI

This file consists of one block with the same name as the file and contains code for GTJFN and GNJFN JSYS's.

IO.FAI

This file contains two blocks; IO and CONVER. These two blocks constitute most of the sequential, random, and dump input/output routines.

JSYS.FAI

This file contains code for most of the file system and directory JSYS's.

DECTAP.FAI

This file contains all device dependent code for driving a standard DEC TD-10 DECtape controller. The file contains two blocks.

DISPLA.FAI

This file contains routines to implement the JSYS's for running the E&S LDS-1 display processor and the interrupt routines and scheduler for same. These routines are not strictly part of the file system but are included in the file system assembly because they were written by the same author and utilize some of the assembly environment provided by that assembly.

LINEPR.FAI

This file contains device dependent routines for driving BBN's line printer. This is an oddball device and the code is probably not pertinent to any other installation.

MAGTAP.FAI

This file contains device dependent routines for operating a DEC TM10A magtape controller.

NETWRK.FAI

This file contains device dependent routines for incorporating the ARPA network into the TENEX file system. Routines in this file are independent of the exact IMP interface and communicate with routines in the file IMPDV.MAC on the level of "send an STR to this host for these socket numbers" etc.

PLOTTE.FAI

This file contains device dependent routines for driving a Calcomp model 563 incremental plotter from the file system.

PTP.FAI

This file contains device dependent routines for driving the paper tape punch.

PTR.FAI

This file contains device dependent routines for driving the paper tape reader.

ZLAST.FAI

This file contains the epilogue for the file systeM assembly. It causes the end of swappable code to be printed, the length of resident code to be printed, and assembles the bugstring information for subsequent processing by POSTLD.

UTILITY SOURCE FILES

This group contains several independent functions which are part of the monitor support. They are rarely changed and are therefore assembled independently.

MFLIN,MFLOUT,MR

The floating point input and output conversion routines, and a double precision floating point arithmetic package used by them.

DATIME

The TENEX date and time to string conversion routines.

EDDT

Exec-mode DDT loaded as part of the resident monitor, used for debugging basic monitor functions.

TENDMP

A version of TENDMP with a pre-stored command string which bootstraps the swappable monitor from dectape to the monitor address space. Although loaded into lower core, it is assembled to run in page 377 so that it may be above everything it loads. It is run once, called by system initialization code in SWPMON.

SECTION II - BUILDING A SYSTEM

In general, building a system requires that a set of basic characteristics be determined (e.g. devices available, size of tables) and that parameters be set appropriately. Parameters for the major size items and for conditioning the existence of devices are grouped in two files, PARAMS and FPARAM. The assemblies and loading are controlled by command files, so the system builder need only know which command files must be run. Placing the system on dectape, starting the system and debugging it involve manual operations discussed here.

Parameter Files

PARAMS

(see also "TENEX Source Files - Monitor") File contains parameters to set the sizes of principal monitor tables, condition devices, and define PI channels. Items:

1. System name and version text. A short string defined with a macro specifying the installation and version of the system (e.g. BBN-TENEX 1.28, SYSTEM-A). The parameter SVNM should be defined as the version number of the system consistent with the text (e.g. SYNM==↑D128) as it is used as the file version number for several related files on which the system is dependent (MONSYMS.TBL, BUGTABLE.IMG, BUGSTRINGS.TXT, RLRMON.SAV).

2. NJOBS the size of job-indexed tables and therefore the maximum number of simultaneous jobs possible on the system. There are 4 such tables, and in addition the number of forks (NFKS) is defined to be 3*NJOBS, and there are 12 fork-indexed tables, so the number of words of storage directly controlled by this parameter is 40*NJOBS.

3. NLINES - the total number of terminals available in the system, including ARPA network lines if any. There are 19 tables indexed by line, so the storage directly controlled is NLINES*19.

4. NTTYS - the number of scanner lines in the system (.LE. NLINES). The difference NLINES-NTTYS gives the number of network lines.

5. NTTBF - the number of terminal buffers available. TTSIZ is the size of each buffer in words, so the storage used is TTSIZ*(NTTBF+1). This pool of buffers

is used for terminal input, output, and echo output; buffers are assigned as needed and released when empty. Several physical buffers may be used as one logical buffer for greater capacity; the number used for input and for output is independently recorded for each line, while for echo output, 1 buffer is used. The maximum number of buffers possibly in use simultaneously is therefore

$$\sum_{I=1}^{NLINES} \text{INPUT ASSMT+OUTPUT ASSMT+1}$$

In the system as distributed, input and output assignments are both 2, so the maximum is 5*NLINES. In practice, considerably fewer buffers are allocated, (utilizing the fact that lines are active only a small fraction of the time) typically between 2 and 3 times NLINES. The system presently has no provision to handle the buffer pool becoming empty.

6. SSPT - Operating size of the SPT. The hardware allows a maximum of 8K (20000 octal), but considerably less is generally allocated. Two tables are allocated by this parameter. The SPT must be used for JSBs (1 per job), PSBs and UPTs (1 each per fork), and is also used for mapped file pages if not full.

7. NDST - Size of the drum status table, i.e. 1 word per drum page. This must be consistent with the actual drum space available for swapping as defined in the drum driver source (DRMDMY,etc.).

8. TMZONE - the time zone in which the system is operated; it equals the number of hours local time lags GMT. It is used only in DATIME.

9. PI Channel assignment for all devices in the system are defined in PARAMS. The devices are generally known by 3-letter codes (DTA, LPT, etc.), and the channel assignment symbol for each device is formed by appending CHN to the device code (DTACHN, LPTCHN, etc.). If the channel assignment is not defined for a particular device, the links to that device for PI service, initialization, etc. (in PISRV) will not be assembled. If the driver code for the device is in the monitor group of source files, assembly of it is also conditioned by the channel assignment. (In the filesystem group, assembly conditioning parameters are in FPARAM.).

10. Resident storage. To use the space between CST0 and the SPT (5000-20000), several large blocks of storage are defined with explicit parameter assignments, including CST2, CST3, DST, SPTH, and optionally TTBUFS. Some set of these (including possibly RESLOC, the other resident storage) is assigned starting at CST0+MAXCOR (5000 for 256K or less systems), and the remainder is assigned starting at SPT+SSPT.

11. MAXSYM - the highest permissible end of the symbol table. Used by small installations to conserve core. The POSTLD routine will remove symbols from the table on a per-file or per-block basis according to a list in POSTLD until the top of the symbol table is below MAXSYM or the list is exhausted.

12. Monitor address space boundaries. NRESBG and NRPLBG have considerable free space remaining, so it is unlikely that these need ever be changed. SWPMPC is chosen based on the size of the swappable portion of the filesystem assembly and is the initial (absolute) location for the swappable monitor and utility code. After the filesystem assembly (FILE), the assembler prints the location of the end of the swappable code. SWAPPC is also defined to have this value, and a check for overlap is made after loading.

13. Bug string locations. These are used only during loading to hold the BUGHLT and BUGCHK locations and comment strings. POSTLD processes the information and moves it onto files, and the area is overwritten with resident storage at runtime.

FPARAM

A parameter is defined for each optional device which is to be included in the assembly. For multiple unit devices this symbol specifies the number of units on the device which are to be supported by the software.

The parameters include the following.

SWPMP0

Swappable monitor page 0. Defines where swappable monitor code begins.

MAXSWP

Location where other swappable code begins. May be used for an overlap check in ZLAST.

DTAN

This should be defined to be the number of DECtape units wanted in the system.

MTAN

This should be defined to be the number of magtape units wanted in the system.

LPTN

Number of lineprinters. (Note that the current system does not support multiple line printers.) Not defined if no line printer wanted.

PLTN

This symbol should be defined if a plotter exists.

PTPN

This symbol should be defined if code for the paper tape punch is wanted.

PTRN

This symbol should be defined if paper tape reader code is wanted.

NLINES

This parameter must be set to the total number of scanner lines plus console TTY plus network virtual terminals. This must agree with the definition of NLINES found in PARAMS.installation.

NETN

This symbol should be defined if code for the network is wanted.

NSKT

This should be defined to be the number of network connections (sockets) which can be accommodated at one time. Because the index into socket tables is used to compute the link number, NSKT must be less than 70 (decimal).

NNETBF

Total number of network buffers. This can be less than NSKT because NVT's do not require buffers.

LHOSTN

Local host number. This must be defined correctly for each different host

NDP

Number of display processes on E&S LDS-1.

NDC

Number of display consoles on E&S LDS-1.

If other devices are added, switches of the form "devN" should be added to govern their inclusion in the assembly.

Assemblies

There are a total of 11 loader files which must be available prior to loading. There are 3 command files to drive assemblies, one for each of the three source file groups. To produce the assemblies for each group, one need only start RUNFIL and give the appropriate file name. The command input and all assembler printouts appear on the terminal as usual, and aside from checking for assembly errors, no further action need be taken by the operator.

The assembly command files are:

Monitor - MACMON	(MACRO assembler)
Filesystem - ASSFIL	(FAIL assembler)
Utility - MACUTL	(MACRO assembler)

Groups or files within groups may be individually reassembled so long as common parameter files are not changed; i.e. if a parameter file is changed, all assemblies of which it is a part must be reassembled.

Loading

All monitor code is loaded into a single core image just as it will appear in the monitor virtual address space at runtime. The loading is controlled by the command file LOD10X which, like the assembly command files, is processed by RUNFIL.

The load address for the resident monitor exists as an octal number (preceding LDINIT) in this file. It must be adjusted from time to time as resident storage requirements change so as to cause the resident code to be loaded just after the resident storage ends (see Monitor Virtual Address Space).

Several additional operations are controlled by LOD10X after the loading itself is completed:

1. A storage map and list of undefined globals (if any) is produced; in some cases undefined globals may be patched, in general there should be none.

2. An SSAVE file is made containing the entire loaded core image.

3. In DDT, various parameters are compared to check for storage overlap. A negative number (printed by DDT as unsigned magnitude) indicates overlap, and correction of relevant parameters and reloading and possibly reassembly is required.

4. The POSTLD procedures are started (see TENEX source files). This may be manually aborted if errors in the preceding operations have made the loading unuseable. The files MONSYMS.TBL, BUGSTRINGS.TXT, BUGTABLE.IMG, and RLRMON.SAV are produced.

5. Another SSAVE file is made containing the final core image and compacted symbol table.

The core image of the monitor is now in the file (e.g. AMON.SAV) produced by LOD10X. The file BUGSTRINGS.TXT should be listed; it provides an ordered list of all BUGHLT and BUGCHK locations and a message describing the cause. This is generally posted near the console for operator and system programmer convenience. The files MONSYMS.TBL and BUGTABLE.IMG should be in the <SYSTEM> directory when the system is run; they may be copied or renamed, but the file version number must be kept the same. These files will match only the core image from which they were created, so the file version number is set to be the same as the system version number, and the system will use only those files with the correct version number.

The basic system startup procedure involves a bootstrap from dectape. This dectape contains two SAV files, one for the resident and one for the swappable portion of the monitor. The procedure for constructing a bootstrap dectape containing a freshly loaded system is:

```
@ASSIGN DTA1:
@CLEAR DTA1:
  [CONFIRM]
```

Due to limitations in TENDMP, the two SAV files must be written on monitonically increasing block numbers on dectape. Clearing the dectape before writing the monitor files allows core images up to about 20K to be written before reaching the end. If this is still insufficient, then the dectape block spacing in the monitor (DTASPC) must be patched so as to reduce the spacing from the normal 3 down to 2 or possibly 1. The present monitor as run on the BBN systems may be written with no change to the block spacing.

If not already present a 48K or greater TENDMP should be put on the dectape. This may be copied from another dectape by using DTACPY.

Next, the two monitor SAV files are written from the core image produced by the loader.

```
@GET AMON.SAV
@121/ 123777
```

Location 121 gives the highest address in the resident area, which is in fact the end of the symbol table. This address is used in the save command:

```
@ SAVE 20 (TO) 123777 (ON) DTA1:TENEX.128
```

Any file name may be used. The one shown is common, and identifies the version of the system. Next, the swappable monitor must be saved. The code starts in SWPMP0 (in FPARAM) which is presently 240000. The last page used for swappable code is reported by POSTLD at the start of its operation. The message "272 IS LAST PAGE OF SWAPPABLE CODE" provides the information needed for the save command for the swappable monitor.

```
@SAVE 240000 (TO) 272777 (ON) DTA1:TENEX.SWP
```

The file name shown is the one assembled into the special TENDMP and so must be used exactly as shown.

These are the only files necessary for starting the system without refreshing the disk. It is useful to have on the same tape a save file of DUMPER for those occasions when the disk is to be refreshed and loaded from magtape; a save file of DLUSER and the user information file it produces to load user information ~~when the disk is refreshed; RLRMON.SAV (copied from the~~ file produced by POSTLD) for quick bootstrapping from disk; and PMDDT (described below) for monitor debugging with an empty disk.

There are a few remaining initializations necessary in the core image which cannot be done in user mode. Therefore, the first time the new monitor is to be used, the following should be done:

1. Press READIN, wait for EOL from TENDMP
2. Load and start resident monitor image file, i.e. MS1 SO

The starting address for this file is set by POSTLD to be 100, and that location always contains a jump to DDT, so when loading is completed the console teletype will be in control of exec-mode DDT. The following breakpoint setup operations are necessary:

```
    $I+1/  1777    1020
    BUGH0+2(BUGHLT)$8B
    BUGC0+1(BUGCHK)$7B
    137400$G
($ = altmode)
```

The first operation causes DDT to leave PI channels 0-2 and 4-7 enabled on a breakpoint; the second and third set the BUGHLT and BUGCHK breakpoints; and the last returns control to TENDMP (a different address must of course be typed if other than 48K TENDMP is in use). TENDMP is then used to rewrite the resident monitor file, e.g.

```
D$MS1 SO
```

Starting the System

When loading from a dectape setup as described above, the procedure is:

1. Mount tape on DTA0, press "READIN"

2. To TENDMP, type MS1 SO

3. When tape motion stops, the console teletype will type EOL, and DDT will be in control.

4. Set DBUGSW to the desired value (see below) if different than on tape.

The following presumes that the disk contains an intact file structure and will not be refreshed.

5. The system full start (without refreshing the disc) address is SYSGO1. To DDT, type SYSGO1$G

6. The system will promptly begin spinning DTA0 to load the swappable monitor from the file TENEX.SWP.

7. "TENEX RESTARTING, WAIT..." will be typed on all connected terminals.

8. The consistency of the file structure will be checked (using <SYSTEM>CHECKDSK.SAV), an operation which reads all file index blocks. It takes a length of time approximately equal to: average disc access*(number of files+8*number of directories).

9. When the above is completed, a job will be started on the console teletype, and the EXEC will request the date and time to be entered. Until the date and time have been set, any EXEC started on any line will request it. When this has been entered, the system is in regular operation.

DISK Bootstrap

The system may also be bootstrapped from disk, replacing steps 1-3 above. Whenever the system is started from dectape as above, an image of the resident and swappable monitor is written onto a reserved area of the disk. Subsequently, the short file RLRMON.SAV will serve to get the resident monitor, and the initialization procedure will load the swappable monitor from disk rather than dectape. The RLRMON.SAV file itself (produced by POSTLD as described above) may be loaded from dectape, paper tape, or any other available readin device on which it can be put. As with the dectape loading, control will be given to DDT after the loading, and the procedure should continue with step 4, above.

Refreshing the Disk

To refresh the disk, loading is performed as above through step 4. It then continues:

5. To DDT, type SYSLOD$G

Steps 6 and 7 will occur as described

8. The system will type several messages on the console teletype reporting files which could not be found, e.g. "NO CHECKDSK", "NO EXEC".

9. Because the regular EXEC file does not exist, the first job will be started in the mini-exec, a simple command interpreter built into the monitor and capable of performing various maintenance functions. The mini-exec's ready character is a period. All of its commands are uniquely determined by the first letter, and the mini-exec immediately types the remainder of the command.

10. Type: `.INIT BIT TABLE.`

This must be done at least once. It creates the system disk bit table file and accepts bad spots to be marked. The system then says: "READ BAD SPOTS FROM FILE." The operator should enter the name of a dectape file containing the bad spot list, or he may supply them from the console teletype by giving TTY: as the file name. Control-Z indicates the end of the bad spot list when entering from TTY:. If there are no badspots to be entered, TTY: should be specified and control-Z immediately typed. The Init bit table operation may be repeated if additional badspots are to be entered from a different file.

11. Next, the directories may be defined. For this purpose, a copy of DLUSER.SAV and a text file of directory names produced by it should be available. Assuming they are on dectape, the following dialogue is appropriate:

```
.GET FILE DTA0:DLUSER.SAV
.GET FILE DTA0:DLUSER.SAV
.START
DUMP OR LOAD? L
FILE: DTA0:USERS
DONE.
```

12a. If magtape or some other medium for which DUMPER has been modified contains a dump of the disk, then loading may now be initiated by:

```
.GET FILE DTA0:DUMPER.SAV
.START.
```

12b. If files are to be loaded singly or in groups, the following procedure is appropriate:

```
.GET FILE DTA0:EXEC.SAV
.START.
```

This will start the regular EXEC. The time and date must be set as above, then the EXEC itself should be put in its usual place in the system directory by:

```
@GET DTA0:EXEC.SAV
@SSAVE 0 (TO) 777 (ON) <SYSTEM>EXEC.SAV[NEW FILE]
```

Until the file EXEC.SAV exists in the system directory, any job which is started will go into the mini-exec, a situation which should not be permitted to happen to unprivileged users. Various other items are required for the system to operate normally.

Briefly, they are:

1. Directory <ACCOUNTS>. The FACT file resides in this directory, EFACT will fail if it does not exist, and various routines which call EFACT will BUGCHK on failure. Following the load users step above, <ACCOUNTS> should exist. No files need be loaded into it.

2. File <SYSTEM>ERROR.MNEMONICS, the error number-message equivalence file. ERSTR will fail if not present.

3. File <SYSTEM>EXEC.SAV, the TENEX EXECUTIVE program, as above.

4. Directory and file <SUBSYS>PA1050.SAV, the TOPS-10 compatibility. Without this file, all TOPS-10 UUU's (40-77) will be illegal instructions.

5. Directory <PMFDIR0>. PMFs for job numbers greater than 7 are placed in this directory.

6. Subsystems in directory <SUBSYS> and related files in <SYSTEM> (e.g. LIB40.REL) as needed by the installation.

Stopping the System

Two procedures exist for stopping the system, both necessary only to ensure that all current information in core or swapping storage has been backed to disk.

The system may be stopped from any terminal on which a privileged (operator or wheel) user is logged. From the EXEC, the procedure is:

```
@ENABLE
!QUIT                   ;enter mini-exec
INTERRUPTED AT...
 .HALT TENEX.           ;give halt command
```

The job 0 function will run for a few seconds; when the console lights become stable, the processor may be stopped.

The system may also be stopped by manipulating the console switches, without using any terminal.

The procedure is:

1. deposit a bit 2 (100000,,000000) in 20

2. wait a few seconds until lights stop blinking

3. deposit a bit 0 (400000,,000000) in 20.

Lights will immediately become stable and processor may be halted.

Both of the above procedures presume that all jobs have logged out. Jobs still logged in will vanish without a trace, and in particular without an entry in the FACT file.

SECTION III - SYSTEM DEBUGGING

Debugging a complex, multi-process software system is largely a matter of absorbing sufficient knowledge, experience and folklore about the particular system, with a considerable element of personal preference, or 'taste', also involved. This section documents the features built into the system to aid debugging, and such folklore as can be described in written English.

Exec-mode DDT (EDDT)

As described in the section on system building, exec-mode DDT is loaded with the monitor. Since its load address is variable, a jump to DDT is always placed in location 100, so 100 may be considered the regular start address of DDT. A pointer to a resident copy of the monitor symbol table is stored in location 36 and is used only by EDDT.

Breakpoints may be inserted in the resident monitor with EDDT, but not in the swappable monitor in general, because its pages may be swapped out and be unavailable to EDDT. As described under System Building, breakpoints 7 and 8 are set at the BUGCHK and BUGHLT locations respectively (in PISRV). With appropriate settings of DBUGSW, the various monitor internal consistency checks will reach these breakpoints, causing a printout of the form: $8B>>BUGHO+2 BUGHLT/ locn. If desired, BUGCHKs may be proceeded with $P and will resume at the instruction following the JSR BUGCHK. BUGHLTs may not be preceded in this way; if the cause can be corrected, system operation can be resumed only by a jump to a specific location (e.g. FOO$G). The original instruction at BUGHO+2 is a machine halt (JRST 4,), so that if a BUGHLT results from an interrupt from EDDT or when the breakpoint is not set, the machine will halt.

BUGHLT,BUGCHK

The monitor contains a considerable number of internal redundancy checks which generally serve to prevent unexpected hardware or software failures from cascading into severely destructive reactions. Also, by detecting failures early, they tend to expedite the correction of errors.

There are two failure routines, BUGCHK and BUGHLT, for lesser and greater severity of failure. Calls to them with JSR are included in code by use of a macro which records the location and a text string describing the failure. The general form is:

```
     BUG(type,<string>)
```

where type is HLT or CHK, and string describes the cause.

For example,

```
     BUG(HLT,<PAGER TRAP FROM SCHEDULER>)
```

The strings and a table of locations are constructed during loading and are dumped on two files. The BUGSTRINGS.TXT file will produce an ordered listing of the BUG locations and messages for operator or programmer use; the BUGTABLE.IMG file is used by the system itself to log BUG occurrences.

BUGCHK is used where the inconsistency detected is probably not fatal to the system or to the job being run, or which can probably be corrected automatically. Typical is the sequence in MRETN in file PISRV.

```
     AOSGE INTDF
     BUG(CHK,<AT MRETN - INTDF OVERLY DECREMENTED>)
```

This BUGCHK is included strictly as a debugging aid; detection of a failure takes no corrective action. This situation usually results from executing one or more excessive OKINT operations (not balanced by preceding NOINT). It is considered a problem because a NOINT executed when INTDF has been overly decremented will not inhibit interrupts and not protect code changing sensitive data.

BUGHLT is used where the failure detected is likely to preclude further proper operation of the system or where file storage might be jeopardized by attempted further operation. For example, the following appears in file SCHED:

```
     MOVE 1,TODCLK     ;current time
     CAML 1,CHKTIM     ;time at which job 0 overdue
     BUG(HLT,<JOB 0 NOT RUN FOR TOO LONG>)
```

This check accomplishes two things:

1. A function of job 0 is to periodically update the disk version of bit tables, file directories, and other open files; absence of this function would make the system vulnerable to considerable loss of information on a crash which loses core and swapping storage. Job 0 protects itself against various types of malfunction, this BUGHLT detects any failure resulting in a hangup.

2. Detects if the entire system has become hung due to failure of the swapping device or some such event, on the basis that if job 0 isn't running, nobody's running.

DBUGSW

A monitor cell, DBUGSW, controls the behavior of BUGHLT and BUGCHK when they are called. DBUGSW is set according to whether the system is attended by system programmers.

If C(DBUGSW)=0, the system is not attended by system programmers, so all automatic crash handling is invoked. BUGCHK will return +1 immediately, appearing effectively as a NOP. BUGHLT will

1. if called from a process, crash that process, reset flags, counts etc., and type a message on the controlling TTY;

2. if called from the scheduler or at PI levvel, invoke a total reload from disc and restart of the system.

In either case, if the system continues to run or is restarted properly, the location of the BUG (saved over a reload) and its message will be reported on the logging teletype.

If C(DBUGSW) = 0, the system is attended, and one of the EDDT breakpoints will be hit. This allows the programmer to look for the bug and/or possibly correct the difficulty and proceed. There are two defined non-0 settings of DBUGSW, 1 and 2, which have the following distinction. C(DBUGSW) = 1, operation is the same as with 0 except for breakpoint action. In particular, the swappable monitor is write protected and CHECKDSK is run at startup as described. C(DBUGSW) = 2 is used for actual system debugging. The swappable monitor is not write protected so it may conveniently be patched or breakpointed, and the CHECKDSK operation is not run to save time. BUGCHK and BUGHLT procedures are the same as for 1.

The following is a summary of DBUGSW settings:

	0	1	2
Meaning	Unattended	attended	Debugging
BUGCHK action:	NOP	$7B	$7B
BUGHLT action:	CRASH JOB or SYSTEM	$8B	$8B
SWPMON write protect?	YES	YES	NO
CHECKDSK on startup?	YES	YES	NO

Other Console Control Functions

In addition to EDDT, several other entry points are defined at absolute addresses. The machine may be started at these as appropriate.

100	Start EDDT
140	Soft restart
146	Reload from disk and total restart
147	Total restart

The soft restart (address 140, SYSRST) reinitializes all I/O devices, but leaves the system tables intact. If it is successful, all jobs and all (or all but 1) processes will continue in their previous state without interruption. This may be used if an I/O device has malfunctioned and not recovered properly, or in any case where an IOB reset is desired but core and swapping storage are intact. The total restart initializes core, swapping storage and all monitor tables as described in the section on Starting the System.

A very limited set of control functions for debugging purposes and operated by the console switches has been built into the scheduler. To invoke the function, the appropriate bit or bits are set in the data switches, and deposited into location 20. The monitor does not look at the data switches, only location 20. The word is scanned from left to right (JFFO); the first 1 bit found will select the function

Bit 0

Causes scheduler to dismiss current process if any and stall (execute a JRST .), with -1 in AC0. Useful to effect a clean manual transfer to EDDT. System may be resumed at SCHED0 if no IOB reset done.

Bit 1

Causes the job specified by data switch bits 18-35 to be run exclusively. Temporarily defeats job 0 not running BUGHLT.

Bit 2

Forces running of job 0 backup function; see Stopping System.

MDDT

A version of DDT which runs in monitor space is available. It can examine and change the running monitor, and can breakpoint code running as a process but not at PI or scheduler level. When patching or breakpointing the swappable monitor, the normal write protection must be defeated, either by setting DBUGSW=2 on startup, or by merging a bit 3 (write access) into the map word of relevant pages. If you insert breakpoints with MDDT, remember that monitor code is reentrant and shared so that the breakpoint could be hit by any other process in the system. In this event, the other process will most likely crash as it will be executing a JSR to a page full of 0s.

MDDT is entered from the mini-exec by typing a slash (/). Control-P is used to return to the mini-exec.

The image of MDDT resides on the file <SYSTEM>MDDT.SAV in a special format. When the first process tries to enter MDDT, the monitor maps that file into a buffer in the monitor address space, and then copies it into a reserved block in the per-process address space. Subsequent processes entering MDDT invoke the copying operation. When the first MDDT is started, the monitor also checks for a symbol table file having the name <SYSTEM>MONSYMS.TBL;sys-version. If such a file exists, its contents are copied into a buffer in the swappable monitor area, and this table is shared among all users of MDDT. If the file is not available, the system attempts to use the resident symbol table pointer in 36.

The special file MDDT.SAV is made by running the program PMDDT.SAV while connected to the system directory. It types out the name of the file it is about to write and awaits a confirming EOL. PMDDT is created from DDT and PMDDT.MAC which contains instructions on assembling and loading.

Patching the Monitor

To make patches permanent, they must be incorporated into the dectape files of the resident and swappable monitor. To patch the resident monitor, one should load with TENDMP a clean coopy (MS1.SO), make the patches, return to TENDMP and rewrite the file. Patching the swappable monitor is somewhat more involved. Patches should be made with MDDT (which presumes that the system is viable to some extent), after defeating the usual SWPMON write protection. When the patches have been installed, the following mini-exec sequence will write an updated dectape file:

```
.RESET.                       ;clear user map
.BLT SWP MON.        ;copy SWPMON to user space
.DUMP ON FILE DTA0:TENEX.SWP$ ;write dectape file
.RESET.
```

The BLT operation will also update the image on disk.

If a bug in the swappable monitor prevents the system from running well enough to support the above, patches may be made with EDDT via the following procedure.

1. Load resident mon from dectape as usual (MS1 S0).

2. Put EDDT breakpoint at EXEC0+5 (immediately following swappable monitor load)

3. Start system as usual (SYSGO1$G)

4. When breakpoint is hit, all SWPMON pages will be in core, but no SWPMON code will have been executed. Install patches with EDDT.

5. Remove breakpoint, proceed system.

6. When system is running, write new dectape file with mini-exec as shown above.

The monitor may also be patched in user mode under time sharing; GET the file(s) from dectape and install patches with user DDT in the same way as you would patch any user program. Then rewrite the dectape files, noting the block spacing caveats listed under Building a System. The patches will of course not exist in the running monitor until the system is reloaded from dectape and restarted.

The Mini-Exec

Previous mention has been made of the mini-exec, in connection with various maintenance procedures in the system. The following is a complete description of its functions and commands.

The code constituting the mini-exec is entirely contained within the monitor, and therefore does not require any disk files to exist for its operation. The mini-exec code is found within the first dozen or so pages of

SWPMON.MAC.

The mini-exec consists of a very simple command interpreter, commands are uniquely determined by their first character, and the mini-exec immediately types the remainder of the command. The mini-exec is entered as a super-default when any of the following occurs:

1. There is no <SYSTEM>EXEC.SAV file available when a job is started.

2. The PMF for a job cannot be opened when the job is started.

3. The top fork of a job is terminated (executes a HALTF).

This last is the method by which a privileged user may enter the mini-exec: the sequence

```
@ENABLE
!QUIT
```

causes the EXEC to execute a HALTF. The mini-exec notes this by the message "INTERRUPT AT xxxxxx," and enters its command wait.

When the mini-exec is started in this manner, it enables control-P as an interrupt character to cause a return of control to the mini-exec. The message "ABORT" indicates this. Control-P may be used to leave MDDT, the EXEC, or to abort a partial mini-exec command.

The mini-exec ready character is a period. The commands are:

1. EXEC (requires no confirmation) - Loads the EXEC into the user space and starts it.

2. GET FILE - Takes name of SAV file, merges it into user space.

3. START (confirm with period) - Start program in user space via entry vector.

4. DUMP ON FILE - Takes name of file onto which entire user space is SAVED.

5. RESET - Clear user space, closes all files, etc.

6. MOUNT DTA - Accepts single digit n, mounts DTAn.

7. BLT SWP MON (confirm with period) - Copies swappable monitor into user space.

8. INIT BIT TABLE (confirm with period) - Creates bit table file if necessary and loads badspot file.

9. HALT TENEX (confirm with period) - Forces backup of current information to disk, stops system cleanly.

10. WRITE MON SYM TAB (confirm with period) - Writes the monitor symbol table from its swappable buffer location onto file MONSYMS.TBL;version. Necessary if symbol values have been changed or added. File should be written or renamed into <SYSTEM>

11. / (slash no confirmation) - Starts MDDT, loading it first if necessary.

12. ! (exclamation point, no confirmation) Starts user DDT (at 770000), loading an image of MDDT first if necessary.

13. ↑ (up-arrow, no confirmation) - Does an MRETN, thus returning to the last user PC +1.

SECTION IV - REAL CORE AND MONITOR VIRTUAL ADDRESS SPACE

The monitor operates in its own virtual address space which, like the user's, appears to be a full 256K of memory. Unlike the user space, however, the monitor space contains several distinct areas. It is important to understand the distinction between the monitor virtual address space and real core, since only in one area are they the same.

Real Core Boundaries

Real core is divided into two main areas, one which is part of the monitor virtual address space and one which contains pages for swapping. The lower portion of real core (see map) is seen by the monitor and various devices for reference cells, etc. It must be contiguous. The cell SWPCOR in the monitor defines the first page used for swapping. Above that, all core that the monitor can find (by referencing and checking for NXM) is used for swapping. The BBN Pager is capable of referencing real core up to one million words (3777777 octal), but swapping devices currently in use cannot reference above 256K words, so relevant monitor tables (CST0-4) are built for a maximum of 256K on current systems.

Monitor Space Boundaries

The monitor address space is divided into four areas by the pager: resident, per-processor, swappable, and per-process. The monitor uses the lower 48K of the per-process region for job-common storage by placing indirect pointers in those slots of the monitor map for each process which point to a job area map in the JSB. This gives the five areas shown in Fig. 2.

The boundary between resident and per-processor areas is effectively not fixed, as any additional resident pages needed can simply be mapped in the per-processor map(s). Also, the pager provides optional mapping of the resident area, and while the contents of this are logically not swappable, the mapping can serve to write- (or other access) protect the resident code. This can be done in the current monitor by issuing a CONO to the pager to enable resident monitor mapping; the map is always set up. There is a slight cost in speed in doing this as 1) the mapping time for each monitor instruction is added, ~~and 2) associative registers will be used (e.g. in PI~~ routines) increasing the amount of reloading necessary for the running process.

The map for the monitor virtual address space is segmented according to its function. The system monitor map (MMAP, fixed location 2000) maps the resident monitor (if enabled), the per-processor region (MMAP+100 to MMAP+177 for processor 0, MMAP+600 to MMAP+677 for processor 1), and the swappable monitor (MMAP+200 to MMAP+577). The top 64K of monitor address space is mapped by the top 128 words of the PSB, hence changing the PSB changes the mapping of the per-job and per-process areas of the address space.

Monitor Space Storage Assignments

Fig. 3 is a quite detailed picture of the storage assignment of the entire monitor map. To aid the reader's understanding, absolute addresses are shown for each item. Some of these (indicated by *) are determined by hardware and so are quite unlikely to change. Some however are determined as the system is loaded and depend on the size of storage assignments, etc., so the numbers given here should be considered only as representative of a system of about the size of BBN's System-A (e.g. 40 jobs, 73 terminals, etc.). Also shown are the parameter symbols which control the loading or variable assignments, or which results from the loading.

Notes on Assignments

Hardware fixes the JSYS transfer vector at 1000, the monitor map at 3000, the core status table (only one of the four parts is used by the pager) at 4000, and the SPT at 20000. We attempt to use the space between these locations for other resident storage. Another part of the core status table exactly fits in 2000-2777 (for 256K core max).

The large area from 5000 (end of CST0) to 20000(SPT), is used for a variety of purposes, varying from installation to installation, including the drum status table and the terminal buffers. The remaining resident variable storage is assigned following the SPT, some large items (CST2, CST3, SPTH) with explicit parameter assignments, and the remainder at load time with the ASSIGN pseudo-op of the assemblers and loader.

The resident code follows the end of resident storage, but since the storage is assigned at load time, its total size cannot be known by the loader until loading is complete. Therefore, the load address of the resident monitor must be arbitrarily chosen by the programmer. When loading is completed, the end of resident storage (RESLOC) is checked. If it overlaps the resident code or leaves excessive space, a second loading must be done, adjusting the resident code load address accordingly.

The resident parts of all assemblies are relocatable and so are loaded compactly with no boundaries defined. Exec-mode DDT is relocatably loaded following the resident code.

The swappable region contains code and two variables areas. The first is assigned in arbitrary size blocks while the second is assigned in units of one page thus assuring that each block will begin and end on a page boundary. This is convenient for buffers which must be locked in core.

Our assemblers and loader are capable of maintaining only one relocatable PC, so those assemblies which contain both resident and swappable code assemble the swappable code into absolute locations. There are two such assembles, producing the files SWPMON and FILE. The set of utility library-type routines (MFLIN, MFLOUT, DATIME) are entirely swappable, so they are assembled relocatably and are loaded after the mixed assemblies, beginning at the first location after SWPMON.

The region beginning at PJMA is common to all processes of a job. It contains the job storage block (JSB), JFN storage, and a pool of pages which is dynamically assigned for file window pages, string storage etc.

The region beginning at PPMA is independent in each process. It contains pages reserved for various process functions including fork and map manipulation and pseudo-interrupt storage. An area is reserved for mapping file directories, and one for running monitor DDT. The top page contains the PSB for the process; below that is mapped the page table for the user address spaces and the page to hold AC blocks on monitor calls. UMOVE and XCT mapped references to effective addresses 0-17 are converted to virtual addresses 775.ACBAS.E. There is space for an adequate number of AC blocks in the PSB, so the PSB is mapped into page 775 as well as 777.

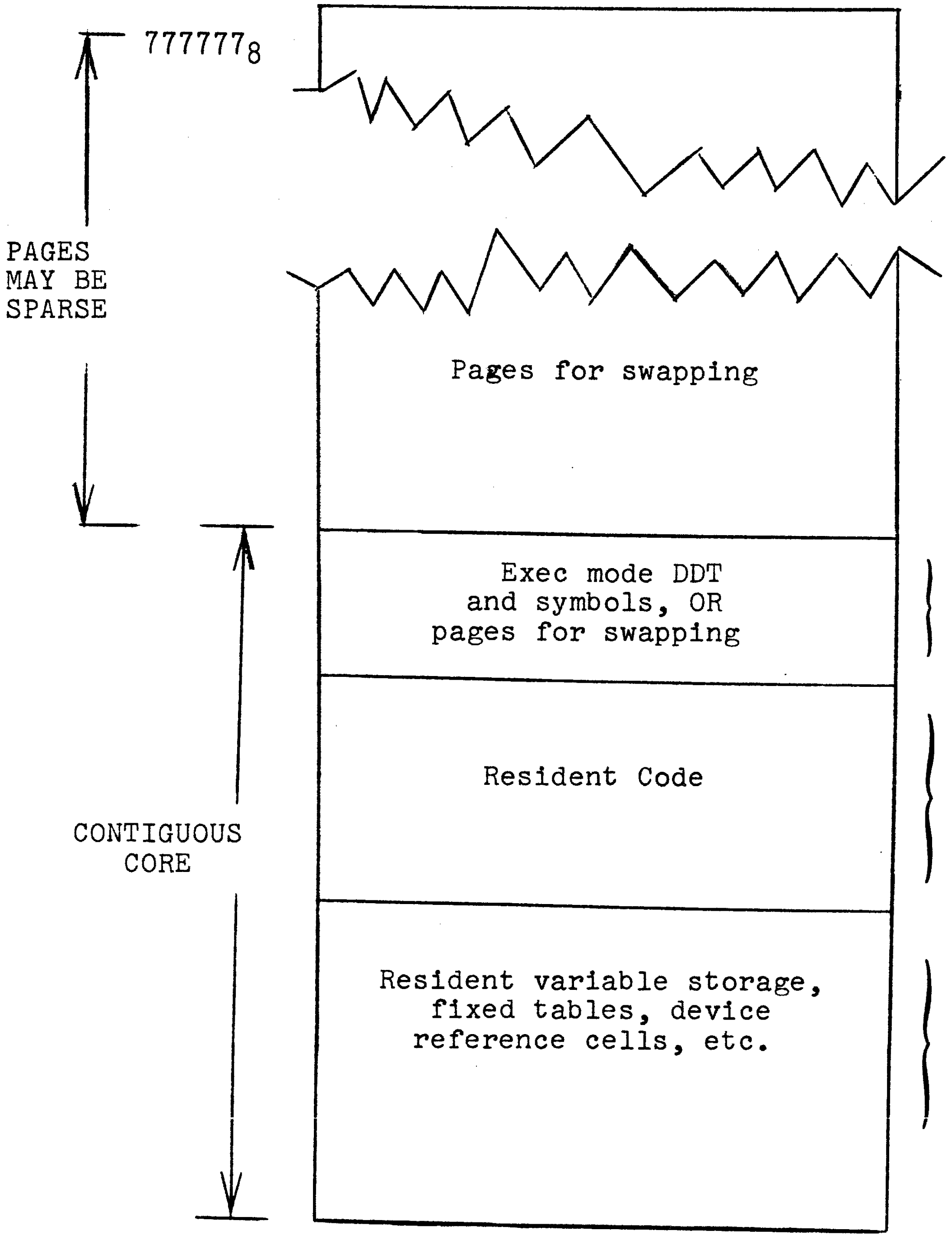

777777_8
PAGES
MAY BE
SPARSE
Pages for swapping
Exec mode DDT
and symbols, OR
pages for swapping
3-10k
Resident Code
8-12k
CONTIGUOUS
CORE
Resident variable storage,
fixed tables, device
reference cells, etc.
8-24k

Fig. 1

Monitor Virtual Address Space

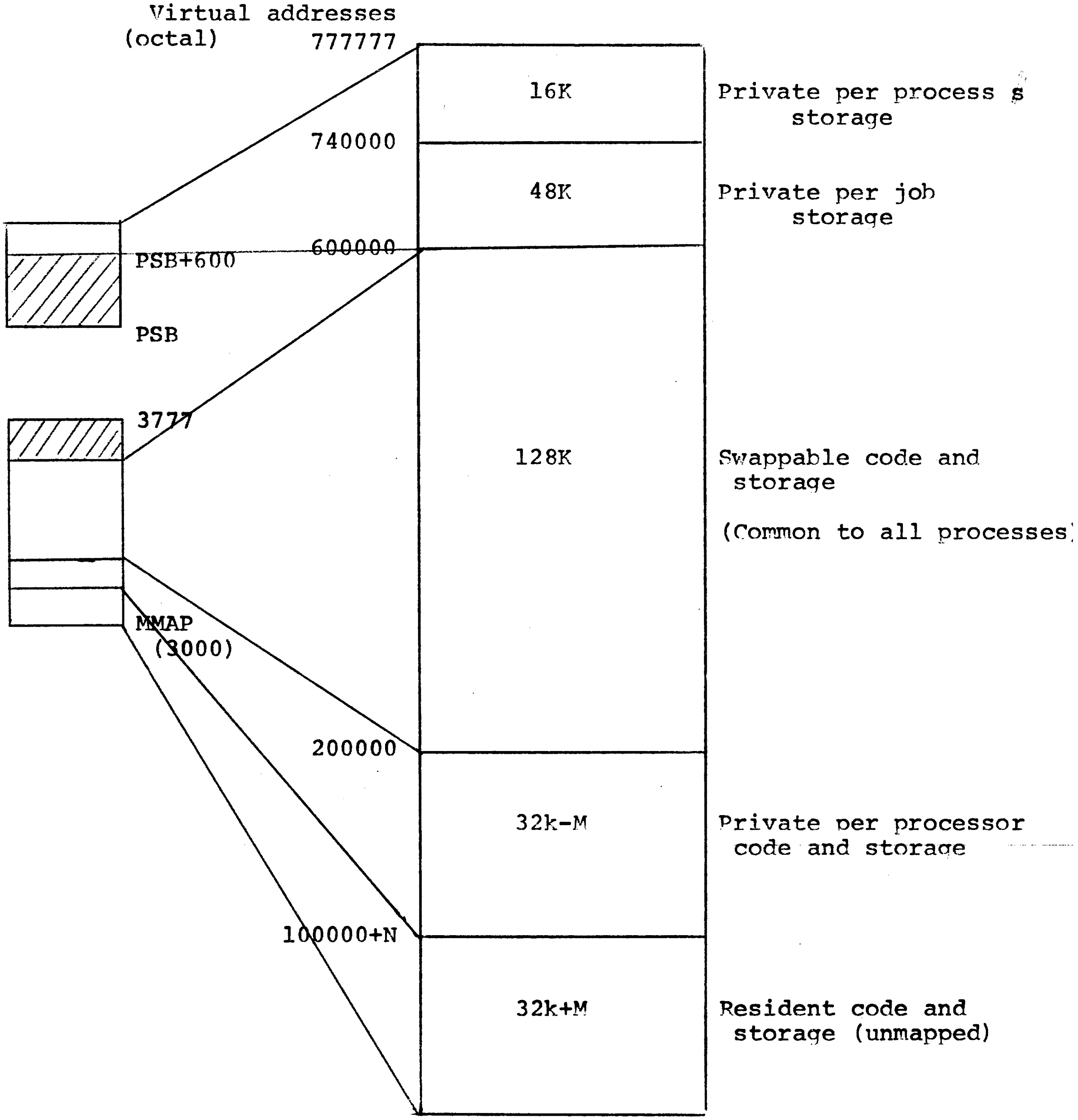


Fig. 2

Fig. 3 Monitor Map Storage Assignments

V.A.	Parameter	Description
0	*	AC's device reference cells, start locations, TENDMP
1000	*	JSYS transfer vector
2000	CST1	Core status table, part 1
3000	MMAP*	Monitor map
4000	CST0*	Core status table, part 0 (referenced by pager)
5000		Resident variable storage, usually large tables, e.g. drum status table, terminal buffers
20000	SPT*	Special pages table, referenced by pager
26000	SPT+SSPT	Resident variable storage, usually a few larger tables (e.g. CST2, CST3)
36000	RESLOC	Resident variable storage assigned at load time (via LS, GS macros, ASSIGN pseudo-op of assemblers and loader)
51000	RESMON,P1,P2,FFF	Resident code. The first 300(8) locations are patch space. FFF is the current next free patch location.
100467	C(100)	Exec mode DDT
101000	C(MONCOR)	First page used for swapping if DDT not retained.
104610	C(116)	Resident symbol table
124000	C(SWPCOR)	First page used for swapping if DDT is retained.
140000	PPRMA	First "private-per-processor" page; end of resident, unmapped area
200000	SWPMA*	First page of "swappable" area
200000	NRESBG	Swappable variable storage (assigned in arbitrary size blocks at load time via ASSIGN)
240000	SWPMP0	Filesystem swappable code

261000 to 272777	SWPMPC C(SWCEND)	Monitor swappable code; utility (FLIN, FLOUT, etc.) swappable code
377200	LWTEND	Running location of TENDMP used to bootstrap swappable code from DTA.
410000	NRPLBG	Swappable variable storage assigned in multiples of 512 words (1 page) via ASSIGN
600000	PJMA*	Job common area.
600000 to 603777	JSB	Job storage block, storage assigned in arbitrary size blocks via ASSIGN
604000	FREJPA	Pool of pages assigned dynamically for various purposes (e.g. file window pages, strings etc.)
740000	PPMA	Process private area
740000		Per-process pages used by swapper, map routines, FORK, PSI, etc.
750000 to 757777		Reserved for expansion of File directory
760000 to 767777	DIRORG	Currently mapped file directory
770000 to 774777	MDDT	DDT in monitor mode under time sharing
775000	UACPG	AC blocks (swapped to JSB)
776000	UPTA	User Page table
777000	PSB	Process storage block, storage assigned in arbitrary blocks at load time by ASSIGN.

*Address fixed or partially determined by hardware.

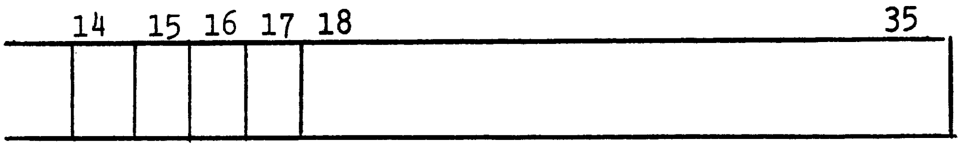
SECTION V - MONITOR TABLES AND DATA STRUCTURES

This section describes the format of the central tables used by the schedule and swapper. They include:

CST0 CST1 CST2 CST3	Four-part core status table
SPT SPTH	Two part special pages table
DST	Drum Status Table
JOBDIR JOBNAM JOBRT JOBPT	Job Tables indexed by job number
FKPGS FKSTAT FKWSP FKPGST FKOLDS FKPT FKINT FKINTB FKJOB FKNR FKTIME	Fork Tables indexed by system fork number
FKCNO	
BALSET NBP	Balance set tables

Storage Addresses

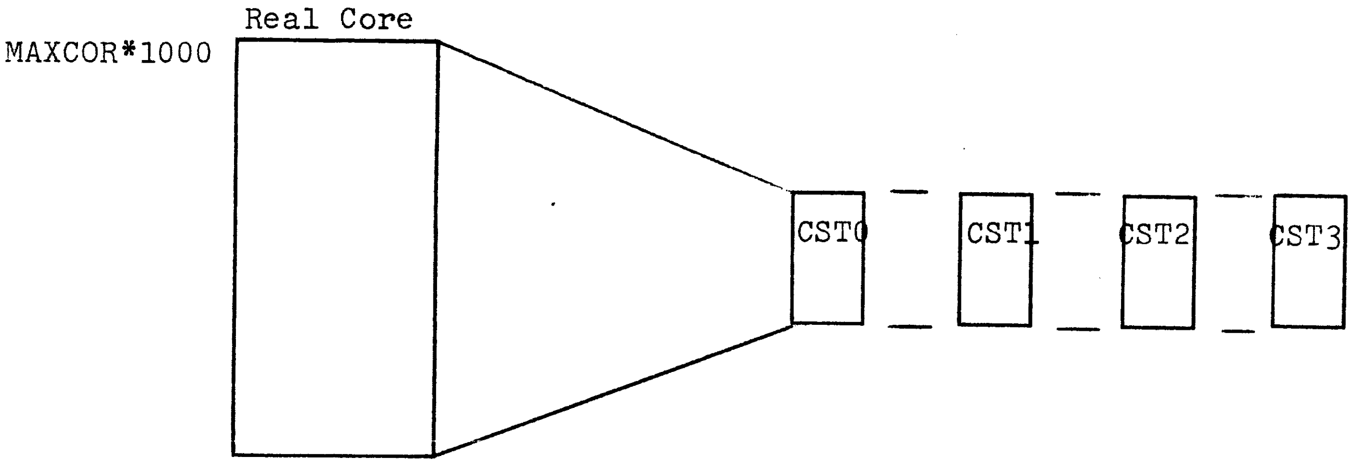
Throughout the system, storage addresses are used which identify the medium on which the storage exists as well as the location on that medium. These addresses are always 22 bits, and they exist in page tables, SPT, CST, etc. They are usually kept right justified in a storage word and are formatted as follows:



If bit 14 is 1, bits 15-35 are a disk address
If bits 14-15 are 0 and bit 16 is 1, bits 17-35 are a drum address
If bits 14-17 are 0, bits 18-35 are a core address
Disk addresses are in units defined by the disk driver module, e.g. 128 words for RP02 and Bryant disk systems.
Drum addresses are in units defined by the drum driver module, e .g. 512 words for Bryant drum.
Core addresses are in units of 512 words (one page), and thus are core page numbers addressing real core.

CST

The core status table maps real core; its four parts are parallel



CST0 - Format principally defined by pager,

0 8	9	10 35
AGE	M	PUR

AGE, Bits 0-8: Contents of pager age register at last AR reload if page in use, otherwise encoded to show state:

000 - on replaceable queue
010 - to be put on replaceable queue
020 - read completed
040 - write in progress
060 - read in progress

M, Bit 9 - Modified bit, set by pager on any write reference. Will be 1 if page has been written since last operation.

PUR, Bits 10-35 - Process use register; bit n is a 1 if process with core number n has referenced it.

CST1

0 13	14 35
LOCK CNT	BACKUP ADDRESS

LOCK CNT, Bits 0-13 - Number of reasons page is locked in core (e.g. page table containing other core addresses). Page will not be considered for swapping if bits 0-13≠0.

BACKUP ADDRESS, Bits 14-35 - Storage address of next level of storage for the page (i.e. DISK or DRUM) or 1000000 if unassigned

CST2 - Home map location

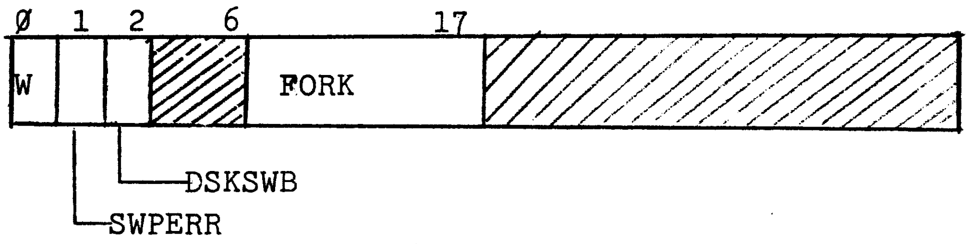
PTN	PN

or

Ø	SPTN

Gives the home map location for the page, i.e. the page table which contains the core address pointing to this page. If LH=0, the home map is the SPT, and the RH gives the SPT index. If LH=0, the home map is a page table or index block, PTN is the SPT index of that map, and PN is the page number within that map.

CST3 - Chain Word



This word is used for a variety of purposes, generally as a list pointer for groups of pages on various queues. The format shown above exists when the page is in use.

W, Bit 0 - Write in progress. This bit is 1 if the page was referenced and assigned while a write to swapping storage was in progress. The bit is cleared by the swapper when the write completes.

SWPERR, Bit 1 - Set if an unrecoverable error occurred when reading in this page from disk or drum.

DSKSWB, Bit 2 - Swap to disk requested by DDMP.

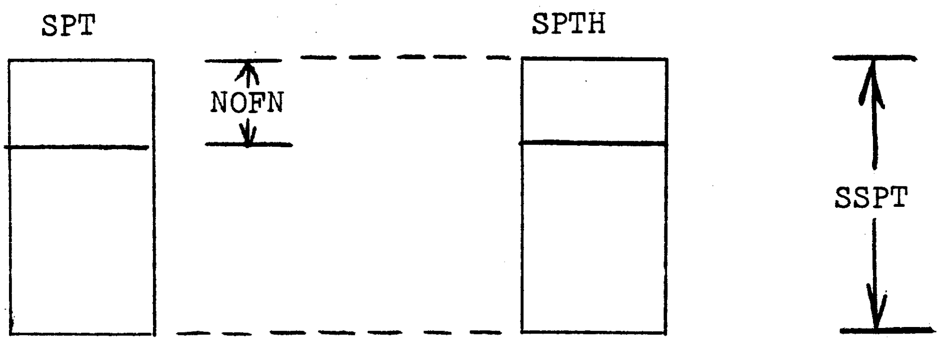
FORK, Bits 6-17 - Process to which this page is assigned, 7777 if not assigned.

When on the replaceable queue, the LH and RH contain backward and forward list pointers respectively.

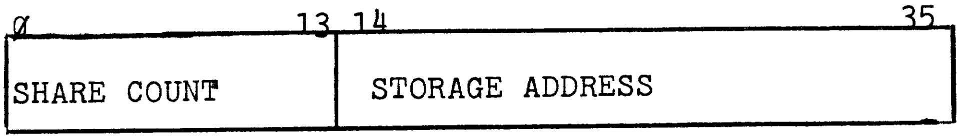
When on a swapping device queue, the RH contains a forward list pointer and B0 is 1 if write, 0 if read.

SPT, SPTH

The SPT (special pages table) is referenced directly by the pager; SPTH is parallel to it but is referenced only by the software. The first part of this table (of length NOFN) is used for index blocks for open files, and an index into this part is often referred to as an OFN (open file number). The remainder of the table is used for PSB's, JSB's, UPTs, and shared file pages.

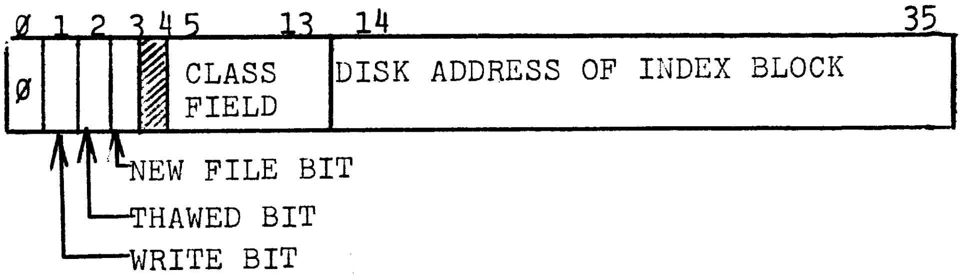


In either part of SPT, a cell which is in use contains a storage address and a share count in the format



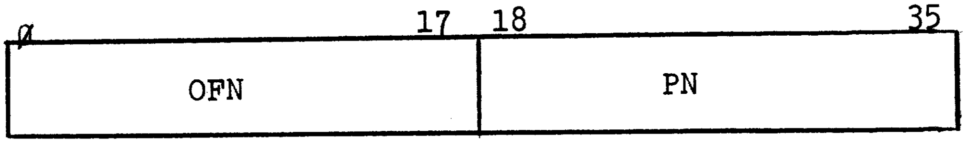
For open files, the share count is indexed for each opening of the file and for each shared page within the open file. For other entries, the share count is indexed for each sharing of the page.

The format of SPTH entries is different in each of the two parts. In the OFN part, the format is:



A file is opened by searching the OFN part of SPTH for the index block address. If the address is found and the write and thawed bits are legal, it is a shared opening and the same index is used. If the address is not found, a new entry is made (free slots have -1 in SPTH).

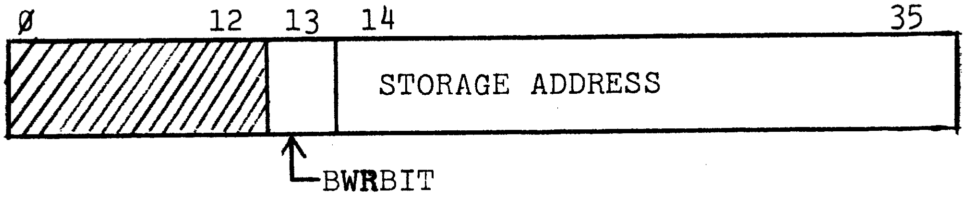
In the other part of SPTH, the word serves to show where the page came from. For a shared file page, this is indicated by



where PN is the page number within open file OFN. For PSBs, JSBs, and UPTs, the SPTH cell contains 0. The free slots in the second part are on a list chained through SPT, and the free list pointer resides in FRESPT.

DST

The DST (drum status table) is indexed as a function of the drum address. The routine GDSTX converts a drum address into a DST index. The DST hold the address of the next lower level of storage (usually disk) for the page stored at that address on the drum. The format of an entry is:

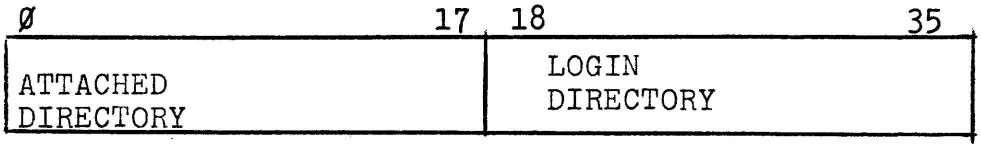


BWRBIT is 1 if the page has been changed since being read from the lower level storage. The page will not be copied back onto the lower level storage if BWRBIT is 0 when the page is no longer in use. A slot not in use contains -1.

JOB TABLES

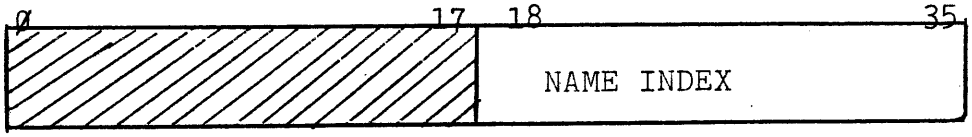
These tables are indexed by job number

JOBDIR



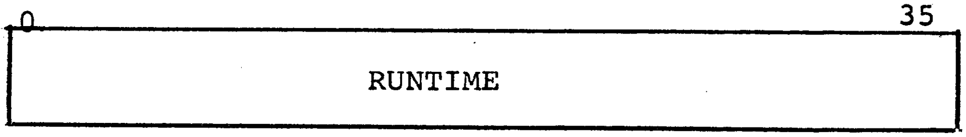
This cell gives the number of the attached directory and login directory (user number) for the job.

JOBNAM



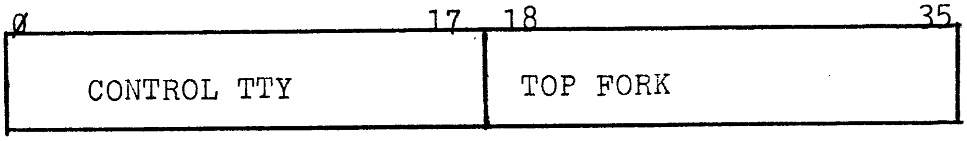
The RH of this cell is an index into the subsystem name tables (SNAMES, STIMES, etc.) indicating what subsystem the job is running. This is for statistics only and is not used by the monitor.

JOBRT



This cell contains the total run time of the job (sum of all forks) in milliseconds. If this cell contains -1, the job does not exist.

JOBPT



The LH contains the number of the controlling terminal, or -1 for a detached job. The RH contains the index of the top fork of the job.

Free job slots are chained on a list through JOBPT, and the free list pointer is FREJOB.

FORK TABLES

A fork is identified by an index into these tables.

FKPT

0 17	18 35
CURRENT LOCATION	LIST POINTER

When a fork is on the wait-list or go-list, the RH contains the list pointer to the next fork, and the LH contains WTLST or GOLST respectively. When the fork is in the balance set, the RH contains the balance set index, and when the fork is running, the LH contains the processor number of the processor on which it is running (currently always 0).

FKSTAT

For a fork on WTLST:

Ø 17	18 35
TEST DATA	TEST ROUTINE

For a fork not on WTLST:

Ø 17	18 35
QUEUE NUMBER	PRIORITY VALUE

FKPGS

Ø 17	18 35
PAGE TABLE	PSB

Contains the page table and PSB
(SPT indexes).

FKWSP

Ø 17	18 35
TAV	N

LH: average interfault interval in milliseconds

RH: number of pages now assigned to fork

FKPGST

Ø 17	18 35
TEST DATA	TEST ROUTINE

For a fork in balance set wait status.

FKOLDS

Receives C(FKSTAT) on initiation of a pseudo-interrupt.

FKINT

Ø 17	18 35
BITS	DATA

Pseudo-interrupt communication register. LH contains bits recording type of request (see SCHED p.43, NEWFKF, NEWJBF, etc.). RH contains data if necessary.

FKINTB

Ø 35

Buffer for pseudo-interrupt channel requests

FKJOB

Ø 17	18 35
JOB NUMBER	JSB

The job number and JSB (SPT index) for the job to which this fork belongs.

FKNR

Ø 17	18 35
AGE	WORKING SET EST

The current working set size estimate and the age counter (for the pager age register) for this fork.

FKTIME

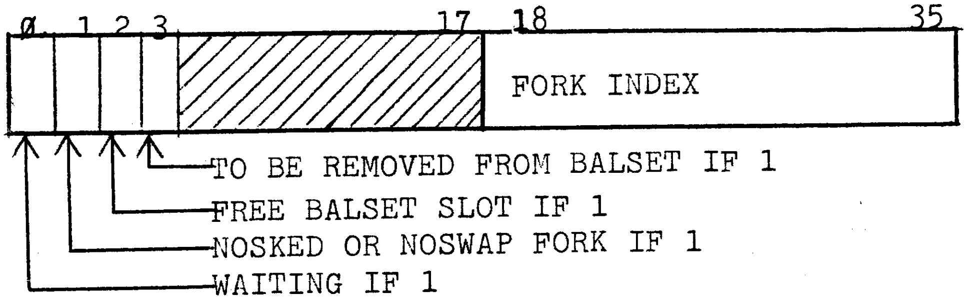
The time (TODCLK) at which the fork was put on its current run queue.

FKCNO

Ø 17	18 35
▨	CORE NUMBER

The core number for this process, used to set a bit in the pager process use registers.

Balance Set Tables

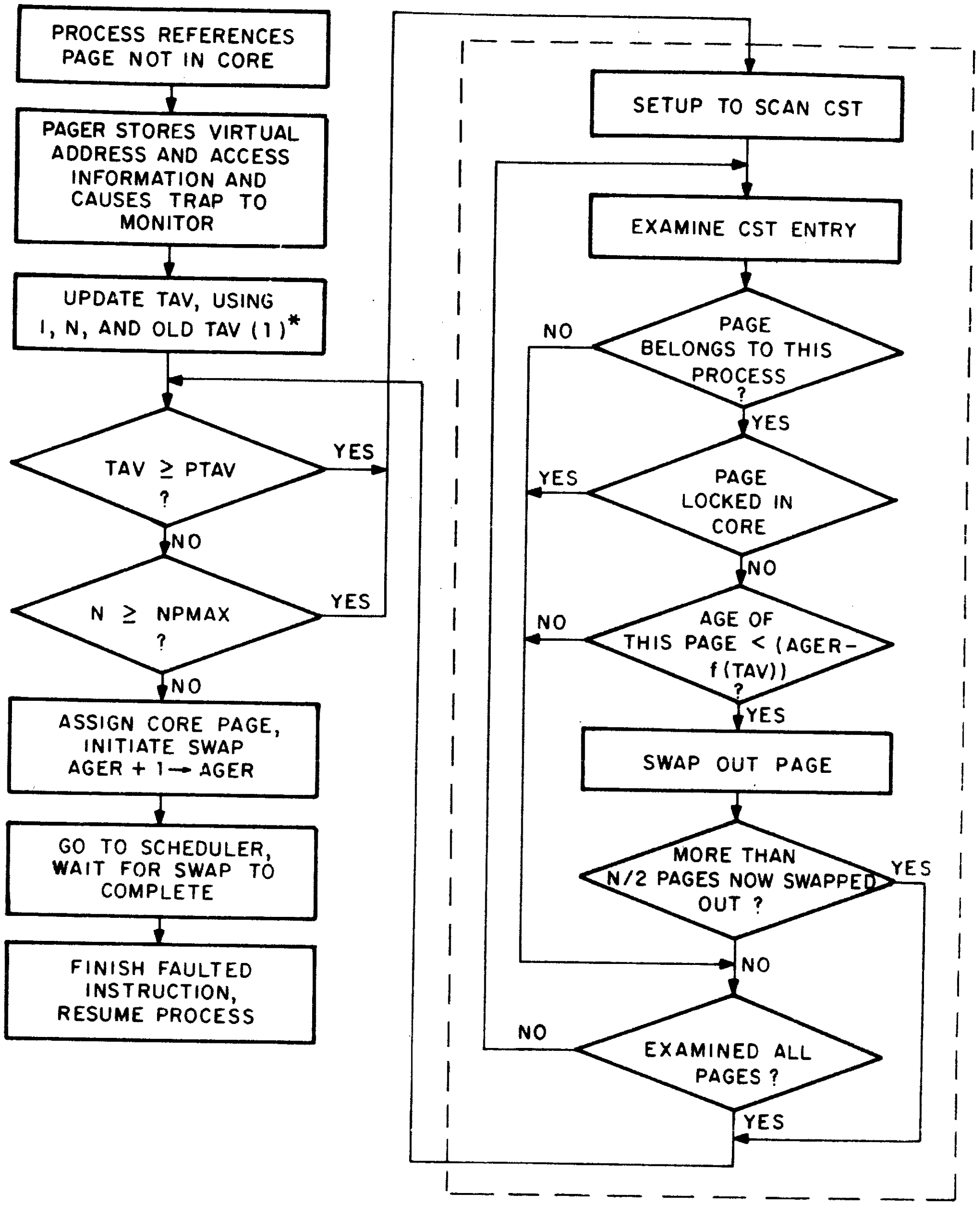


BALSET

NBT

Contains runtime in ms. since entering balance set.

FLOW OF PROCESS CORE MANAGER


PROCESS REFERENCES PAGE NOT IN CORE
PAGER STORES VIRTUAL ADDRESS AND ACCESS INFORMATION AND CAUSES TRAP TO MONITOR
UPDATE TAV, USING I, N, AND OLD TAV (1)*
TAV ≥ PTAV ?
YES
NO
N ≥ NPMAX ?
YES
NO
ASSIGN CORE PAGE, INITIATE SWAP AGER + 1 → AGER
GO TO SCHEDULER, WAIT FOR SWAP TO COMPLETE
FINISH FAULTED INSTRUCTION, RESUME PROCESS
SETUP TO SCAN CST
EXAMINE CST ENTRY
NO
PAGE BELONGS TO THIS PROCESS ?
YES
YES
PAGE LOCKED IN CORE
NO
NO
AGE OF THIS PAGE < (AGER - f(TAV)) ?
YES
SWAP OUT PAGE
MORE THAN N/2 PAGES NOW SWAPPED OUT ?
YES
NO
NO
EXAMINED ALL PAGES ?
YES

SYMBOLS USED IN FIGURE

TAV - AVERAGE TIME BETWEEN PAGE FAULTS FOR THIS PROCESS
I - INTERVAL SINCE LAST PAGE FAULT FOR THIS PROCESS
N - NUMBER OF PAGES IN CORE FOR THIS PROCESS
PTAV - FIXED PARAMETER. PROCESS IS EXACTLY AT WORKING SET SIZE WHEN TAV = PTAV
NPMAX - MAX N ALLOWED FOR REASONS OF PHYSICAL CORE SIZE
AGER - PAGER AGE REGISTER
CST - CORE STATUS TABLES

* (1) $\frac{TAV * N + I}{N + \rightarrow} \rightarrow TAV$

Adding a New Device

There are four steps to adding a new device to TENEX. Three of these involve additions to the filesystem the other two involve additions to PARAMS.MAC and PISRV.MAC.

The first step is to write the device dependent code. An existing file of device dependent code may be used as a model. The body of the file should be enclosed in a conditional assembly on the definition of the symbol devN; where "dev" stands for the device name. Furthermore, the body of the file should be enclosed in a BEGINP/BENDP pair. Thus the file looks like:

```
IFDEF devN,<
BEGINP dev ...other identification...
 ...body of file...
BENDP>
```

References to routines in this file are made through the device dispatch table and the PI service routine. The standard name for the PI service routine entry is devSV. The PI service routine is called with "JSYS devSV".

The device dispatch table is referenced by "PUSHJ P,@offset(DEV)" with DEV pointing at the base of the dispatch table. Offset depends on the operation wanted. Thus the device dispatch table should contain addresses of routines to be called with PUSHJ P,.

The layout of the device dispatch is shown in FILE.FAI. Offsets into the dispatch vable are parameterized to allow change. The names of the offses are given below with a brief description of the charcteristics of the routine poited to by that entry.

DLUKD

This points to the directory setup routine. It should return no skip for directoryless devices. One skip for directory devices if failure. Two skips in directory device success. The routine should setup whatever conditions are needed for a subsequent call to NLUKD.

NLUKD

This points to the name lookup routine. If either of the flags NREC or NREC1 are set, the routin should not perform recognition. If the flag UNLKF is set, the routine should establish the proper context for a subsequent call to ELUKD.

ELUKD

This points to the extension lookup routine. The same input conditions apply here as for NLUKD.

VLUKD

This points to the version lookup routine. If the flag UNLKF is set, the routine should establish a context for subsequent manipulation of information about this file.

PLUKD, ALUKD, and SLUKD

These point to the protection, account, and status insertion routines respectively.

OPEND

This points to the file opening routine.

BIND

This points to the byte input routine.

BOUTD

This points to the byte output routine.

CLOSD

This points to the file closing routine.

REND

This points to file rename routine.

DELD

This points to the file delete routine.

DMPID and DMPOD

These point to dump input and output respectively.

MNTD and DSMD

These point to routine for mounting and dismounting a device respectively.

INDD

This points to a routine to initialize the directory of a device.

MTPD

This points to the routine for doing MTOPR operations.

GDSTD and SDSTD

These point to routines for getting and setting device status respectively.

DIRECT.FAI

The file DIRECT.FAI consists of 3 blocks named DEVICE, LOOKUP, and DIRECT. These blocks constitute the routines for doing device name lookup, device independent file name lookup, and disc file directory management.

DEVICE

This block contains the routines for performing operations concerning the device tables. One routine not included in this block which also manipulates device tables is CHKDEV appearing in the file JSYS.FAI.

There are four device tables used and maintained by these routines, and one table that is used for initialization. These tables are named DEVNAM, DEVCHR, DEVUNT, DEVDSP, and DEVINT. The first four tables are initialized by the routine DEVINI when the system is first started.

(DEVNAM)DEVNAM

This table has the device name in SIXBIT for each device in the table (e.g. DTA0). This table is searched by the routine DEVLUK to find entries in parallel tables for a device with a particular name.

(DEVCHR)DEVCHR

This table has the device characteristics word as given by DVCHR except for bit 5 (available to this job). The device type field of this word, in conjunction with DEVUNT, is searched by CHKDEV to find the entry for a particular device given a TENEX device designator.

(DEVUNT)DEVUNT

The left half of this word contains the job number to which this device is assigned. -1 is used to indicate that the device is unassigned.

The right half of this word contains the unit number of this device. E.g., DTA2 would have a 2 here. Unitless devices such as DSK have -1 here.

(DEVDSP)DEVDSP

The right half of this word contains the location of the device dispatch table for this device. A device dispatch has entries for each primitive device dependent routine. A more complete description of the device dispatch table appears elsewhere.

The left half is not used but is reserved for additional information about a particular device.

(INIDVT)INIDVT

This table is used to initialize the device tables. Each entry in INIDVT is three words long and consists of the name prefix (e.g. DTA), unit count, dispatch table location, DEVCHR without unit number. This table is expanded by DEVINI to form the other device tables when the system is started up.

Directory Format

Each TENEX (disc) file directory consists of eight* sequential pages of the file "<SYSTEM>DIRECTORY.;1". These eight pages start on the page in the file equal to eight times the directory number. Thus directory 1 starts on page 10 (octal) and directory 136 starts on page 1360 (octal). The eight pages which constitute a directory are mapped into the monitor process private area starting at location DIRORG (currently 760000 octal). Once a directory is mapped, it is referenced directly.

A directory is divided into three areas. The area at the beginning of the directory provides fixed storage for various pointers, counters, etc. The middle area from location DIFREE to the location specified by the contents of FRETOP is a dynamically assigned free storage area. The area from the end of the directory down to the location specified by FRETOP is used for a sorted symbol table of pointers to file name strings and account strings which are stored in the dynamic area.

* Subject to imminent change

The Fixed Area

DIRLCK

This word is a lock used to prevent simultaneous conflicting accesses to the directory by multiple processes. This word is -1 if the directory is not in use, and not -1 if busy.

DIRUSE

This word is used to hold the process number of the last locker of the directory. Used for debugging purposes.

DIRNUM

The directory number is held here. Before a directory is used, this cell is checked for equality with the desired directory. In this case it is not necessary to remap the directory. This cell is also checked after mapping to verify that the directory has not been horribly smashed.

SYMBOT

This cell contains a relative pointer to the low end of the current symbol table.

SYMTOP

This cell points just above the end of the current symbol table. The difference between SYMBOT and SYMTOP is the length of the table. If SYMBOT equals SYMTOP, the table is empty.

DIRFRE

This is a block of 7 words as required by ASGFRE/RELFRE routines. The LH of word 0 points to the first free block in the dynamic area. Word 1 is a lock word. Word 2 has a count of the total free words in the dynamic area. The remaining words are used for temp storage by the free storage routines.

FRETOP

This cell contains a pointer to the end of the free storage area.

DIRDPW

Default protection word. The contents of this word are inserted into FDBPRT cell of each new FDB created in this directory.

DIRPRT

Directory protection. The contents of this cell are used to determine if a particular user has appropriate access to this directory.

DIRDBK

Default backup.

DIRGRP

Group mask for this directory.

SPARE

These cells are provided for expansion of storage in the fixed area without changing its size. This avoids regenerating all directories due to such an addition.

DIRLOC, DIRINP, DIRINC, DIRMSK, and DIRSCN

These cells are used for temporary storage by various directory routines. DIRLOC and DIRSCN typically contain pointers to things in the directory. DIRINP contains a pointer to the input string for lookup. DIRMSK contains a mask covering real characters in the last word of the input string. DIRINC has the current increment for scanning the symbol table.

The Free Area

Storage in the free area is allocated in variable sized blocks. Each block always has as its 0-th word a header. The right half of the header contains the size of the block including the header word. The left half of the header indicates the current use of the block. For a free block, the left half is a pointer to the next free block. For non-free blocks, specific markers are used to indicate what the block contains. This information is largely redundant and thus serves to improve the chances of detecting malfunction and recovering therefrom.

All free blocks are chained through the left half of each header. The chain is anchored in the left half of DIRFRE. Furthermore, the free chain is kept in order of numerically increasing addresses so that when a block is released it can be merged with an adjacent free block.

The Symbol Table Area

The symbol table contains two types of entries: file name entries and account string entries. The entry type is indicated by bits 18-20 of the entry. The left half of the entry points to a file name block or account string block in the dynamic area depending on the entry type. The table is sorted by entry type and then alphabetically. For file names, bits 21-35 point to the first FDB with that file name. These bits are not used for account strings.

FDB Format

Each file has an associated FDB (File Descriptor Block) in some directory. The format of an FDB is as follows.

(FDB format goes here)

LOOKUP

This block contains device independent routines for looking up file names. Each routine except the top level one (GETFDB) has two entry points. One is used to establish a context for the higher level lookup routine to work in. The other is used for a direct call. In addition, the name and extension lookup routines have another entry if recogntion is desired. These routines also take care of the indexing needed for GNJFN .

NAMLKX, NAMLUK, and NAMLUU

These are the non-recognize, recognize and context setting entries respectively for performing file name field lookup.

Arguments: in A, a standard lookup pointer or 0 if first name wanted; in DEV, device dispatch table location; in RH(FILDDN(JFN)), the directory number in which to look; bits DIRSF or NAMSF set to one will cause indexing.

Calls: DLUKD(DEV), NLUKD(DEV).

Returns +1 if unsuccessful. Smashes A, B, C, D, and miscellaneous flags.

Returns +2 if successful entry from NAMLUU with context established.

Returns +2 if ambiguous.

Returns +3 if successfull. The complete name will be appended or substituted for the input argument if appropriate.

EXTLKX, EXTLUK, and EXTLUU

These are the non-recognize, recognize, and context setting entries respectively for performing file extension field lookup.

Arguments: in A, standard lookup pointer or 0 if first one wanted; in DEV, device dispatch table location; in RH(FILDDN(JFN)), the directory number in which to perform the lookup; in LH(FILNEN(JFN)), pointer to the name string; bits DIRSF, NAMSF, or EXTSF set to one will cause indexing.

Calls: NAMLUU, ELUKD(DEV).

Returns same as NAMxxx above.

DIRECT

This block together with DISC contains all the routines which manipulate disc directories. In addition, user index routines appear in this block except for CRDIR and GTDIR JSYS's which appear in the file JSYS.

DIRCHK

This routine performs access check to the currently mapped directory.

ACCCHK

Performs access check to a specific file.

DIRLUU, DIRLUK, and DIRLKX

These are the context setting, recognition, and non-recognition entries for looking up a directory string in the user index.

GDIRST

Yields the location of a directory name string block given the directory number. This used by several routines needing the name string.

INIBLK

Initializes a directory.

GETDDB

Yields the location of a directory descriptor block (DDB) given a directory number.

HSHLUK

Performs a hash table lookup in the user index to translate a directory number into a DDB location and a hash table location.

INSACT and INSACO

Insert an account number/string into an FDB.

INSPRT

Inserts a protection word into an FDB.

FDBINI

Initializes an FDB.

SETDIR

Maps and locks a given directory.

MAPDIR

Maps a given directory.

MDDDIR

The device dependent directory lookup routine for multiple directory devices (disc). This interfaces SETDIR to NAMLUK.

MDDNAM

This is the device dependent name lookup routine for MDD devices.

MDDEXT

This is the device dependent extension lookup routine for MDD devices.

MDDVER

This is the device dependent version lookup routine for MDD devices.

LOOKUP

This routine looks up a file name or account string in a directory.

FREE.FAI

This file contains routines for managing storage associated with the file system.

ASGFRE

Assigns a block of storage in one of several areas as specified by the call. This routine assigns storage in the dynamic area of a directory.

RELFRE

Releases a block of storage assigned by ASGFRE.

ASGPAG

Assigns a page in the job area of a process.

RELPAG

Releases a page in the job area.

ASGDFR and RELDFR

These routines provide ways of calling ASGFRE and RELFRE respectively for space in a directory's dynamic area.

GCDIR

Garbage collector for a directory dynamic area. Compacts storage to eliminate fragmentation.

GTJFN.FAI

This file consists of one block with the same name as the file and contains code for GTJFN and GNJFN JSYS's.

.GTJFN

This is the code for GTJFN JSYS. It reads, parses, recognizes a file name string and returns a Job File Number (JFN) for the specified file to the user's program.

.GNJFN

This is the code for the GNJFN JSYS. It associates a JFN with the "next" file after the file with which it is currently associated.

ASFJFN and ASGJF1

These routines assign an unused JFN for .GTJFN and mark it as "name being collected".

RELJFN

This routine releases a JFN and the storage associated with it.

IO.FAI

This file contains two blocks; IO and CONVER. These two blocks constitute most of the sequential, random, and dump input/output routines.

IO

This block contains code for all sequential and random byte and string and dump input/output JSYS's.

.BIN, PBIN, .SIN, .RIN, .BOUT, .PBOUT, .SOUT, .PSOUT, .ROUT, .DUMPI, and .DUMPO

These are the entries to the routines for the corresponding JSYS's.

CHKJFN

This routine checks all source/destination designators for validity, converts primary i/o designators and indicates to the caller the type of designator.

UNLCKF

This routine undoes what CHKJFN does.

CONVER

This block contains code for the integer conversion routines NIN and NOUT.

JSYS.FAI

This file contains code for most of the file system and directory JSYS's.

DECTAP.FAI

This file contains all device dependent code for driving a standard DEC TD-10 DECtape controller. The file contains two blocks.

DECTAP

This block contains only routines which do not run at interrupt level. All filesystem primitives are implemented for DECtape in this block.

DTA

This block contains those DECtape routines which may run at interrupt level including data, search and flag interrupt routines and the controller scheduler.

DISC

This file contains the device dependent routines for device DSK. Because of their length, the blocks containing routines for devices TTYn, NIL, and string pointers and for filesystem initialization are also included in this file.

DISC

This block contains routines to implement the device dependent portions of the file system JSYS's down to the level of map changing and directory manipulation. The actual input/output transfers occur as the result of page faults .

STRING

This block contains routines to implement sequential input/output operations for string pointers.

TTY

This block contains code which interfaces the file system to the terminal routines .

NIL

This block contains code to implement sequential input/output operations to the NIL device.

INIT

This block contains code for initializing the file system at system startup time and for initializing a new job with respect to the file system.

DISPLA.FAI

This file contains routines to implement the JSYS's for running the ES LDS-1 display processor and the interrupt routines and scheduler for same. These routines are not strictly part of the file system but are included in the file system assembly simply because they were written by the same author and utilize some of the assembly environment provided by that assembly.

LINEPR.FAI

This file contains device dependent routines for driving BBN's lineprinter. This is an oddball device and the code is probably not pertinent to any other installation.

MAGTAP.FAI

This file contains device dependent routines for operating a DEC TM10A magtape controller.

NETWRK.FAI

This file contains device dependent routines for incorporating the ARPA network into the TENEX file system. Routines in this file are independent of the exact IMP interface and communicate with routines in the file IMPDV.MAC on the level of "send and STR to this host for these socket numbers" etc.

PLOTTE.FAI

This file contains device dependent routines for driving a Calcomp model 563 incremental plotter from the file system.

PTP.FAI

This file contains device dependent routines for driving the paper tape punch.

PTR

This file contains device dependent routines for driving the paper tape reader.

ACCT1Ø

A PDP-1Ø ACCOUNTING SYSTEM

I. INTRODUCTION

II. BASIC STRUCTURE OF ACCT1Ø

III. COMMANDS TO ACCOUNTS

IV. COMMANDS TO REPORTS

V. ADJUSTMENTS PROGRAM

VI. HOW TO USE ACCT1Ø

APPENDIX

A. FACT FILE FORMATS

B. DETAILED STRUCTURE OF ACCT1Ø

C. YEAR-TO-DATE FILE FORMAT

D. HISTOGRAM

E. HOW TO MODIFY ACCT1Ø

ACCT1Ø - A PDP-1Ø ACCOUNTING SYSTEM

I. INTRODUCTION

ACCT1Ø is a subsystem program which performs accounting for the PDP-1Ø TENEX time sharing system. Information as to logins, logouts, cpu and console time used, and file storage is located in the binary FACT files, which can be found under directory <ACCOUNTS> and a duplicate under directory <TIMESHEET>. These files serve as the raw data for ACCT1Ø. More detailed information about the structure of the FACT files and of ACCT1Ø can be found in the Appendix.

Associated with each user and every account number that he uses is an acct-user entry in ACCT1Ø. Computer charges are made for each of these acct-user entries. Compiling charges for any one user or a certain account number may involve several acct-user entries. ACCT1Ø allows you to adjust those charges, insert new acct-user entries, and add extra special charges such as for a home teletype. Inserting new acct-user entries is permissible only if the user ID code is known to the system, i.e. it is in the user ID directory.

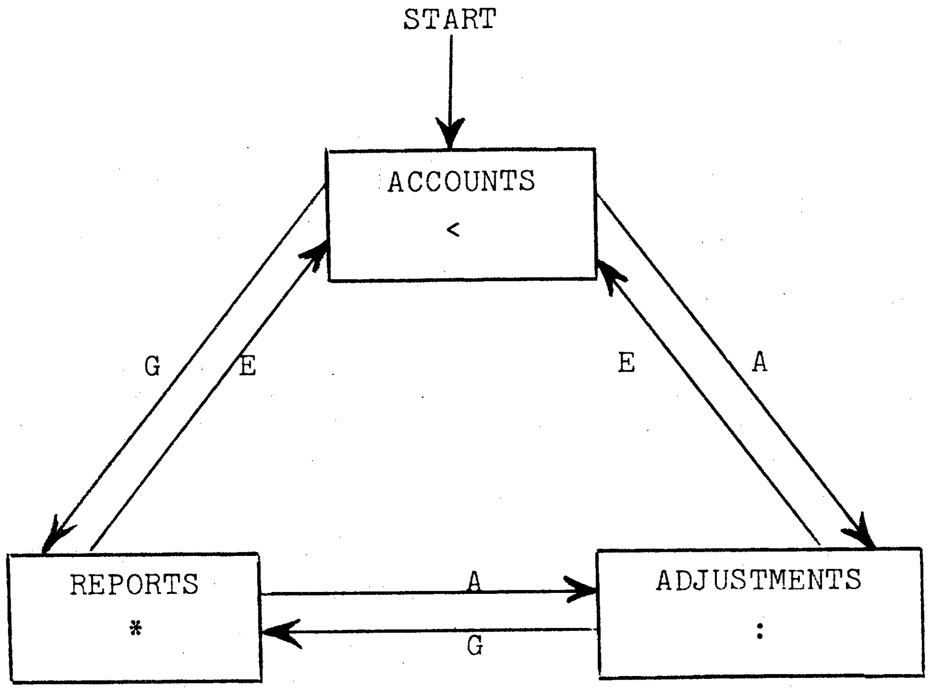
The system distinguishes two types of users: numeric users and alphanumeric users. A numeric user may use only numeric account numbers, while an alphanumeric user can use alphanumeric strings as account numbers. Therefore, acct-user entries are ordinarily either numeric or alphanumeric. However, there may be cases (usually due to actions taken by the operator, like manipulation of files) when the account number and the user of an acct-user entry are not of the same type. Such entries are flagged on the accounting summary sheet for that user and ACCT1Ø has the capabilities to have such situations rectified, if desired.

The charges for every acct-user entry are computed using one of three sets of rates: government rates, commercial rates, and no-charge rates (i.e. zero charges). For numeric users, each account number is assigned one of the three rates. For alphanumeric users, the rates can be assigned according to the user as well as the account number. These rate assignments are incorporated in the program. Hence, a change in the rates themselves or in the rate assignments entails appropriate changes in the program, as described in the Appendix.

II. BASIC STRUCTURE OF ACCT1Ø

ACCT1Ø conceptually consists of three programs: ACCOUNTS, REPORTS, and ADJUSTMENTS, as shown in Fig. 1. Starting ACCT1Ø transfers control to ACCOUNTS which is used to initialize the accounting system and read in the binary FACT files; REPORTS produces all accounting reports; and ADJUSTMENTS allows you to make adjustments to user charges. Associated with each of the three programs is a ready character (see Fig. 1) that appears on the teletype indicating that the program is in ready status waiting for you to type in a one-character command. The program automatically performs recognition on the command. In case of nonrecognition the character ? is typed and ready status is assumed once more. Transfer of control among the three programs is facilitated by means of the three commands shown in Fig. 1. Within each program, typing CONTROL-E (↑E) closes any read file that might be open and returns you to the ready status of that program. (In ADJUSTMENTS it returns you to initial command status; see Section V.) While entering parameter values, typing CONTROL-A (↑A) n times erases the last n characters typed in.

Following is a list of the available commands. Characters typed in by the user are shown underlined. Note that all file names are in TENEX format. While entering a file name, typing ALT MODE will perform recognition on the file name. A carriage return must be entered after recognition.



Commands

EXIT

GO TO REPORTS

ADJUST CHARGES

Fig. 1 Basic Structure of ACCT1Ø

III. COMMANDS TO ACCOUNTS

```
<Z INITIALIZE ACCOUNTING SYSTEM
 PAY PERIOD IS:  NOV  1, 1971 TO NOV  15, 1971↙

 ACCOUNTING INFORMATION?(Y,N)Y
 LOGOUT DATA FILE?(Y,N) Y
 BINARY-FILE NAME: LOGDATA↙
```

This command initializes the accounting system and should be executed before using the R command. Giving this command at any other time reinitializes the accounting system. The pay period can be any text of up to 28 characters terminated in a carriage return. The pay period can be reset at any time using the P command below.

The two questions that follow the pay period specification above cause two parameters to be set in the program. Once set these parameters can be reset only be reinitializing the accounting system using the Z command. The answers are typed as Y for yes and N for no.

Typing Y to the first question sets a parameter which upon reading a FACT file (using the R command) causes cpu and console time charges for each acct-user entry to be output onto the currently open output ASCII file OUTFILE (see command W). Typing N to the question will not cause any such output.

Typing Y to the second question, followed by specifying a file name LOGDATA, opens LOGDATA as a binary output file and sets a parameter which upon reading a FACT file causes logout entries to be output onto LOGDATA. (As can be seen in the Appendix Fig. A-3, a logout entry consists of 5 words; the last 4 words are output onto LOGDATA.) The data on LOGDATA could be used later for statistical computations concerning the usage of the time sharing system. The file LOGDATA is automatically closed upon typing either command G or A. Typing N to the second question will not cause the following line to be typed and no logout data is output.

Giving the Z command also causes a message of the form 'FACT FILE RECORD FOR PAY PERIOD NOV 1, 1971 TO NOV 15, 1971' to be output onto OUTFILE.

```
<PAY PERIOD IS:  NOV  16, 1971 TO NOV  30, 1971↙
```

This command allows one to reset the pay period.

```
<WRITE ONTO FILE:  OUTFILE↙
```

This command opens OUTFILE as an ASCII write file, then closes the previously open output ASCII file. The initial setting for OUTFILE is the teletype TTY:. Note that in order to close an open OUTFILE you must give the W command again and open another file (perhaps TTY: or any other file). All comments and error messages from ACCOUNTS and all cost summaries from REPORTS are written onto OUTFILE as they are generated.

```
<READ FACT FILE:  FACTFILE↙
```

Reads the binary FACT file FACTFILE and stores information about jobs, cpu and console time used, and file storage. Error messages and accounting information (if requested in the Z command) are output onto OUTFILE. The FACT files for the pay period should be read in chronological order. Commands G and A should not be given before all desired FACT files are read in.

```
<FILE DISCOUNT RATE: .75↙
```

Reads in a decimal which is used to multiply the cost of file storage for the pay period. The initial setting of the discount rate is 1.Ø, which is equivalent to no discount.

```
<HISTOGRAM
 OUTPUT FILE:  OUTFIL2↙
```

Outputs a histogram of cpu and console time used per session for that pay period on OUTFIL2. (As OUTFIL2 is opened for output, OUTFILE is closed. After the histogram is output, OUTFIL2 is closed and TTY: opened as with the command W.) The format structure of OUTFIL2 can be found in the Appendix. This command should be given only after the desired FACT files had been read in.

```
<DATE (TENEX FORMAT): 12Ø45Ø15654Ø↙
 I-NOV-71  1Ø:44:32
```

Takes a 12-digit octal number as a date in TENEX format and gives the corresponding local time.

```
<BINARY-FILE LISTING

 WRITE ONTO FILE:  OUTFILE↙
 READ BINARY FILE:  BINFILE↙
```

Outputs the binary file BINFILE as octal numbers on the output file OUTFILE. OUTFILE is then closed and TTY: is automatically opened as the new output file. This command is useful in looking at the contents of FACT files, or any binary file.

<GO TO REPORTS

Transfers control to the REPORTS program. The transfer is successful only if there are charges to be reported. (See the important note below.)

<ADJUST CHARGES

Transfers control to the ADJUSTMENTS program. (See the important note below.)

Important Note for Commands G and A

If either of the commands G and A is given after reading any FACT files (using the R command) the message 'CLASSIFICATION OF USERS AND ACCOUNTS ETC.' is written on the teletype, and the following sequence of actions takes place automatically:

a) Accounting for jobs not logged out.
b) The file LOGDATA (see command Z) is closed, if applicable.
c) If accounting information was asked for in command Z, then all alphanumeric accounts and their corresponding numeric equivalents are output onto OUTFILE. (In the accounting information, alphanumeric account strings are represented by negative numbers.)
d) Classification of users and accounts, etc.

Useful Hint

Suppose that you have 5 FACT files in your pay period. Call these files Fl, F2, F3, F4 and F5. Suppose you wish to see the results of the accounting after F3 has been read. If you wish later to read F4 and F5, then in order to avoid reading all 5 files again, you should SAVE your program immediately after reading F3, say on file FIL.SAV. Then you could CONTINUE and get your accounting up till F3. Then if you wish to read F4 and F5 to form a cumulative record for all 5 files you should GET FIL.SAV, START it and Read F4 and F5.

IV. COMMANDS TO REPORTS

All accounting reports are written onto OUTFILE.

*COST SUMMARY

Produces a cost summary by category: government, commercial, chargeable overhead, non-chargeable, computer center.

*TOTAL COST

Produces a total cost sheet summary by account number for all numeric users and by user ID code for all alphanumeric users. (Numeric accounts are listed in ascending numeric order and user ID codes are listed alphabetically.)

*FINAL COST SUMMARY FOR ACCOUNTING

Same as T above except that all free accounts are excluded. (Free accounts include the non-chargeable and computer center accounts.)

*JOB SUMMARIES

Produces detailed charge summaries by account number for numeric users and by user ID code for alphanumeric users.

*USER COST SUMMARIES

Produces detailed charge summaries for each user ID including numeric and alphanumeric users.

*DO ALL COST SUMMARIES

Produces summaries done by commands C,F,T, and J.

```
*INDIVIDUAL JOB SUMMARY FOR (N,U): N 345678↙
                                or U USERID↙
```

Produces a summary for either the account number or the user ID typed in. N indicates a numeric account number and U indicates a user ID code. The space after N or U is supplied by the program. If anything other than N or U is typed initially, a ? is typed and ready status is assumed again.

```
*YEAR-TO-DATE ACCOUNTING
IS THIS THE FIRST PAY PERIOD OF THE YEAR? (Y,N) N
MOST RECENT YEAR-TO-DATE FILE:YTDIF↙
NEW YEAR-TO-DATE OUTPUT FILE:YTDOF↙
```

This command performs cumulative year-to-date accounting by account number for numeric users and by user ID for alphanumeric users. The answer to the first question must by typed as Y for yes and N for no. If the answer is Y then the line that follows the question above is not typed. File names are in TENEX format. The YTD files produced are binary files. In order to produce a cost summary that can be printed out, use the command S.

*SUMMARY FOR YEAR-TO-DATE FILE: YTDF↙

Produces two ASCII versions of the file YTDF on OUTFILE. The two cost summaries produced are of the same type as those produced by the commands F and T.

*WRITE ONTO FILE: OUTFILE↙

This command is identical to the W command in ACCOUNTS. It is repeated in this program for extra convenience.

*EXIT

Transfers control to ACCOUNTS.

*ADJUST CHARGES

Transfers control to ADJUSTMENTS.

V. ADJUSTMENTS PROGRAM

Upon entering ADJUSTMENTS the program is in initial command status and it types out

```
:ADJUST

:
```

waiting for the initial command of an adjustment to be typed in. An adjustment is accomplished by entering a series of commands, one at a time, ending in the command B, which begins execution of the adjustment. After each command is typed and the appropriate parameters specified, the program returns to the ready status and types a colon. If a command or parameter value given causes the program to output an error message, that command is neglected and the program returns to ready status. However, if the error message is typed after the last command B, then no adjustment takes place and the program goes back to initial command status waiting for a new series of commands. If the adjustment was successfully executed the program types out *OK* and returns to initial command status. If a command appears more than once in a single adjustment, only the last appearance of that command and associated parameter values are recognized by the program. At any time before the command B is typed, typing ↑E returns you to initial command status without performing the adjustment just entered.

ADJUSTMENTS has two modes of operation: ADJUST mode and INSERT mode. When in initial command status the program is in ADJUST mode. In order to change to INSERT mode the initial command must be I. The appearance of the command I at any other time is illegal and will be neglected after an error message is typed out. After computing the adjustment (whether in ADJUST or INSERT mode) the program returns to ADJUST mode. INSERT mode is mainly used to enter adjustments for an account-user entry that is yet nonexistent for that pay period.

A single adjustment can affect one or more acct-user entries. If the account number and the user ID are both specified, then the adjustment affects the corresponding acct-user entry only. If the user ID is not specified, then the adjustment can affect all acct-user entries associated with the given account number. However, there are certain restrictions on the latter usage depending on the adjustments to be performed and whether one is in ADJUST or INSERT mode. Figure 2 shows the legal and illegal account-user ID combinations for each mode. Other restrictions will be noted with the commands below. It is clear from Fig. 2 that the only case in which the user ID does not have to be specified is if the account number is numeric and the program is in ADJUST mode. For that special case the adjustment affects all acct-user entries with that account number; C and P adjustments are divided among all associated entries proportional to the charges for each entry.

ACCT-USER COMBINATION		MODE	
ACCOUNT NO.	USER ID	ADJUST	INSERT[1]
N	N	legal	legal
N	A	legal	illegal
A	N	legal	illegal
A	A	legal	legal
N	-	legal	illegal
-	A	legal[2]	legal[3]
A	-	illegal	illegal
-	N	illegal	illegal

N = Numeric ; A = Alphanumeric

1. In INSERT mode if there are $ to be charged, then a Reason must be given.
2. Legal only with the command Delete
3. An alphanumeric account **** is automatically supplied by the program.

Fig. 2 Account-User Combinations Permissible Under the Two Modes.

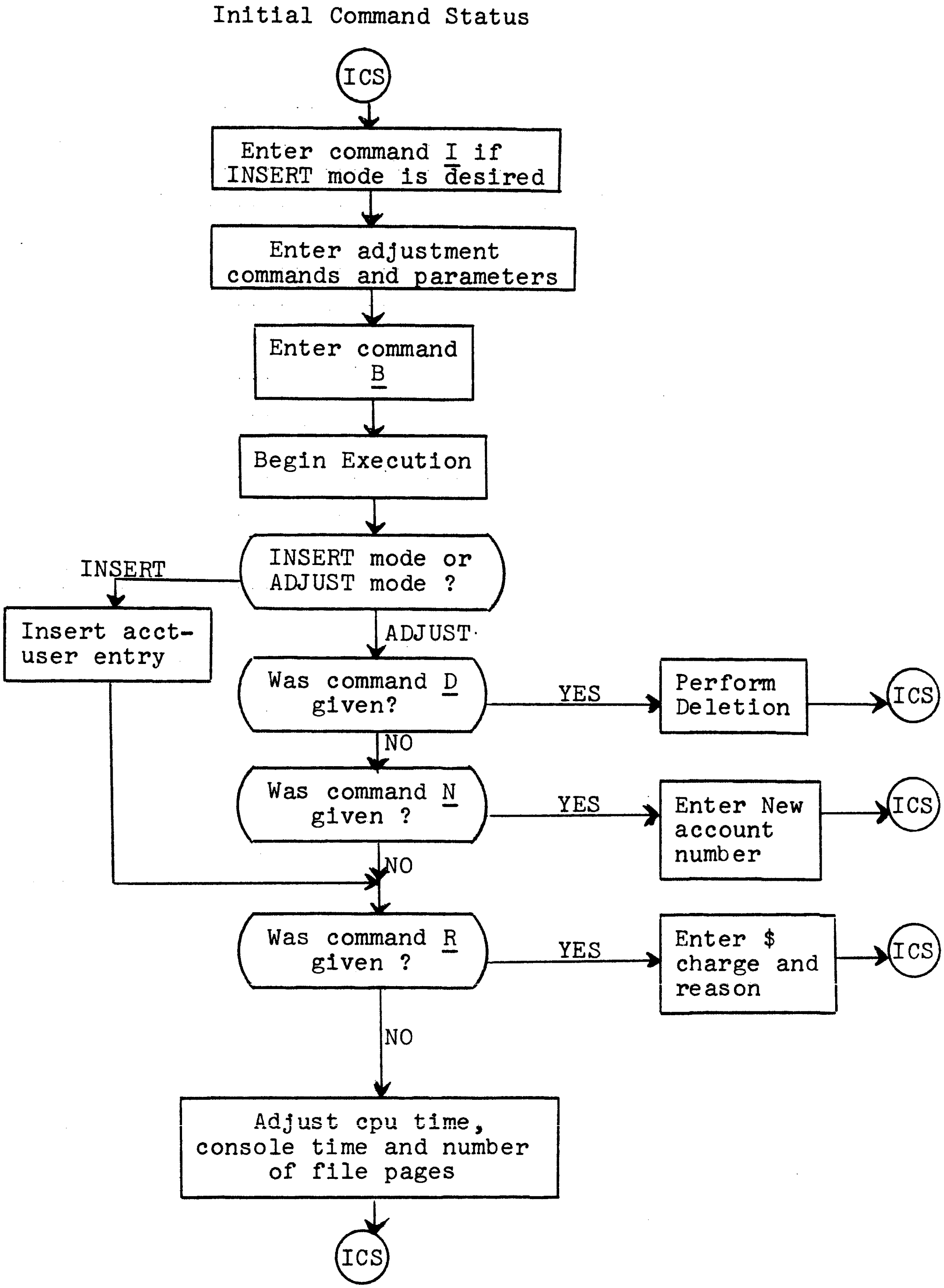


Fig. 3. Flow Chart for Entering and Execution of Commands in the Program ADJUSTMENTS. ICS = Initial Command Status

COMMANDS

Note: Except for commands I and B, all other commands can be entered in any order desired. Figure 3 is a flow chart for entering and execution of commands. Note the precedence of command execution.

:INSERT

Change from ADJUST to INSERT mode. This command must be given as an initial command if one desires to change to INSERT mode for that adjustment.

:BEGIN

Begin adjustment with current parameters. This should be typed after all parameters of the adjustment have been entered.

```
:ACCOUNT NUMBER (N,A) = N 345678↙
                   or   A ALPHA2↙
```

Sets the account number for that adjustment. N must be typed for a numeric account number and A for an alphanumeric account number. The space after N and A is supplied by the program.

:USER ID CODE: USERID↙

Sets the user ID for that adjustment. Typing ALT MODE performs recognition on the user ID. A carriage return must be typed after recognition.

```
:NEW ACCOUNT NUMBER (N,A) = N 134567↙
                       or   A NEW↙
```

Change all charges with account number and user ID specified to this new account number. If user ID was not specified the change applies to all acct-user entries with the specified account number. This command is legal only in ADJUST mode.

:DELETE

Delete the specified acct-user entry plus all associated special charges. If the specified account number is numeric and the user ID is not specified then all acct-user entries with that account number are deleted. If the user ID is specified and alphanumeric, and if the account number is not specified then all acct-user entries with that user ID are deleted.

:CPU TIME = -:3:14↙
CONSOLE TIME = 1:15:7↙

Sets the cpu time and console time adjustments. Enter time as hr:min:sec. The two colons as well as the number of seconds must be entered, but the hours and/or minutes may be omitted. If a negative time adjustment is desired then a minus sign should be entered first.

:PAGE ADJUSTMENT = 15↙

Adjusts the number of file pages by the given amount. The quantity typed can be either positive or negative but it must be an integer. Each page is equal to 512 words and each word can store up to 5 characters.

:FILE PAGE DISCOUNT = .92↙

Reads in a decimal which is used to multiply the number of file pages for the specified acct-user entry (entries). The initial setting is 1.Ø. (Make sure you note the difference between this command and the F command in ACCOUNTS. The F command here is equivalent to a page adjustment for a particular entry while the F command in ACCOUNTS affects the file charges for all acct-user entries without changing the number of pages.)

:REASON FOR CHARGE: HOME TELETYPE↙

Enters reason for the dollars to be charged as specified with the command $ below. The charge appears in the OTHER charges column in the accounting reports. The character string that is typed should not exceed 39 characters, not including the carriage return. Excess characters are neglected.

:$ TO BE CHARGED = 25.↙

Enters the extra dollars to be charged. The amount must be a decimal and may be negative. If the reason for the charge is given using the R command, then the amount is considered as OTHER charges. However, if the reason is not given then the adjustment is "hidden" by proportionately adjusting the cpu time and console time for the specified entry (entries). If, in the latter case, the adjustment results in negative cpu or console time, an error message is typed out and the adjustment is not performed.

:GO TO REPORTS

Transfers control to the program REPORTS. This command and the E command can be typed any time the program is in ready status. If any parameters had been specified they are neglected.

:EXIT

Transfers control to the program ACCOUNTS.

VI. How to Use ACCT1Ø

In order to use ACCT1Ø you must have the necessary privileged capability to read the FACT files from <TIMESHEET> or <ACCOUNTS>.

The reason the FACT files exist on two directories is to increase the probability that at least one copy of each file is free from disc errors. Normally, you should use the files in <TIMESHEET> because the on-going logging information is entered in <ACCOUNTS>. (After a FACT file is closed in <ACCOUNTS>, it is copied onto <TIMESHEET>.)

A normal procedure to use ACCT1Ø is the following:

a) CONNECT to TIMESHEET.

b) Type ACCT1Ø to start the program. The ready character < will appear on the teletype.

c) Type W and specify the output file for messages, etc. (the default output file is TTY:).

d) Type Z to initialize the system and set the pay period, etc.

e) Using the R command, read all FACT files in chronological order.

f) After all FACT files are read, type G to go to REPORTS. This has the effect of performing the accounting for jobs not logged out.

g) Type ↑C to get back to the EXEC. Then, SAVE core on a file. This will make sure that you have all the processed FACT file information saved away, and further errors will not necessitate rereading of the FACT files.

h) Type CONTINUE↙ or START↙ and go ahead with reports and adjustments. Note that STARTing closes all input and output files and opens TTY: as the output file.

Figure 4 shows an example of an actual usage of ACCT1Ø.

```
 ↑C
@CONNECT (TO DIRECTORY) TIMESHEET (PASSWORD)
@ACCT1Ø


<WRITE ONTO FILE: MESSAGES [NEW FILE]

<Z INITIALIZE ACCOUNTING SYSTEM
 PAY PERIOD IS:  DEC 15,1971 TO DEC 18,1971

  ACCOUNTING INFORMATION?(Y,N)N
  LOGOUT DATA FILE?(Y,N) N

<READ FACT FILE: 15-DEC-71/18ØØ.;1

<READ FACT FILE: 16-DEC-71/18Ø2.;1

<READ FACT FILE: 17-DEC-71/1814.;1

<GO TO REPORTS
CLASSIFICATION OF USERS AND ACCOUNTS, ETC.


*WRITE ONTO FILE: TTY: [CONFIRM]

*↑C
@SAVE Ø (TO) C\C777777 (ON) RAWDATA [NEW FILE]
@START


<GO TO REPORTS


*WRITE ONTO FILE: TIMESHEET [NEW FILE]

*DO ALL COST SUMMARIES
OST SUMMARY
INAL COST SUMMARY FOR ACCOUNTING
OTAL COST
OB SUMMARIES

*WRITE ONTO FILE: TTY: [CONFIRM]

*↑C
@COPY MESSAGES.;1 (TO) LPT: [OK]
[LPT: BUSY-GO]
@COPY TIMESHEET.;1 (TO) LPT: [OK]
```

Figure 4. Example of Usage of ACCT1Ø.

APPENDIX

A. FACT FILE FORMATS

A FACT file consists of a sequence of 36-bit word blocks. The blocks are variable in length depending on the information contained, e.g. login, logout, etc. There are 10 types of word blocks, shown in Fig. A-1. Associated with each block is a code which denotes the type of information contained in the block, and the size of the block, which is the total number of words in the block. The code and size of the block are in the first word of the block as shown in Fig. A-1.

Figures A-2 through A-5 show the detailed formats for the different types of word blocks. An explanation for some of the terms that are used follows:

TSS JOB #: is the job number assigned to a particular running job by the system. It is assigned to the job upon login and serves to uniquely identify that job as far as the system is concerned. This association is terminated upon logout. (TSS Job # Ø is normally assigned to the system itself.)

TTY #: is the TTY channel number to which the user is connected.

DIRECTORY #: is simply the user ID code that identifies the user. Directory # and user ID will be used synonymously.

ACCOUNT NUMBER: is a "number", entered by the user upon login, to which the user wishes to make charges until either the account number is changed, in which case new charges are made to the new account, or the job is terminated. Disc file charges are made to the account number which was current when the file was created.

Account numbers are either numeric or alphanumeric. (Numeric users use numeric accounts and alphanumeric users use alphanumeric accounts.) Numeric accounts are represented by a 33-bit integer, bits 3-35, with zeroes in bits Ø-2. Alphanumeric accounts are represented by an alphanumeric string that can have a maximum of 39 characters. In the FACT file, the negative of the number of characters in the account string is given followed by the account string itself written in 7-bit ASCII code, packed 5 characters per word with bit 35 containing zero.

DATE AND TIME OF DAY - This represents the date and time of day the entry was made into the FACT file, written in TENEX internal format. (See TENEX Date and Time

Standards in the Introduction to the JSYS Manual.) The date and time of day can be reset by the operator. This step is necessary if the system is restarted after a crash. A corresponding entry TIME SET is entered into the FACT file. A date of -1 indicates that the system does not know the date; this happens usually immediately after a crash.

RUNTIME AND CONSOLE TIME - Wherever entered in the FACT file, they are in milliseconds. This means that in a 36-bit word (with BØ=Ø) one can represent a maximum of 397 days + 16 hrs. + 22 min. + 18.36 sec.

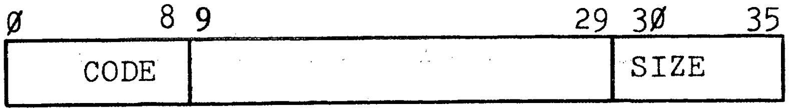
CONSOLE ATTACH AND DETACH - Although these entries are made into the FACT file upon ATTACHing or DETACHing a teletype console, they are completely neglected by the current accounting program.

CHKPNT - CHKPNT entries are entered in the FACT file by running the subsystem program CHKPNT.SAV. (You must be ENABLEd in order to run this program.) One entry per active job is made.

SYSTEM STARTED FROM SCRATCH - This entry is made automatically every time the system is restarted from scratch. Note that there is no way to tell exactly when a crash occurs. (The variable NCRASH in ACCT1Ø contains the number of system restarts. The variable UPTIME contains the approximate time in seconds when the system has been up.)

START OF DISC UTILIZATION STATISTICS - This entry immediatly precedes all disc utilization entries. These entries are also made by the subsystem CHKNPT.SAV. (The variable NFLCK in ACCT1Ø contains the number of file checks. The total number of file pages for the pay period is divided by NFLCK to find the average file usage for every acct-user entry.)

DISC UTILIZATION ENTRY - There is one such entry for every acct-user combination that has any files stored on the disc. The total number of file pages for each acct-user is recorded. One page contains 512 words and each word is considered to contain 5 characters.



CODE	SIZE	BLOCK TYPE
5Ø1	5 + K	Login
5Ø2	5 + K	Change Account Number
141	5	Logout
142	3	Console Attach
143	3	Console Detach
2Ø1	5	Checkpoint
74Ø	1	System Started from Scratch
741	3	Time Set
54Ø	1	Start of Disc Utilization Statistics
6Ø1	5 + K	Disc Utilization Entry

a) K = Ø for numeric accounts

b) 1 ≤ K ≤ 8 for alphanumeric accounts;
The account string is stored in K words at 5 characters per word.

Figure A-1. Types of FACT File Word Blocks

LOGIN

Ø 8	9 17	18 29	3Ø 35
5Ø1	TSS JOB #	TTY #	5 + K
Ø		DIRECTORY #	
DATE		TIME OF DAY	
Ø			

if ≥ Ø Numeric Account Number
if < Ø -(# Chars in Alphanumeric Acct.)			

Alphanumeric Account String K words (K = Ø for numeric accounts)			

CHANGE ACCOUNT NUMBER

Ø 8	9 17	18 29	3Ø 35	
5Ø2	TSS JOB #	TTY #	5 + K	
Ø		DIRECTORY #		
DATE		TIME OF DAY		
RUNTIME (msec) (old acct #)				

if ≥ Ø Numeric Account Number
if < Ø -(# Chars in Alphanumeric Acct.)
				New Account

Alphanumeric Account String K words (K = Ø for numeric accounts)				

chars = characters

Figure A-2

LOGOUT

Ø 8	9 17	18 29	3Ø 35
141	TSS JOB #	TTY #	5
Ø		DIRECTORY #	
DATE		TIME OF DAY	
RUNTIME (msec)			
CONSOLE TIME (msec)			

CONSOLE ATTACH

142	TSS JOB #	TTY #	3
Ø		DIRECTORY #	
DATE		TIME OF DAY	

CONSOLE DETACH

143	TSS JOB #	TTY #	3
Ø		DIRECTORY #	
DATE		TIME OF DAY	

Figure A-3

CHKPNT (Checkpoint)

Ø 8	9 17	18 29	3Ø 35
2Ø1	TSS JOB #	TTY #	5
Ø		DIRECTORY #	
DATE		TIME	
RUNTIME (msec)			
Ø			

SYSTEM STARTED FROM SCRATCH

Ø 8	9 29	3Ø 35
74Ø	Ø	1

TIME SET

Ø 8	9 17	18 29	3Ø 35
741	TSS JOB #	TTY #	3
Ø		DIRECTORY #	
NEW DATE		NEW TIME	

Figure A-4

START OF DISC UTILIZATION STATISTICS

Ø 8	9 29	3Ø 35
54Ø	Ø	1

DISC UTILIZATION ENTRY

<table>
<tr><th>Ø 8</th><th>9 17</th><th>18 29</th><th>3Ø 35</th></tr>
<tr><td>6Ø1</td><td>Ø</td><td>Ø</td><td>5 + K</td></tr>
<tr><td colspan="2">Ø</td><td colspan="2">DIRECTORY #</td></tr>
<tr><td colspan="2">DATE</td><td colspan="2">TIME OF DAY</td></tr>
<tr><td colspan="4">Number of File Pages Used</td></tr>
<tr><td colspan="4">if ≥ Ø Numeric Account Number
if &lt; Ø -(# Chars in Alphanumeric Acct.)</td></tr>
<tr><td colspan="4">Alphanumeric Account String
K words
(K = Ø for numeric accounts)</td></tr>
</table>

Figure A-5

B. DETAILED STRUCTURE OF ACCT1Ø

There are two source files which comprise the accounting program. They can be found under directory <SOURCES> as ACCT1Ø.F4 and ACCT1Ø.MAC. ACCT1Ø.MAC is written in MACRO-1Ø and contains a number of subroutines which are called by the main program. ACCT1Ø.F4 is written in FORTRAN 4 and consists of a main program, a DATA subprogram, and two subroutines RFACT and ACCT.

The DATA subprogram defines the contents of several symbols that are placed in COMMON and are used by the rest of the program. All symbols that appear in DATA statements never have their contents changed.

RFACT reads the binary FACT files word by word, interprets the different types of word blocks and calls upon ACCT to compile time and file charges. Certain inconsistencies in the FACT files are detected by RFACT and appropriate messages are written onto the currently open output file.

ACCT is responsible for inserting new acct-user entries, user ID entries and account number entries, and compiling time and file charges for those entries.

Information concerning users, account numbers and charges is arranged in several arrays. Following is a description of the structure of those arrays.

ACTNM(I) Contains a list of all numeric account numbers as obtained from the FACT files. The symbol NNACT contains the number of those numeric accounts. A single account number is stored per word. The first free entry in the array contains -1, i.e. ACTNM(NNACT+1) = -1.

ACTAL(I) Contains a list of the alphanumeric account number strings. Each account number occupies from 2 to 9 words of storage, depending on the length of the account string. The first word is a header which contains the number of characters in the account string and also the number of words it occupies (see Fig. B-1). The position of the header in the array for a particular account number identifies that account to the rest of the program. For example, if ACTAL(63) contains the header for alphanumeric account ALPHA, then the number 63 identifies the position of ALPHA in ACTAL(I).

ID(I), ACNT(I) These two arrays together constitute what is known as the acct-user entries. ID(I) contains a list of user ID's and pointers to ACNT(I) where account numbers and compiled charges are recorded by the subroutine ACCT.

Associated with every user is a 1-word user entry in ID(I) and a number of 5-word entries in ACNT(I), one entry for each account number the user had used. NID contains the number of users and NENTRY contains the number of account entry blocks in ACNT(I). Fig. B-2 shows an example of the structure of entries for a user with three account numbers.

ID(I) Format:

$$\text{BØ} = \begin{cases} \text{Ø; numeric user} \\ \text{1; alphanumeric user} \end{cases}$$

B1-17 Pointer to the last account entry for that user.

B18-35 User ID (Directory #)

ID(I) = Ø → first free entry

Account Entry Format in ACNT(I)

The account entry block consists of 5 words:

Word 1 (NACCT)

$$\text{BØ} = \begin{cases} \text{Ø; numeric account} \\ \text{1; alphanumeric account} \end{cases}$$

$$\text{B1-2} = \begin{cases} \text{ØØ; Free (no charges)} \\ \text{1Ø; Commercial rates} \\ \text{Ø1; Government rates} \end{cases}$$

B3-35 Account number for numeric accounts or a pointer to ACTAL(I) for alphanumeric accounts

Bits 1-2 tell which rates to apply to that account number. These two bits are appropriately set after all FACT files had been read; they contain zero before that. If Word 1 = 677777,,777777 = NBIT2, then that 5-word block is considered "empty". This occurs, for example, when an account number is deleted during ADJUSTMENTS. The variable EMPTY is equal to the number of such empty blocks in ACNT(I). Empty blocks can be reused when new acct-user entries are inserted during ADJUSTMENTS.

The first word of the very first word block in ACNT(I) is ACNT(2). ACNT(1) points to the first free entry.

Word 2:

BØ = { 1; indicates first account entry for a user
Ø; otherwise

B1-17= { Ø; no special charges for this account number,
otherwise,
special charges exist in SP(I)

B18-35 { if BØ=Ø, points to next account entry for that user
if BØ=1, contains zero

Word 3: CPU time charged to that acct-user entry in msec.

Word 4: Console time charged in msec.

Word 5: Number of file pages charged.

SP(I) Contains all special charges as entered using commands $ and R in ADJUSTMENTS. Fig. B-3 shows the format for a special charges entry. The format for the account number is identical to that described above for ACNT(I).

LOG(I) This array is used in RFACT to keep track of jobs that are logged in. Every active job has a 5-word block entry in LOG(I) (see Fig. B-4). If a CHKPNT was made while a job was active, then words 4 and 5 contain the CHKPNT date and time and the runtime for that job when the CHKPNT was made. When a job is logged out, the corresponding entry in LOG(I) becomes free (the first word is put equal to -1) and can be reused. The format for the account number is identical to that in ACNT(I).
The array LOG(I) is also used to alphabetize users in the REPORTS program.

READING OF FACT FILES BY RFACT

The binary FACT files are read word by word and interpreted a block at a time. (Note that the block size depends on the type of information it contains.) The program makes a check on the format of each block. In case of an error, the message 'FACT FILE ERROR' is written onto the currently open output file along with the word causing the error and the location of that word in the FACT file (both numbers are written in octal).

As the word blocks are read they are classified into one of the 1Ø possible entries shown in Fig. A-1, after which appropriate action is taken as shown in the flow charts of Figs. B-5 through B-12. (ATTACH and DETACH entries are simply bypassed by the program.) In the flow charts, each box that contains a capitalized and quoted message indicates that the message is written onto the output file. Along with the message is written the most recent date and time of day as read from the FACT file, both in TENEX internal format and local date and time.

Ø 17	18 35
# Chars in String	K
Alphanumeric Account String K words	

K=Ø; first free entry

Fig. B-1 Alphanumeric Account Entry Block in ACTAL(I)

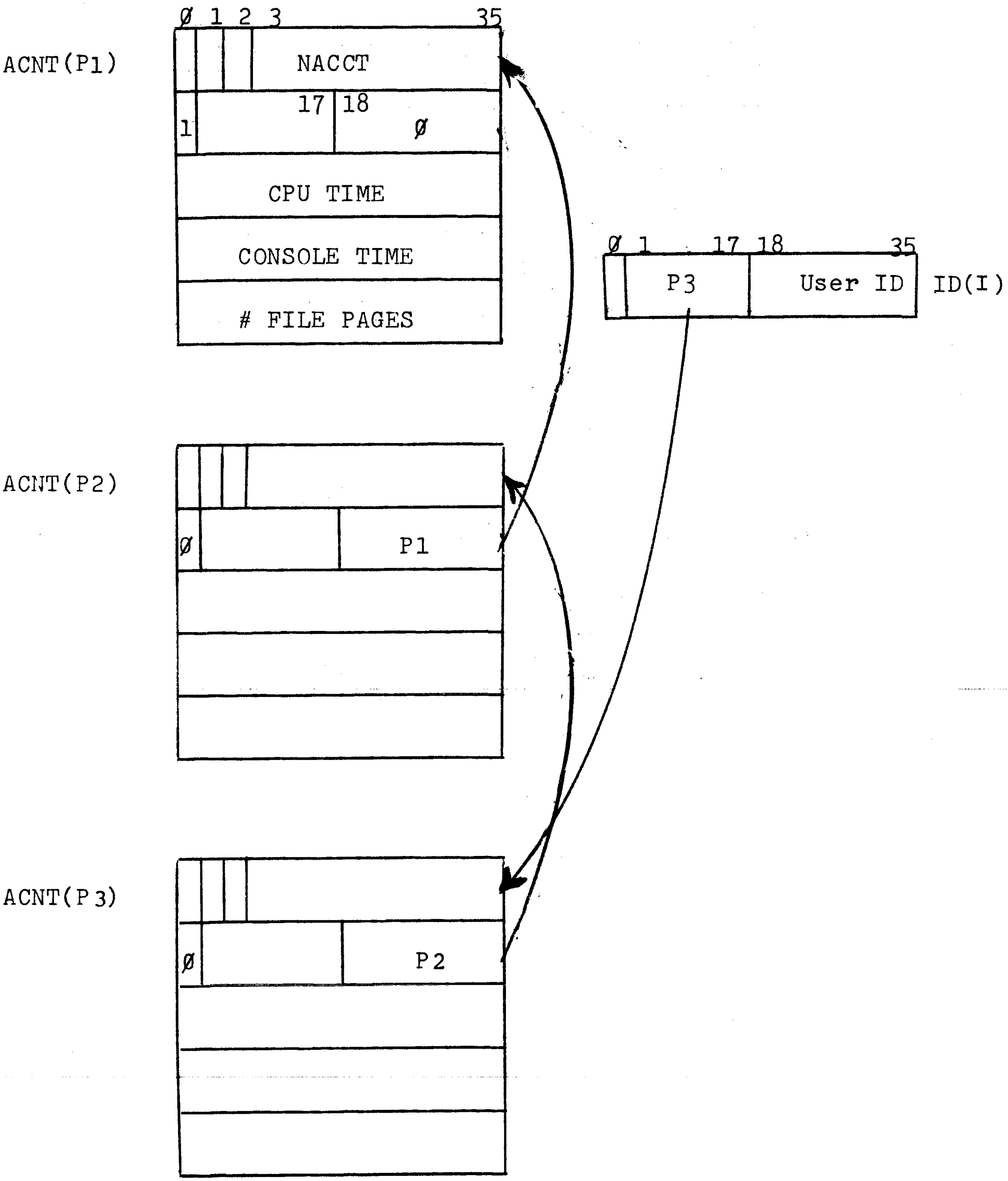
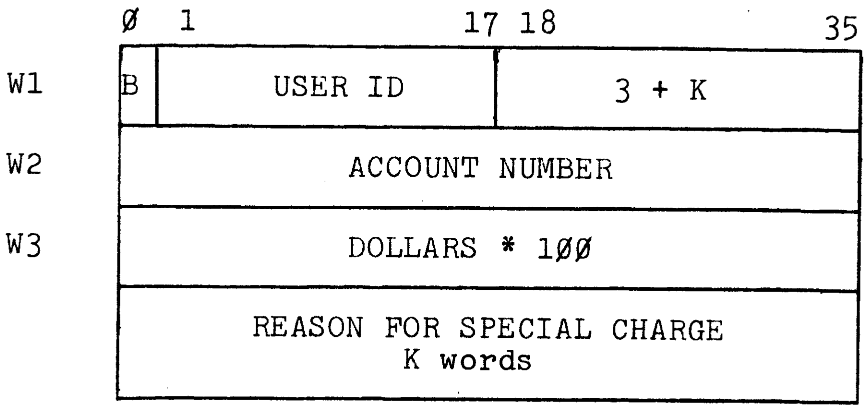


Figure B-2 Structure of Acct-User Entries



B = { Ø; valid entry
1; empty (non-valid) entry

W1 = Ø → first free entry

Figure B-3 Special Charges Format

	Ø 8	9 17	18 35
W1	Ø	TSS JOB #	USER ID
W2	ACCOUNT NUMBER		
W3	LOGIN DATE		LOGIN TIME
W4	CHKPNT DATE		CHKPNT TIME
W5	RUNTIME (msec)		

W1 = -1 ; free entry
W1 = -2 ; first free entry after the last used entry

W4 = -1 ; no CHKPNT entry, W5 is meaningless

Figure B-4 Format for Active Job in LOG(I)

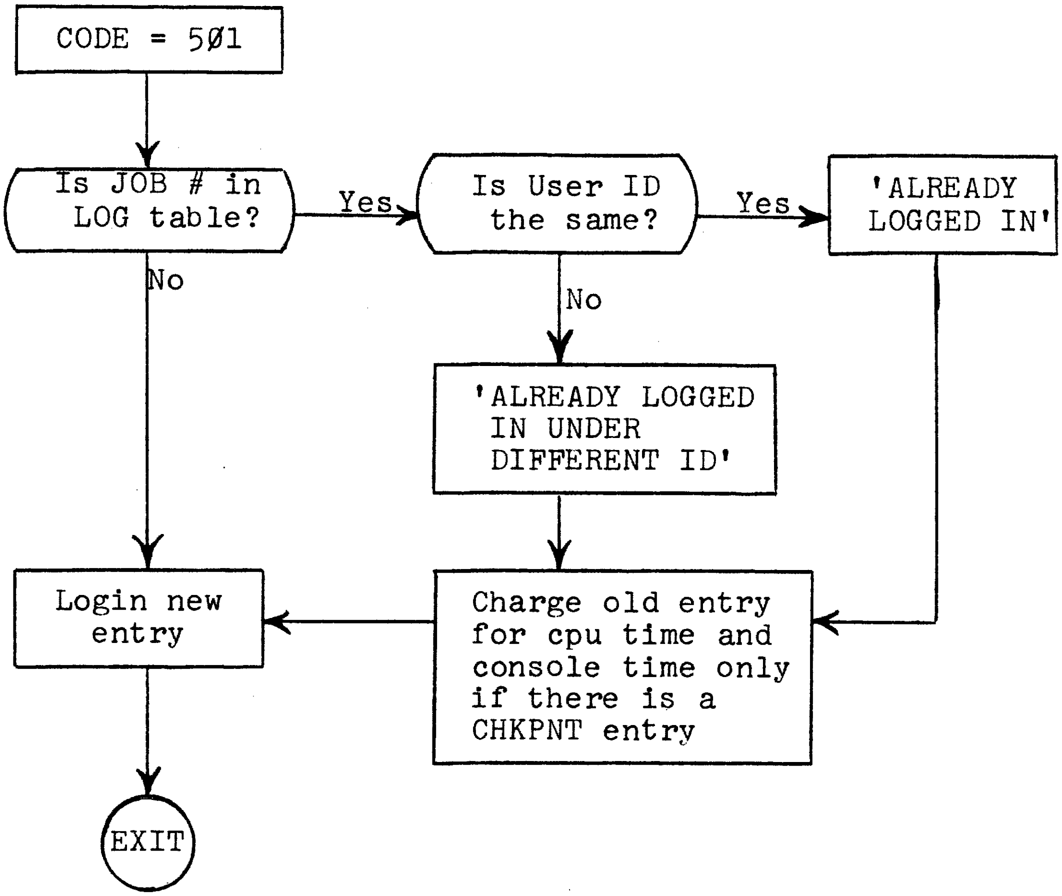

CODE = 5Ø1
Is JOB # in LOG table?
Yes
Is User ID the same?
Yes
'ALREADY LOGGED IN'
No
No
'ALREADY LOGGED IN UNDER DIFFERENT ID'
Login new entry
Charge old entry for cpu time and console time only if there is a CHKPNT entry
EXIT

Figure B-5 Flow Chart for login

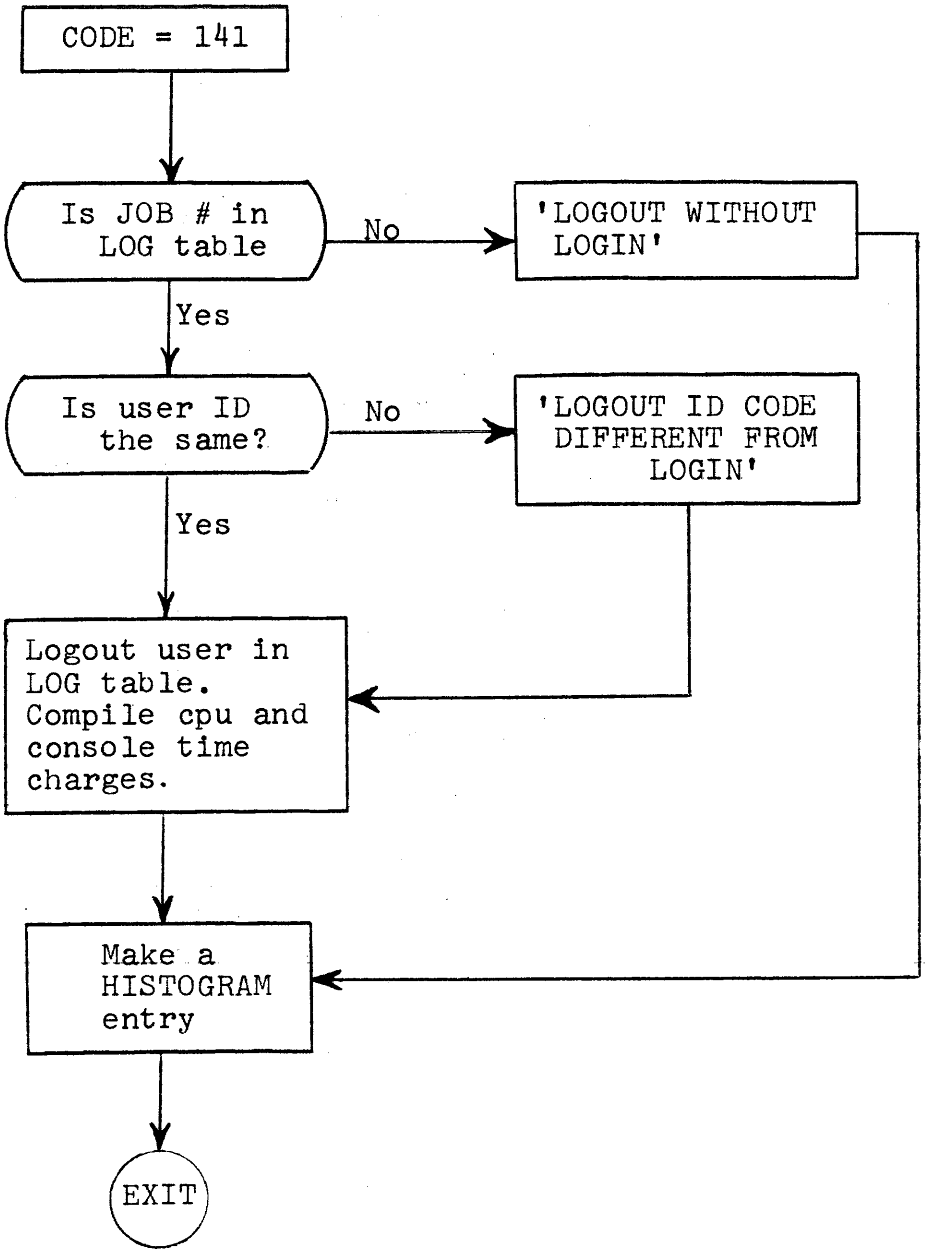


Figure B-6 Flow chart for logout

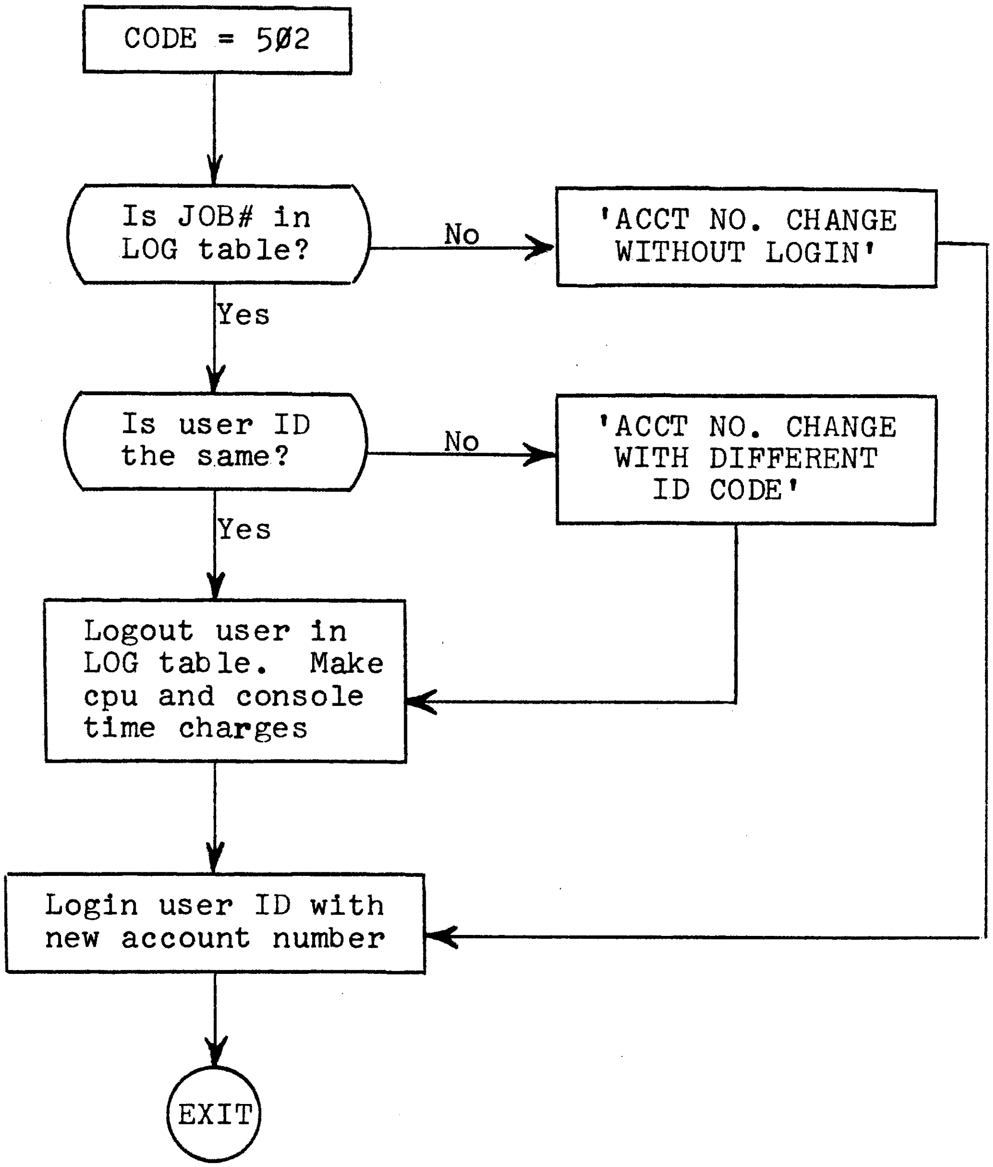

CODE = 5Ø2
Is JOB# in LOG table?
No
'ACCT NO. CHANGE WITHOUT LOGIN'
Yes
Is user ID the same?
No
'ACCT NO. CHANGE WITH DIFFERENT ID CODE'
Yes
Logout user in LOG table. Make cpu and console time charges
Login user ID with new account number
EXIT

Figure B-7 Flow chart for account number change

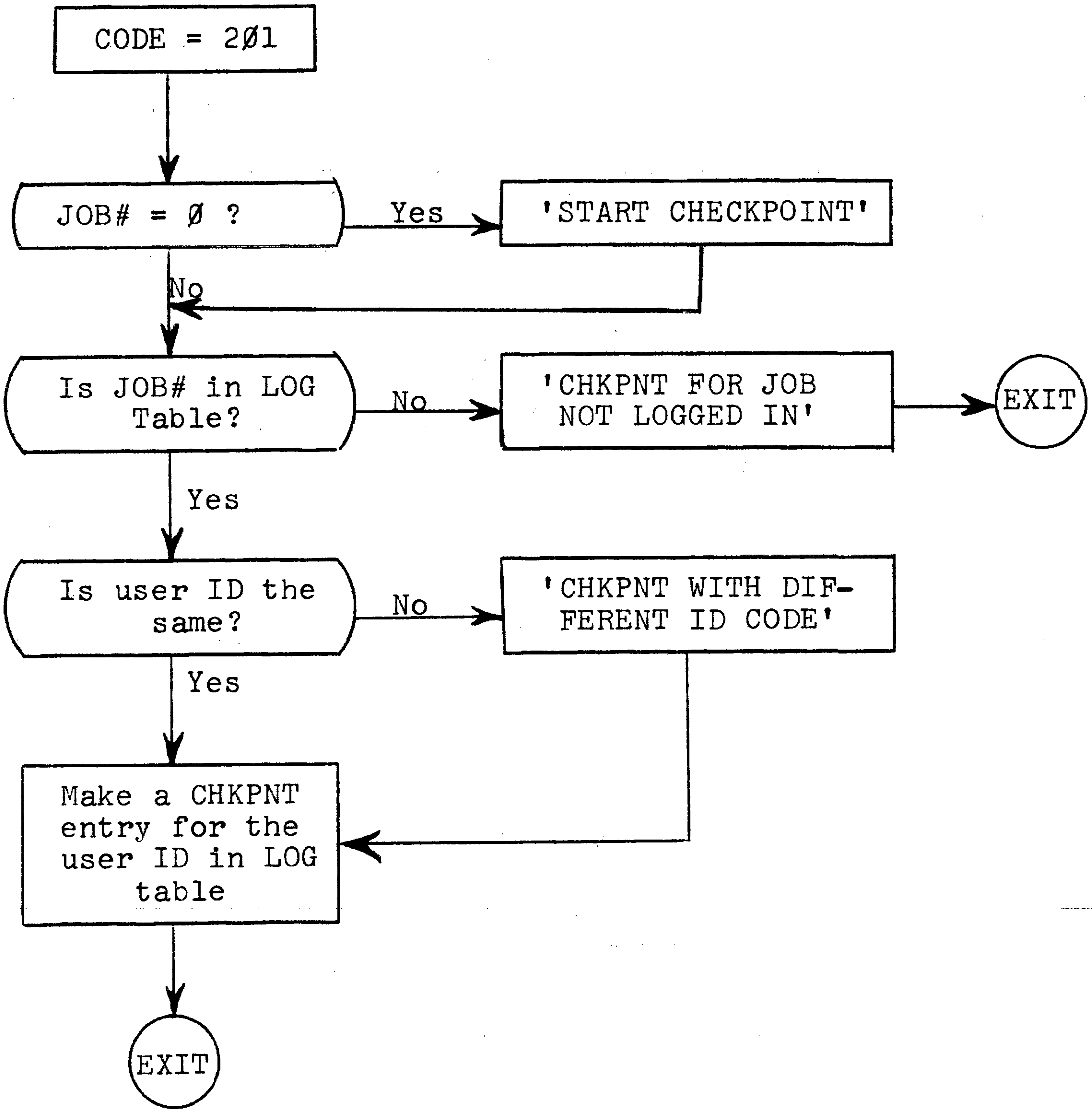

CODE = 2Ø1
JOB# = Ø ?
Yes
'START CHECKPOINT'
No
Is JOB# in LOG Table?
No
'CHKPNT FOR JOB NOT LOGGED IN'
EXIT
Yes
Is user ID the same?
No
'CHKPNT WITH DIFFERENT ID CODE'
Yes
Make a CHKPNT entry for the user ID in LOG table
EXIT

Figure B-8 Flow chart for CHKPNT

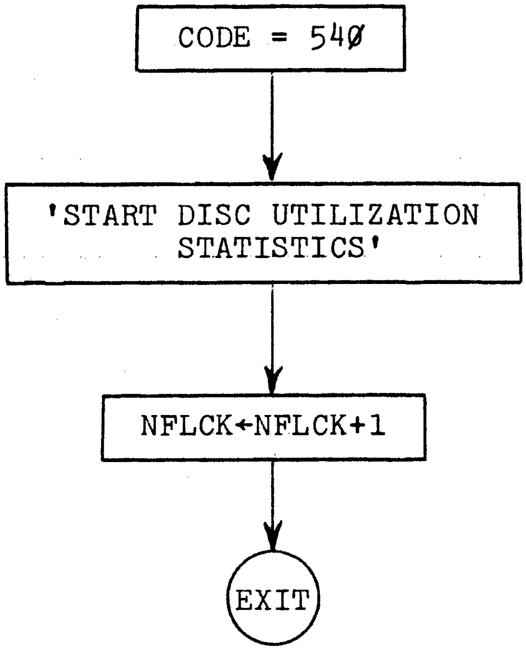


Figure B-9 Flow chart for start of disc utilization statistics

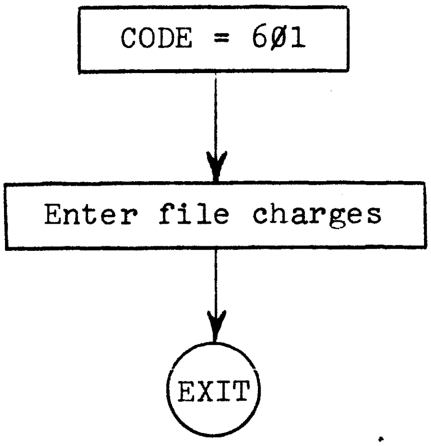


Figure B-1Ø Flow chart for disc utilization entry

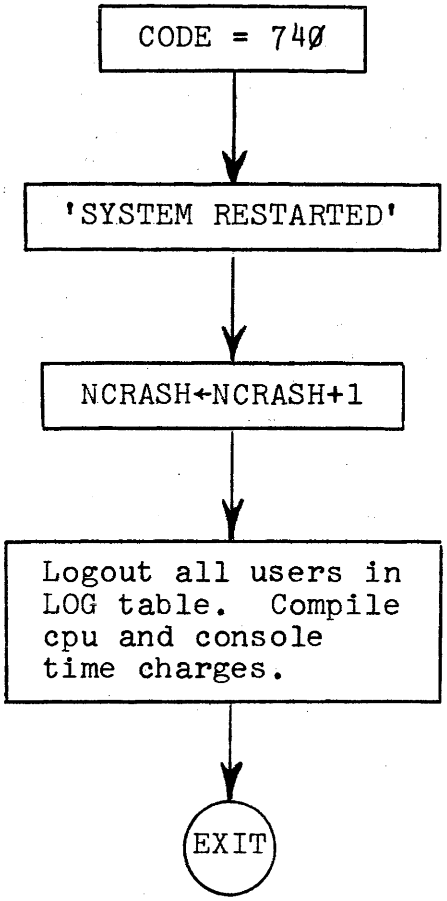

CODE = 74Ø
'SYSTEM RESTARTED'
NCRASH←NCRASH+1
Logout all users in LOG table. Compile cpu and console time charges.
EXIT

Figure B-11 Flow chart for system started from scratch

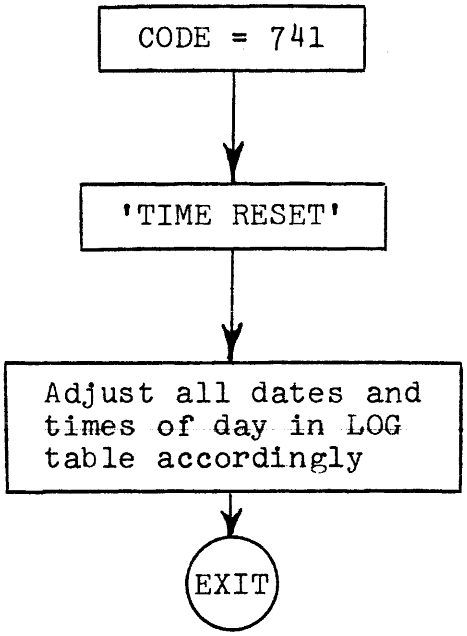


Figure B-12 Flow chart for time set

C. YEAR-TO-DATE FILE FORMAT

The first six words contain the pay period as an ASCII string. The remainder of the file consists of 6-word blocks whose format is shown in Fig. C-1. The first word of each block contains the account number for numeric accounts or the negative of the user ID for alphanumeric accounts.

Account Number or - (USER ID)
CPU TIME (SEC)
CONSOLE TIME (SEC)
No. OF FILE PAGES
OTHER CHARGES * 100
TOTAL CHARGES * 100

Figure C-1 Year-to-Date File Format

D. HISTOGRAM

A histogram of cpu time and console time used per session for the pay period is produced by using the command H in ACCOUNTS. This section gives the details concerning how the histogram is produced and what is the format of the file on which it is written.

As the FACT files are read by RFACT, an account is made of how much cpu and console time are used by every job that gets logged out. The job is then classified and a 1 is added to the appropriate array element in HIST(I,J). An explanation of how this classification is made follows.

The cpu time axis and the console time axis are each divided into a number of time slots. The time slots for each axis are sequentially numbered. Let I represent the cpu time slots and J the console time slots. Then, any part of cpu and console time values can be classified into a two-dimensional slot (I,J). Let HIST(I,J) represent the number of user sessions whose cpu and console time values fall in the slot (I,J). Then, HIST(I,J) is a histogram of cpu and console times for the pay period. There remains the explanation of how the cpu and console time axes are divided into slots.

Let t represent time on one of the axes, and let t(i) correspond to the time value at the ith division:

$$t(i) = (i \bmod N)\, 2^{\left[\frac{i}{N}\right]} T + \left(2^{\left[\frac{i}{N}\right]} -1\right)NT, \quad i=0,1,2,\ldots$$

where T has the units of time and N is an integer, and $\left[\frac{i}{N}\right] \equiv$ integer portion of $\frac{i}{N}$.

Define the ith slot as the time portion between t(i-1) and t(i), then:

$$t(i)-t(i-1) = 2^{\left[\frac{i-1}{N}\right]} T \quad , \; i=1,2,\ldots$$

In order to have a finite number of slots on each axis we let the last slot represent all times above a certain maximum t. Let that maximum t be $t(i_m)$ such that:

$t(i_m) = (2^M-1)NT$
where $i_m = NM$
and M is some integer.

Then the total number of time slots is equal to NM+1.

The above shows that the division of the time axis is completely specified by three values: T, N and M.

Let T=T1, N=N1 and M=M1 for the cpu time axis,
and T=T2, N=N2 and M=M2 for the console time axis.
Then, Imax = N1*M1+1
and Jmax = N2*M2+1
where Imax and Jmax are the maximum dimensions for I and J in HIST(I,J).

The values for T1, N1, M1, T2, N2 and M2 are defined in a DATA statement in RFACT, where T1 is in seconds and T2 is in minutes. These values can be changed, if desired, by appropriately changing them in the DATA statement, or by using DDT.

The histogram is written on an output file as a series of numbers (represented in ASCII), each number followed by a comma. The numbers are written in the following order:

```
T1,N1,M1,T2,N2,M2,Imax,Jmax,
HIST(1,1),HIST(1,2),...,HIST(1,Jmax),
HIST(2,1),HIST(2,2),...,HIST(2,Jmax),
.
.
.
HIST(Imax,1),HIST(Imax,2),...,HIST(Imax,Jmax),
```

Histograms from different pay periods can then be processed by a separate program.

E. HOW TO MODIFY ACCT1Ø

This section describes some of the modifications in ACCT1Ø that may be necessary in order to accomodate the accounting system to your needs.

1. Array Dimensions - The arrays of interest here are those whose dimensions depend, for example, on the possible number of simultaneous users, the total number of users, the number of numeric and alphanumeric accounts, etc. Fig. E-1 shows a list of those arrays and the program(s) in which they are dimensioned. MAIN. refers to the main program ACCT1Ø. Since those arrays that appear in more than one program are in COMMON, a dimension change for an array must be performed in both programs in which the array is dimensioned. You will find that most of the arrays have been generously dimensioned and there will be no need for modifications, at least initially.

ARRAY	DIMENSIONED IN
ID(I)	MAIN., ACCT
ACNT(I)	MAIN., ACCT
ACTNM(I)	MAIN., ACCT
ACTAL(I)	MAIN., ACCT
LOG(I)	MAIN., RFACT
SP(I)	MAIN.
HIST(I,J)	RFACT

Figure E-1 Arrays whose dimensions can be changed to reflect the needs of the system.

2. Charge Rates - The accounting system is designed to accomodate three charge rates, one of which is free (no charges). The other two rates are known as: government rates and commercial rates. (Needless to say, you can give the two rates any names you wish.) These rates are defined in a DATA statement in the MAIN. program. The symbols used are:

RATE TYPE	CPU RATE	CONSOLE RATE	FILE RATE
Government	GRCPU	GRCON	GRCHAR
Commercial	CRCPU	CRCON	CRCHAR
	($/MIN)	($/HR)	($/PAGE/PAY PERIOD) 1 PAGE = 5*512 CHARACTERS

The charge rates can simply be changed by appropriately changing the entries in the corresponding DATA statements. Make sure that the numbers you place are decimal (i.e. with a decimal point).

3. Accounts and Rates - For numeric users, each account number is assigned one of the three rates. For alphanumeric users, the rates can be assigned according to the user as well as the account number. These assignments are made in a portion of the MAIN. program called (CLASS) starting at statement number 12ØØ. Transfer to (CLASS) is made whenever account classification is needed. (See, for example, the portion of the program starting at 22Ø.) The two variables that (CLASS) needs for classification is NACCT, the account number, and USER, the user ID. The format for NACCT is given in Section B of this Appendix. Bits 1 and 2 in NACCT are set to reflect the proper charge rate assignment.

The only portion of the program that need be changed is (CLASS) starting at 12ØØ.

4. Cost Summary by Category - The command C in REPORTS produces a cost summary by category. This categorization is a subclassification of account numbers. For example, in this program, free accounts are subclassified into non-chargeable and computer center accounts. The portion of the MAIN. program that executes the C command is located between statement numbers 26Ø and 297+. In particular, the subclassification code itself starts at 296. You may wish to change any or all of this portion of the program to suit your needs.

CHECKPOINT

Program to Check Active Jobs

```
TENEX CHECKPOINT V001 -
$H$ (Lists the following)
TYPE C,L,M,N, OR E FOR THE FOLLOWING ACTIONS:
C - Adds one checkpoint to fact file
L - Lists all fact file extensions and versions
M - Enters a continued checkpoint loop (every N minutes)
N - Renames current fact file to FACT.TIMEDATE and
    takes disc statistics (optional)
E - Exit checkpoint program
$
```

I. TO RUN CHECKPOINT

```
LOGIN (WHEEL OR OPERATOR)

@ENABLE
!CHKPNT.SAV;1
```

$At this point if you type ? the program types out all the commands available and tells you what they stand for.

```
TENEX checkpoint V002 -15 July 1970

C$HECKPOINT (all active jobs)

$E$XIT

!DISABLE

@LOGOUT
```

If the system is crashing a lot, checkpoint should be run every hour on the hour.

II. TO RUN THE EVENING ACCOUNTING WITH DISC STATISTICS

```
LOGIN (wheel or operator)

@ENABLE

!CHKPNT.SAV;1

TENEX Checkpoint V001 - 15 July 1970

$C$HECKPOINT, (ALL ACTIVE JOBS)

$N$EW (renames current fact file)

FACT

FACT.  18 August 1970/1832; 1 NEW FILE

Disc statistics?  (y or N) y

Listing device LPT:   [OK]
```

DELD

Reclaims storages on disc by returning to free status all pages in deleted files. Does "EXPUNGE" (DELOF) on all directories. Run daily by operators. Requires operator capability.

@DELD

(prints each directory expunged)

NOTIFY

Sends message to all teletypes. Used by operators to send notification of system going down, etc. Requires wheel or operator capability.

@NOTIFY

TTYS. -1 for all TTYs, or specific line number, e.g. 40,41,42

MESSAGE: Text up to 1000 characters, terminates with Control-D (↑D).

Prints message on all lines simultaneously, but won't print if line is already outputting. Prints numbers of lines which haven't received message. Returns to EXEC when all lines are completed.

SETMRP

This is a privileged program (operators only). It sets a minimum run percentage for a job, i.e., the system will attempt to keep the specified job at the specified percentage (CPU time/real time) regardless of system load.

ULIST

Lists the USER DIRECTORY and user information in it.

The person responsible for ULIST is Ray Tomlinson, BBN ext 363.

The program operates only for system personnel with wheel or operator status.

ULIST first asks:

```
     PRINT PASSWORDS (Y OR N)?
```

then

```
     LIST USER INDEX ON FILE:
```

The information printed includes each directory name, password (optional), the max # of disc pages used for permanent storage, his privileges, his user type, special resources permission, directory number, default file protection, directory protection, and default file versions retained.

This output is formatted for wide line printer paper.

DISCUSE

Disc file utilization measurement. Types out number of 512 - word pages unused on the disc, followed by the number of pages which are in use.